Contents

- Azure Synapse Analytics
- Switch to dedicated SQL pool (formerly SQL DW) >
- Overview
  - What is Azure Synapse?
  - What's new?
  - Terminology
  - FAQ
- Quickstarts
  - Get started
  - 1 Create a Synapse workspace
  - 2 Analyze using serverless SQL pool
  - 3 Analyze using a Data Explorer pool
  - 4 Analyze using a serverless Spark pool
  - 5 Analyze using a dedicated SQL pool
  - 6 Analyze data in a storage account
  - 7 Integrate with pipelines
  - 8 Visualize with Power BI
  - 9 Monitor
  - 10 Explore the Knowledge center
  - 11 Add an administrator
- Workspace
  - Quickstarts
    - Create a workspace
      - Azure portal
      - Azure CLI
      - Azure PowerShell
      - ARM template
    - Link a Power BI workspace
    - Read ADLS Gen2 into Pandas dataframe

- Connect to Azure Data Explorer
- Partner solutions
- Concepts
  - Shared metadata management
    - Overview
    - Metadata objects
      - Shared databases
      - Shared tables
    - Code artifacts
      - Apache Spark Notebooks
      - SQL script
      - Data Flow
  - Maintenance schedule
  - Source control
  - Continuous integration and delivery
  - Security
    - Security baseline
    - Security Controls by Azure Policy
    - Encryption
    - Network Security
      - Synapse workspace IP firewall rules
      - Connectivity settings
      - Synapse workspace Managed virtual network
      - Synapse workspace Managed private endpoints
      - Synapse workspace data exfiltration protection
      - Synapse workspace with private link hubs
      - Connect to a secured storage account
    - Authentication and pass-through
      - Azure Active Directory
      - Multi-factor authentication
      - SQL authorization
      - Conditional Access

- Access control
  - Overview
  - Synapse RBAC
  - Synapse RBAC roles
  - Understand Synapse RBAC roles
  - System and user-assigned managed identity
  - Credentials for user-assigned managed identity
- How-to guides
  - Manage and Monitor
    - Azure Monitor metrics, alerts, and logs
  - Secure
    - Manage access control
      - Set up access control
      - Review Synapse role assignments
      - Manage Synapse role assignments
      - Set up permissions for Azure Synapse workspace managed identity
    - Data exfiltration protection
      - Create and manage a workspace with data exfiltration protection
    - Manage private endpoints
      - Set up Managed private endpoints to data sources
      - Connect to Azure Synapse workspace with private endpoints
      - Provision and secure a linked service with Managed VNet
      - Connect to Synapse workspace from restricted network
  - Troubleshoot
    - Troubleshoot Synapse Studio
    - Troubleshoot Synapse Studio connectivity issues
    - Troubleshoot connectivity between Synapse Studio and storage
    - Troubleshoot workspaces created by SDK
    - Troubleshoot reading UTF-8 text
  - Move workspace to new region
  - Move workspace to another tenant
- Database templates
  - Overview

- Database templates overview
- Map data overview
- Quickstarts
  - Create a lake database
- Concepts
  - Database templates
  - Lake database
- How-to guides
  - Create an empty lake database
  - Create lake database from database templates
  - Modify a lake database
  - Clone a lake database
- Synapse SQL
  - Overview
    - Synapse SQL architecture
    - Serverless SQL pool
    - Dedicated SQL pool
    - Features overview
    - Resource consumption models
    - Capacity limits
    - Cost management for serverless SQL pool
  - Quickstarts
    - Use serverless SQL pool
    - Create a dedicated SQL pool using Synapse Studio
    - Create a dedicated SQL pool using Azure portal
    - Pause and resume using Azure portal
    - Load data with dedicated SQL pools
      - COPY statement
      - COPY statement examples
      - Synapse Studio
    - Workload management
      - Create workload classifier

- Portal
- T-SQL
- Create workload isolation
  - Portal
  - T-SQL
- Tutorials
  - Use serverless SQL pool to analyze data
  - Create Logical Data Warehouse with serverless SQL pool
  - Power BI Desktop
- Concepts
  - Back up and restore
  - Data ingestion and preparation
    - Overview
    - Best practices
    - Columnstore compression
    - Data loading overview
  - Development
    - Overview
    - Best practices serverless SQL pool
    - Best practices dedicated SQL pools
    - Distribution Advisor
    - Materialized view
      - Performance tuning guidance
      - Create
      - Alter
    - Ordered clustered columnstore index
      - Performance tuning guidance
    - Result set caching
      - Performance tuning guidance
      - Set for a database
      - Set for a session
      - Check cache size

- Clean up cache
- Machine learning predictions
- Tables
  - Overview
  - CETAS
  - CTAS
  - Data types
  - Distributed tables
  - Table constraints
  - External tables
  - Indexes
  - Identity
  - OPENROWSET
  - Partitions
  - Replicated tables
  - Statistics
  - Temporary
- T-SQL language elements
  - Loops
  - Stored procedures
  - Transactions
    - Dedicated SQL pool transactions
    - Dedicated SQL pool best practices transactions
  - User-defined schemas
  - Variable assignment
  - Views
- Querying
  - Dynamic SQL
  - Group by options
  - Labels
  - Storage files
    - Overview

- Storage Access Control
- Apache Spark for Azure Synapse external tables
- Monitoring
  - Overview
  - Resource utilization & query activity
  - Recommendations
- Security
  - Database access control
  - Shared database access control
  - Data security
    - Column-level security
    - Row-level security
  - Data Encryption
    - Transparent Data Encryption overview
    - TDE with bring your own key
- Workload management
  - Overview
  - Workload classification
  - Workload importance
  - Workload isolation
  - Workload management monitoring
  - Memory & concurrency limits
  - Resource classes
- How-to guides
  - Back up and restore
    - Create restore points
    - Restore a dedicated SQL pool
    - Restore a SQL pool from a deleted workspace
    - Pause and resume using Synapse Pipelines
  - Manage and monitor
    - SQL requests - Synapse Studio
    - SQL pools - Synapse Studio

- Monitor Gen2 cache
- Monitor your workload - DMVs
- Monitor your workload - portal
- Query history and analysis
- Manage workloads
  - Analyze your workload
  - Configure workload importance
  - Manage and monitor workload importance
  - Convert resource classes to workload groups
  - Troubleshooting a missed classification
- Ingest and prepare data
  - Into the data warehouse
    - Azure Data Lake Storage
    - Data Factory
    - Azure Databricks
    - SSIS
    - New York taxi cab data
    - Contoso public data
    - Load WideWorldImporters
    - With the copy activity
  - Into the Data Lake
    - With the copy activity
- Change management and deployment
  - Install Visual Studio SSDT
  - Continuous integration and deployment
  - Deploy linked ARM templates with VSTS
- Query external data
  - Overview
  - Query CSV files
  - Query Parquet files
  - Query JSON files
  - Query nested types

- Query Delta Lake
- Query folders and multiple files
- Query partitioned files using file metadata
- Create and use views
- Create and use external tables
- Store query results to storage
- Query Parquet files and containers in Azure Synapse Link for Azure Cosmos DB
- Integrate
  - Build data pipelines
    - Add an Azure Stream Analytics job
- Connect with tools and applications
  - Overview
  - Query with Azure Data Studio
  - Connection strings
  - Query with SSMS
  - Connect to Power BI Desktop
  - Query with Visual Studio SSDT
  - Query using .NET Core
  - Query using Visual Studio .NET
  - Query using Python
  - Query with sqlcmd
- Data Explorer
  - Overview
    - Data Explorer overview
    - Compare with Azure Data Explorer
    - Data ingestion
      - Data ingestion overview
      - Data ingestion properties
      - Data ingestion formats
  - Quickstarts
    - Create a Data Explorer pool using the Azure portal
    - Create a Data Explorer pool using the Synapse Studio

- Ingest data
  - Overview
  - One-click ingestion
  - Streaming ingestion
  - Event Hub
    - Event Hub data connection
    - Azure portal
    - One-click
    - C#
    - Python
    - Azure Resource Manager template
  - Event Grid
    - Event Hub data connection
    - Azure portal
  - Ingest data using pipelines
- Monitor and diagnose pools
- Kusto Query Language
  - Kusto query overview
  - Quick reference guide
  - Tutorial
  - Samples
  - Query best practices
  - Cross-cluster queries
    - Cross-cluster queries and schema changes
    - Cross-database and cross-cluster queries
  - Entities
    - Entity types
    - Entity names
    - Entity references
    - Databases
    - Tables
    - Columns

- Stored functions
- External tables
- Fact and dimension tables
- Data types
  - Scalar data types
  - bool
  - datetime
  - decimal
  - dynamic
  - guid
  - int
  - long
  - real
  - string
  - timespan
  - Null values
  - Unsupported scalar data types
- Functions
  - Function types
  - User-defined functions
- Query statements
  - Query statement types
  - Alias statement
  - Let statement
  - Pattern statement
  - Query parameters statement
  - Restrict statement
  - Set statement
  - Tabular expression statements
  - Batches
- Tabular operators
  - Query operators

- as operator
- consume operator
- count operator
- datatable operator
- distinct operator
- evaluate operator
  - evaluate operator plugins
  - autocluster plugin
  - bag_unpack plugin
  - basket plugin
  - cosmosdb_sql_request plugin
  - dcount_intersect plugin
  - diffpatterns plugin
  - diffpatterns_text plugin
  - infer_storage_schema plugin
  - ipv4_lookup plugin
  - mysql_request plugin
  - narrow plugin
  - pivot plugin
  - preview plugin
  - Python plugin
  - R plugin
  - rolling_percentile plugin
  - schema_merge plugin
  - sql_request plugin
  - sequence_detect plugin
- extend operator
- externaldata operator
- facet operator
- find operator
- fork operator
- getschema operator

invoke operator
join operator
  join operator
  Shuffle query
  Cross-cluster join
  Broadcast join
  Time window join
limit operator
lookup operator
make-series operator
mv-apply operator
mv-expand operator
order operator
project operator
project-away operator
project-keep operator
project-rename operator
project-reorder operator
parse operator
parse-where operator
partition operator
print operator
range operator
reduce operator
render operator
sample operator
sample-distinct operator
scan operator
search operator
serialize operator
sort operator
summarize operator

summarize operator overview
Using hll() and tdigest()
shuffle
take operator
top operator
top-nested operator
top-hitters operator
union operator
where operator
Special functions
cluster()
database()
external_table()
materialize()
materialized_view()
table()
toscalar()
Scalar operators
Bitwise (binary) operators
Datetime/timespan arithmetic
Logical (binary) operators
Numerical operators
String operators
String operators overview
in and !in operators
has_all operator
has_any operator
between operator
not-between operator (!between)
Scalar functions
Scalar function types
abs()

acos()

ago()

around()

array_concat()

array_iif()

array_index_of()

array_length()

array_reverse()

array_rotate_left()

array_rotate_right()

array_shift_left()

array_shift_right()

array_slice()

array_sort_asc()

array_sort_desc()

array_split()

array_sum()

asin()

assert()

atan()

atan2()

bag_keys()

bag_merge()

bag_remove_keys()

base64_decode_toarray()

base64_decode_tostring()

base64_encode_fromarray()

base64_encode_tostring()

beta_cdf()

beta_inv()

beta_pdf()

bin()

bin_at()

bin_auto()

binary_and()

binary_not()

binary_or()

binary_shift_left()

binary_shift_right()

binary_xor()

bitset_count_ones()

case()

ceiling()

coalesce()

column_ifexists()

cos()

cot()

countof()

current_cluster_endpoint()

current_cursor()

current_database()

current_principal()

current_principal_details()

current_principal_is_member_of()

cursor_after()

cursor_before_or_at()

cursor_current()

datetime_add()

datetime_diff()

datetime_part()

dayofmonth()

dayofweek()

dayofyear()

dcount_hll()

degrees()

dynamic_to_json()

endofday()

endofmonth()

endofweek()

endofyear()

estimate_data_size()

exp()

exp10()

exp2()

extent_id()

extent_tags()

extract()

extract_all()

extractjson()

floor()

format_bytes()

format_datetime()

format_ipv4()

format_ipv4_mask()

format_timespan()

gamma()

getmonth()

gettype()

getyear()

has_any_ipv4()

has_ipv4()

has_any_ipv4_prefix()

has_ipv4_prefix()

hash()

hash_combine()

hash_many()

hash_md5()
hash_sha256()
hll_merge()
hourofday()
iif()
indexof()
indexof_regex()
ingestion_time()
ipv4_compare()
ipv4_is_in_range()
ipv4_is_match()
ipv4_is_private()
ipv4_netmask_suffix()
ipv6_compare()
ipv6_is_match()
isascii()
isempty()
isfinite()
isinf()
isnan()
isnotempty(), notempty()
isnotnull(), notnull()
isnull()
isutf8()
log()
log10()
log2()
loggamma()
make_datetime()
make_string()
make_timespan()
max_of()

min_of()

monthofyear()

new_guid()

not()

now()

pack()

pack_all()

pack_array()

pack_dictionary()

parse_command_line()

parse_csv()

parse_ipv4()

parse_ipv4_mask()

parse_ipv6()

parse_ipv6_mask()

parse_path()

parse_url()

parse_urlquery()

parse_user_agent()

parse_version()

parse_xml()

percentile_tdigest()

percentrank_tdigest()

pi()

pow()

radians()

rand()

range()

rank_tdigest()

regex_quote()

repeat()

replace()

reverse()

round()

set_difference()

set_has_element()

set_intersect()

set_union()

sign()

sin()

split()

sqrt()

startofday()

startofmonth()

startofweek()

startofyear()

strcat()

strcat_array()

strcat_delim()

strcmp()

string_size()

strlen()

strrep()

substring()

tan()

tdigest_merge()

to_utf8()

tobool()

todatetime()

todecimal()

todouble(), toreal()

todynamic(), parse_json()

toguid()

tohex()

- toint()
- tolong()
- tolower()
- tostring()
- totimespan()
- toupper()
- translate()
- treepath()
- trim()
- trim_end()
- trim_start()
- unixtime_microseconds_todatetime()
- unixtime_milliseconds_todatetime()
- unixtime_nanoseconds_todatetime()
- unixtime_seconds_todatetime()
- url_decode()
- url_encode()
- url_encode_component()
- weekofyear()
- welch_test()
- zip()
- zlib_compress_to_base64_string()
- zlib_decompress_from_base64_string()

Aggregation functions

- any()
- anyif()
- arg_max()
- arg_min()
- avg()
- avgif()
- binary_all_and
- binary_all_or

- binary_all_xor
- buildschema()
- count()
- countif()
- dcount()
- dcountif()
- hll()
- hll_merge()
- make_bag()
- make_bag_if()
- make_list()
- make_list_if()
- make_list_with_nulls()
- make_set()
- make_set_if()
- max()
- maxif()
- min()
- minif()
- percentiles(), percentile()
- stdev()
- stdevif()
- stdevp()
- sum()
- sumif()
- tdigest()
- tdigest_merge()
- variance()
- varianceif()
- variancep()

Window functions

- Window functions overview

- next()
- prev()
- row_cumsum()
- row_number()
- row_rank()
- row_window_session()

Time series analysis

- Time series analysis overview
- make-series operator
- series_add()
- series_decompose()
- series_decompose_anomalies()
- series_decompose_forecast()
- series_divide()
- series_equals()
- series_fft()
- series_fill_backward()
- series_fill_const()
- series_fill_forward()
- series_fill_linear()
- series_fir()
- series_fit_2lines()
- series_fit_2lines_dynamic()
- series_fit_line()
- series_fit_line_dynamic()
- series_fit_poly()
- series_greater()
- series_greater_equals()
- series_ifft()
- series_iir()
- series_less()
- series_less_equals()

- series_multiply()
- series_not_equals()
- series_outliers()
- series_pearson_correlation()
- series_periods_detect()
- series_periods_validate()
- series_seasonal()
- series_stats()
- series_stats_dynamic()
- series_subtract()

Machine Learning

- autocluster plugin
- basket plugin
- diffpatterns plugin
- diffpatterns_text plugin

Geospatial

- geo_distance_2points()
- geo_distance_point_to_line()
- geo_geohash_to_central_point()
- geo_line_densify()
- geo_point_in_circle()
- geo_point_in_polygon()
- geo_point_to_geohash()
- geo_point_to_s2cell()
- geo_polygon_to_s2cells()
- geo_polygon_densify()
- geo_s2cell_to_central_point()

User analytics

- User analytics plugins overview
- active_users_count plugin
- activity_counts_metrics plugin
- activity_engagement plugin

- activity_metrics plugin
- new_activity_metrics plugin
- funnel_sequence_completion plugin
- funnel_sequence plugin
- session_count plugin
- sliding_window_counts plugin
- Limits and Errors
  - Query consistency
  - Query limits
  - Partial query failures
    - Result truncation
    - Runaway queries
    - Overflows
- Reference material for Kusto Query Language
  - RE2 syntax
  - RE2 library
  - SQL to Kusto cheat sheet
  - Splunk to Kusto cheat sheet
- Management commands
  - Control commands overview
  - Workload groups
    - Workload groups overview
    - Workload groups management
    - Request classification policy
      - Request classification policy
      - Request classification policy commands
    - Request limits policy
    - Request rate limit policy
    - Request rate limits enforcement policy
    - Request queuing policy
  - Schema
    - Best practices for schema management

Databases
.show databases
.show database
.show cluster databases
.alter database prettyname
.show database(s) schema
.execute database script
Tables
Tables management
.alter-merge table
.alter table
.alter table docstring
.alter table folder
.alter ingestion mapping
.create ingestion mapping
.drop ingestion mapping
.show ingestion mapping
.create table
.create table based-on
.create tables
.create-merge table
.create-merge tables
.drop table(s)
.rename table(s)
.show tables
.undo drop table
.show table details
.show table schema
.clear table data
External tables
Create and alter Azure Storage external tables
Create and alter SQL Server external tables

.drop external table
.show external table
.show external table schema
Materialized views
Materialized views overview
.create materialized-view
.alter materialized-view
.create-or-alter materialized-view
.alter materialized-view docstring
.alter materialized-view folder
.alter materialized-view lookback
.alter materialized-view autoUpdateSchema
.show materialized-view commands
.drop materialized-view
.clear materialized-view data
.disable | .enable materialized-view
Materialized views policies
Materialized views data purge
Stored query results
Columns
Columns management
.alter column
.alter-merge table column-docstrings
.drop column
.rename column(s)
Data mappings
Functions
Stored functions management overview
.alter function
.alter function docstring
.alter function folder
.create function

- .create-or-alter function
- .drop function
- .show functions

Policies

- Auto delete
  - Auto delete policy
  - auto delete policy command
- Ingestion batching
  - Ingestionbatching policy
  - ingestionbatching policy command
- Cache
  - Cache policy (hot and cold cache)
  - cache policy command
- Callout
- Capacity
  - Capacity policy
  - capacity policy commands
- Ingestiontime
  - Ingestiontime policy
  - ingestiontime policy command
- Merge
  - Merge policy
  - merge policy command
- Partitioning
  - Partitioning policy
  - partitioning policy command
- Retention
  - Retention policy
  - retention policy command
- Extent tags retention
  - Extent tags retention policy
  - .show extent tags retention policy

.delete extent tags retention policy
.alter extent tags retention policy
Restricted view access
Restricted view access policy
restricted_view_access policy command
Row level security
Row level security policy
row_level_security policy command
Roworder
Roworder policy
roworder policy command
Sandbox
Using sandboxes
Sandbox policy
Sandbox policy command
Sharding
Sharding policy
sharding policy command
Streaming ingestion
Streaming ingestion policy
streamingingestion policy command
Update
Update policy
update policy commands
Query weak consistency
Query weak consistency policy
.alter query weak consistency policy
.show query weak consistency policy
Security roles
Data ingestion
Ingest from storage
Ingest from query

- Inline ingestion
- Streaming ingestion
  - Streaming ingestion and schema changes
  - Clear schema cache for streaming ingestion
- Data export
  - To cloud storage
  - To a SQL table
  - To an external table
  - Continuous data export
    - Continuous data export overview
    - Create or alter continuous export
    - Disable or enable continuous export
    - Drop continuous export
    - Show continuous export
    - Show continuous export artifacts
    - Show continuous export failures
- Database cursors
- Plugin commands
  - .show plugins
  - .enable plugin
  - .disable plugin
- Query results cache
  - Show cache
  - Clear cache
- System information
  - Diagnostics
  - Journal
  - Queries
  - Commands
  - Commands and queries
  - Operations
- Access control

- Identities
- Azure Active Directory
- Principal roles
- Allow cross-tenant queries and commands
- How-To Azure Active Directory authentication
- Advanced data management
- Extents (data shards)
- Extents overview
- Extents commands
- .alter extent
- .drop extents
- .drop extent tags
- .move extents
- .replace extents
- .show extents
- Extent containers clean commands
- Cluster follower
- Data purge
- Errors and failures
- Ingestion error codes
- Ingestion failures

Reference

- API
- API overview
- Client libraries
- Connection strings
- Connection strings overview
- Kusto connection strings
- Storage connection strings
- REST API
- REST API overview
- Authentication

- Query/management HTTP request
- Query/management HTTP response
- Query v2 HTTP response
- UI (Deep Links)
- Streaming ingest

MS-TDS

- MS-TDS overview
- T-SQL
- KQL over TDS
- Clients
- Linked server
- Azure Active Directory
- Known issues

.NET SDK

- General
  - About the SDK
  - Controlling tracing
- Kusto.Data
  - Kusto.Data overview
  - Request properties, ClientRequestProperties
- Kusto.Ingest
  - Kusto.Ingest overview
  - Permissions
  - Client reference
  - Errors and exceptions
  - Operation status
  - Kusto.Ingest best practices
  - Code samples
  - HowTo Data Ingestion with REST API

Python SDK

- azure-kusto-python

R SDK

- azure-kusto-r
- Java SDK
  - azure-kusto-java
- Node SDK
  - azure-kusto-node
- Go SDK
  - azure-kusto-go
- PowerShell
  - Calling Kusto from PowerShell
- Monaco IDE integration
  - monaco-kusto overview
  - Embed web UX in an iFrame

Functions library

- List of functions
- bartlett_test_fl()
- binomial_test_fl()
- comb_fl()
- factorial_fl()
- kmeans_fl()
- ks_test_fl()
- normality_test_fl()
- levene_test_fl()
- mann-whitney-u-test-fl()
- perm_fl()
- predict_fl()
- predict_onnx_fl()
- quantize_fl()
- series_dbl_exp_smoothing_fl()
- series_dot_product_fl()
- series_downsample_fl()
- series_exp_smoothing_fl()
- series_fit_lowess_fl()

- series_fit_poly_fl()
- series_fbprophet_forecast_fl()
- series_metric_fl()
- series_moving_avg_fl()
- series_rate_fl()
- series_rolling_fl()
- time_weighted_avg_fl()
- two_sample_t_test_fl()
- wilcoxon_test_fl()

Apache Spark

- Apache Spark overview
- Quickstarts
  - Create a notebook
  - Create a Spark pool with Azure portal
  - Create a Spark pool using Synapse Studio
  - Create a GPU-accelerated Apache Spark pool
- Tutorials
  - Data visualization with Synapse Studio notebooks
  - Analyze data with Apache Spark and Power BI
  - Read/write ADLS with Pandas
  - Read/write ADLS with filesystem spec
  - Create machine learning models
  - Submit Spark job definitions
- Concepts
  - Apache Spark
  - Apache Spark pool configurations
  - GPU-accelerated Apache Spark pools
  - Delta Lake
  - Optimize Apache Spark writes on Delta Lake
  - Apache Spark autoscale behavior
  - Intelligent Cache
  - Azure Synapse runtimes for Apache Spark

- Runtime for Apache Spark overview
- Azure Synapse Apache Spark 3.2 runtime
- Azure Synapse Apache Spark 3.1 runtime
- Azure Synapse Apache Spark 2.4 runtime
- Lifecycle and Supportability
- Machine learning
- Notebooks
- Package management
- Monitor Apache Spark applications

How-to guides

- Work with Delta Lake
- Manage packages
  - Manage Workspace packages
  - Manage Spark pool packages
  - Manage Notebook session packages
  - Troubleshoot Python package installation errors
  - Create a Conda private channel
  - Manage your package outside Synapse Studio portal
- Use notebooks and IDE tools
  - Manage notebooks
  - Visualize data
  - Connect to Spark with IntelliJ plug-in
  - Connect to Spark pools with VSCode
- Explore languages
  - .NET for Apache Spark
- Security
  - Secure credentials using TokenLibrary
- Optimize performance
  - Apache Spark performance
  - Performance optimization for Apache Spark using Hyperspace indexes
  - Accelerate ETL operations with NVIDIA RAPIDS
- Manage and monitor

- Navigate the Apache Spark pool history server
- Monitor Spark applications
- Monitor Apache Spark pools
- Collect Apache Spark applications metrics using APIs
- Monitor Apache Spark Applications metrics with Prometheus and Grafana
- Monitor your Apache Spark applications with Azure Log Analytics
- Collect Apache Spark applications logs and metrics with Azure Storage account
- Collect Apache Spark applications logs and metrics with Azure Event Hubs
- Manage Apache Spark configuration
- Apache Spark Advisor
- Data sources
  - Azure Cosmos DB Spark 3
  - Azure Cosmos DB Spark 2
  - Azure Data Explorer (Kusto)
  - Azure Spark CDM Connector
  - Azure SQL Dedicated
  - Azure SQL and SQL Server
  - External Hive Metastore
- Synapse Link
  - Overview
    - Synapse Link for Azure Cosmos DB
    - Synapse Link for SQL
    - Synapse Link for Dataverse
  - Quickstarts
    - Connect to Synapse Link for Cosmos DB
  - Concepts
    - Cosmos DB
      - Analytical Store Overview
      - Synapse Link supported features
      - Cosmos DB FAQ
      - Synapse Link use cases
    - SQL Database

- Synapse Link for Azure SQL Database
- SQL FAQ
- SQL Server 2022
  - Synapse Link for SQL Server 2022
  - SQL FAQ
- Dataverse
  - What is Dataverse
  - Configuration settings in Synapse Link
  - Dataverse FAQ
- How-to guides
  - Cosmos DB
    - Configure Synapse Link
    - Connect to Azure Synapse Link for Cosmos DB
    - Synapse Link and integrated Power BI
    - Use Power BI and serverless SQL pool to analyze Azure Cosmos DB
    - Query Azure Cosmos DB analytical store with Apache Spark 3
    - Query Azure Cosmos DB analytical store with Apache Spark 2
    - Copy data from Synapse Link into a dedicated SQL pool using Apache Spark
    - Query Azure Cosmos DB analytical store with serverless SQL pool
  - SQL Database
    - Configure Synapse Link
    - Known limitations and issues
  - SQL Server 2022
    - Configure Synapse Link
    - Known limitations and issues
  - Dataverse
    - Configure Synapse Link for Dataverse
    - Export data to Azure Synapse
- Pipeline and data flow
  - Quickstarts
    - Load data into SQL pools using copy activity
    - Transform data using mapping data flows

- Transform data using Apache Spark job definition
- Concepts
  - Pipeline and activity
  - Linked services
  - Datasets
  - Execution and trigger
  - Integration runtime
  - Differences from Azure Data Factory
  - Data flows
    - Data flow overview
    - Debug mode
    - Schema drift
    - Column patterns
    - Data flow monitoring
    - Data flow performance
      - Overview
      - Optimizing sources
      - Optimizing sinks
      - Optimizing transformations
      - Using data flows in pipelines
      - Memory optimized compute
    - Integration Runtime performance
    - Manage graph
    - Expression builder
    - Expression language
- How-to guides
  - Author
    - Visually author data factories
    - Iterative development and debugging
  - Connectors
    - Connector overview
    - Amazon Marketplace Web Service

Amazon RDS for Oracle

Amazon RDS for SQL Server

Amazon Redshift

Amazon S3

Amazon S3 Compatible Storage

Asana

Avro format

Azure Blob Storage

Azure Cognitive Search

Azure Cosmos DB SQL API

Azure Cosmos DB's API for MongoDB

Azure Data Explorer

Azure Data Lake Storage Gen1

Azure Data Lake Storage Gen2

Azure Database for MariaDB

Azure Database for MySQL

Azure Database for PostgreSQL

Azure Databricks Delta Lake

Azure File Storage

Azure SQL Database

Azure SQL Managed Instance

Azure Synapse Analytics

Azure Table Storage

Binary format

Cassandra

Common Data Model format

Concur

Couchbase

data.world

Dataverse

DB2

Delimited text format

Drill

Dynamics 365

Dynamics AX

Dynamics CRM

Excel format

File System

FTP

GitHub

Google AdWords

Google BigQuery

Google Cloud Storage

Greenplum

HBase

HDFS

Hive

HTTP

HubSpot

Impala

Informix

Jira

JSON format

Magento

MariaDB

Marketo

Microsoft Access

MongoDB

  MongoDB (legacy)

MongoDB Atlas

MySQL

Netezza

OData

ODBC

Office 365
Oracle
Oracle Cloud Storage
Oracle Eloqua
Oracle Responsys
Oracle Service Cloud
ORC format
Parquet format
PayPal
Phoenix
PostgreSQL
Presto
Quickbase
QuickBooks Online
REST
Salesforce
Salesforce Service Cloud
Salesforce Marketing Cloud
SAP Business Warehouse Open Hub
  Load SAP BW data
SAP Business Warehouse MDX
SAP Cloud for Customer
SAP ECC
SAP HANA
SAP Table
ServiceNow
SFTP
SharePoint Online List
Shopify
Smartsheet
Snowflake
Spark

- SQL Server
- Square
- Sybase
- TeamDesk
- Teradata
- Twilio
- Vertica
- Web Table
- Xero
- XML format
- Zendesk
- Zoho

Move data

- Copy data using copy activity
- Monitor copy activity
- Delete files using delete activity
- Copy Data tool
- Format and compression support
- Copy activity performance
  - Performance and scalability guide
  - Troubleshoot performance
  - Performance features
- Preserve metadata and ACLs
- Schema and type mapping
- Fault tolerance
- Data consistency verification
- Format and compression support (legacy)

Transform data

- Execute Data Flow activity
- Synapse notebook activity
- Synapse SQL pool stored procedure activity
- Azure Function activity

- Custom activity
- Databricks Jar activity
- Databricks Notebook activity
- Databricks Python activity
- Data Explorer Command activity
- Data Lake U-SQL activity
- HDInsight Hive activity
- HDInsight MapReduce activity
- HDInsight Pig activity
- HDInsight Spark activity
- HDInsight Streaming activity
- Machine Learning execute pipeline activity
- Machine Learning Studio (classic) batch execution activity
- Machine Learning Studio (classic) update resource activity
- Stored procedure activity
- Script activity
- Compute linked services

Control flow

- Append variable activity
- Execute pipeline activity
- Filter activity
- For each activity
- Get metadata activity
- If condition activity
- Lookup activity
- Set variable activity
- Until activity
- Validation activity
- Wait activity
- Web activity
- Webhook activity

Data flow transformations

- Transformation overview
- Aggregate
- Alter row
- Conditional split
- Derived column
- Exists
- Filter
- Flatten
- Join
- Lookup
- New branch
- Pivot
- Rank
- Select
- Sink
- Sort
- Source
- Surrogate key
- Union
- Unpivot
- Window

Parameterize

- Parameterize linked services
- Expression Language
- System variables
- Data flow parameters

Manage and monitor

- Pipeline runs
- Send email from a pipeline
- Send Microsoft Teams notifications from a pipeline

Create integration runtime

- Azure integration runtime

- Self-hosted integration runtime
- Create triggers
  - Create an event-based trigger
  - Create a schedule trigger
  - Create a tumbling window trigger
  - Create a tumbling window trigger dependency
- Troubleshooting guide
  - Activity troubleshooting
  - Connector troubleshooting
    - Overview and general copy activity errors
    - Azure Blob Storage
    - Azure Cosmos DB (including SQL API)
    - Azure Data Explorer
    - Azure Data Lake (Gen1 and Gen2)
    - Azure database for PostgreSQL
    - Azure files
    - Azure Synapse Analytics, Azure SQL Database, and SQL Server
    - Azure Table Storage
    - DB2
    - Delimited text format
    - Dynamics 365, Dataverse (Common Data Service), and Dynamics CRM
    - FTP, SFTP and HTTP
    - Hive
    - Oracle
    - ORC format
    - Parquet format
    - REST
    - SharePoint Online list
    - Snowflake
    - XML format
  - Continuous Integration and Deployment
  - Security and access control

- Self-hosted Integration Runtimes troubleshooting
- Machine Learning
  - Overview
    - What is machine learning in Azure Synapse?
  - Quickstarts
    - Use a sample notebook from the gallery
    - Use the Retail AI solution
    - Create a linked service
      - Azure Machine Learning
      - Cognitive Services
  - Concepts
    - Train machine learning models
    - Deep learning
  - Tutorials
    - Data access and preparation
      - Visualize data with Apache Spark
    - Train a model
      - With Apache Spark MLlib
      - With automated machine learning
        - Code-free
        - In Python
      - With Horovod and Petastorm
        - Train a model with Horovod and PyTorch
        - Train a model with Horovod and TensorFlow
        - Write and load data with Petastorm
    - Score a model
      - Use the scoring wizard for dedicated SQL pools
      - PREDICT score in Synapse spark using model registered in AML or ADLS
      - With Cognitive Services
        - Overview
        - Sentiment analysis wizard
        - Anomaly detection wizard

- Text Analytics with Apache Spark (SynapseML)
- Computer Vision with Apache Spark (SynapseML)
- Form Recognizer with Apache Spark (SynapseML)
- Translator with Apache Spark (SynapseML)
- With Apache Spark (SynapseML)

Data Catalog and Governance

- Quickstarts
  - Connect a Microsoft Purview Account
- How-to guides
  - Access a secured Microsoft Purview account
  - Discover, connect and explore data in Synapse using Microsoft Purview

How-to

- Plan and manage costs
- Browse an ADLS Gen2 folder with ACLs

Guidance

- Azure Synapse Success by Design
  - Introduction
  - Synapse proof of concept playbook
    - Overview
    - Data warehousing with dedicated SQL pool
    - Data lake exploration with serverless SQL pool
    - Big data analytics with Apache Spark pool
  - Synapse implementation success methodology
    - Overview
    - Assess environment
    - Evaluate workspace design
    - Evaluate data integration design
    - Evaluate dedicated SQL pool design
    - Evaluate serverless SQL pool design
    - Evaluate Spark pool design
    - Evaluate project plan
    - Evaluate solution development environment design

- Evaluate team skill sets
- Perform operational readiness review
- Perform user readiness and onboarding plan review
- Perform monitoring review

Migration guides

- All migration guides
- Migrate to a dedicated SQL pool
- From Teradata
  - 1 Design performance for Teradata migration
  - 2 ETL and load migration considerations
  - 3 Security access operations
  - 4 Visualization and reporting
  - 5 Minimizing SQL issues
  - 6 Microsoft and third-party tools
  - 7 Implementing modern data warehouses
- From Netezza
  - 1 Design performance for Netezza migration
  - 2 ETL and load migration considerations
  - 3 Security access operations
  - 4 Visualization and reporting
  - 5 Minimizing SQL issues
  - 6 Microsoft and third-party tools
  - 7 Implementing modern data warehouses
- From Oracle
  - 1 Design performance for Oracle migration
  - 2 ETL and load migration considerations
  - 3 Security access operations
  - 4 Visualization and reporting
  - 5 Minimizing SQL issues
  - 6 Microsoft and third-party tools
  - 7 Implementing modern data warehouses

Security white paper

- Introduction
- Data protection
- Access control
- Authentication
- Network security
- Threat protection

Reference

- Azure CLI
- Database collation types
- Go
- .NET
- Azure PowerShell
- REST APIs
- Microsoft Spark Utilities
- File mount/unmount API in Synapse
- Samples
- System views
- Azure Policy built-ins
- T-SQL
  - Full reference
  - Synapse SQL T-SQL language elements
  - Synapse SQL T-SQL statements

Resources

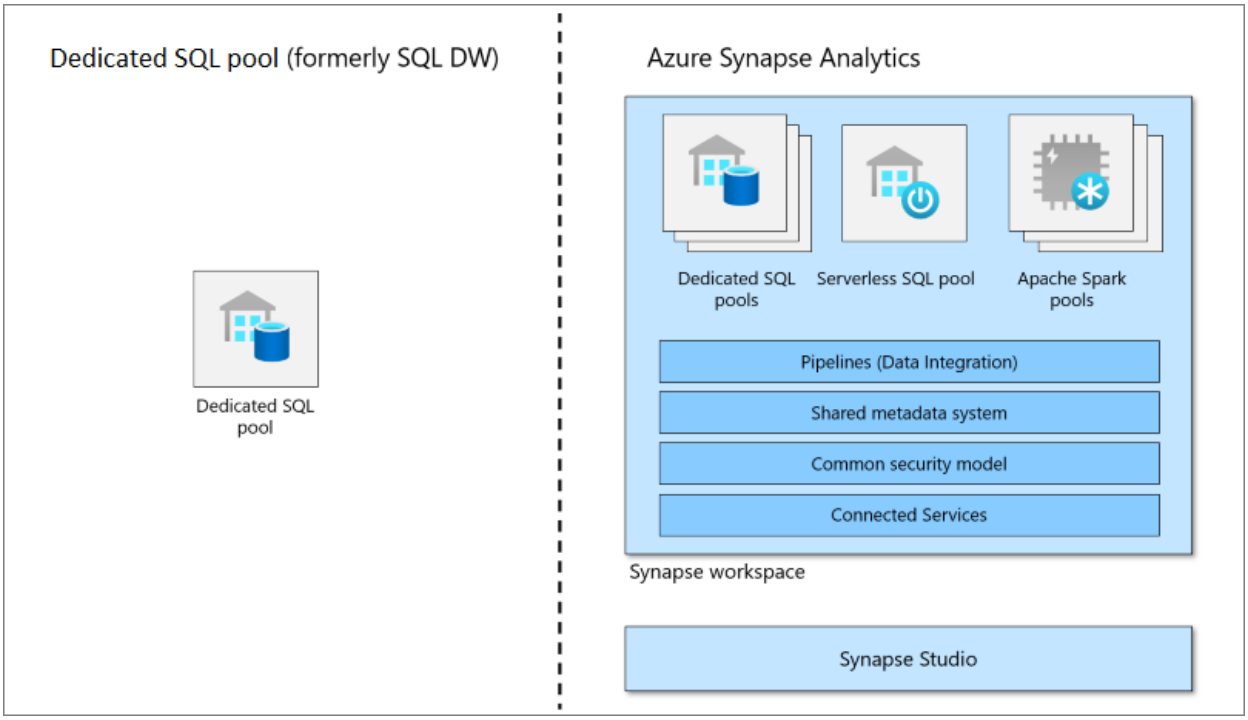
- Azure Roadmap
- Microsoft Q&A question page
- Feature requests
- Pricing
- Pricing calculator
- Questions & Answers
- Monthly updates archive
- Service updates
- Stack Overflow

- Support
- Customer self-help
  - serverless SQL pool
- Partners
  - Business intelligence
  - Data integration
  - Data management
  - Machine learning and AI
  - System integration
  - Compatibility issues with third-party applications

What is dedicated SQL pool (formerly SQL DW) in Azure Synapse Analytics?

8/19/2022 • 2 minutes to read • Edit Online

Azure Synapse Analytics is an analytics service that brings together enterprise data warehousing and Big Data analytics. Dedicated SQL pool (formerly SQL DW) refers to the enterprise data warehousing features that are available in Azure Synapse Analytics.



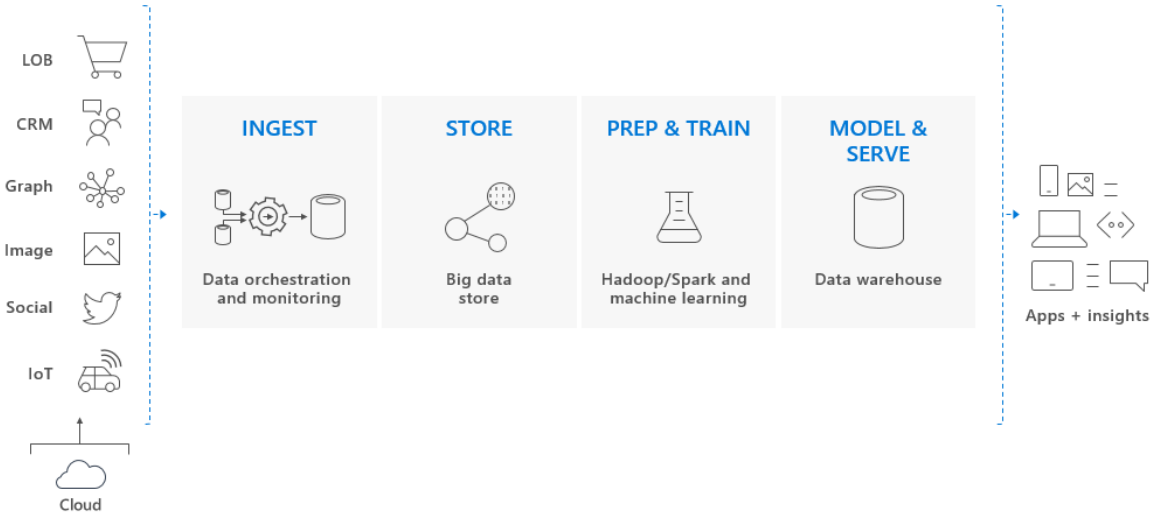
Dedicated SQL pool (formerly SQL DW) represents a collection of analytic resources that are provisioned when using Synapse SQL. The size of a dedicated SQL pool (formerly SQL DW) is determined by Data Warehousing Units (DWU).

Once your dedicated SQL pool is created, you can import big data with simple PolyBase T-SQL queries, and then use the power of the distributed query engine to run high-performance analytics. As you integrate and analyze the data, dedicated SQL pool (formerly SQL DW) will become the single version of truth your business can count on for faster and more robust insights.

> **NOTE**
> Not all features of the dedicated SQL pool in Azure Synapse workspaces apply to dedicated SQL pool (formerly SQL DW), and vice versa. To enable workspace features for an existing dedicated SQL pool (formerly SQL DW) refer to How to enable a workspace for your dedicated SQL pool (formerly SQL DW). Explore the Azure Synapse Analytics documentation and Get Started with Azure Synapse.

Key component of a big data solution

Data warehousing is a key component of a cloud-based, end-to-end big data solution.



In a cloud data solution, data is ingested into big data stores from a variety of sources. Once in a big data store, Hadoop, Spark, and machine learning algorithms prepare and train the data. When the data is ready for complex analysis, dedicated SQL pool uses PolyBase to query the big data stores. PolyBase uses standard T-SQL queries to bring the data into dedicated SQL pool (formerly SQL DW) tables.

Dedicated SQL pool (formerly SQL DW) stores data in relational tables with columnar storage. This format significantly reduces the data storage costs, and improves query performance. Once data is stored, you can run analytics at massive scale. Compared to traditional database systems, analysis queries finish in seconds instead of minutes, or hours instead of days.

The analysis results can go to worldwide reporting databases or applications. Business analysts can then gain insights to make well-informed business decisions.

Next steps

- Explore Azure Synapse architecture
- Quickly create a dedicated SQL pool
- Load sample data.
- Explore Videos
- Get Started with Azure Synapse

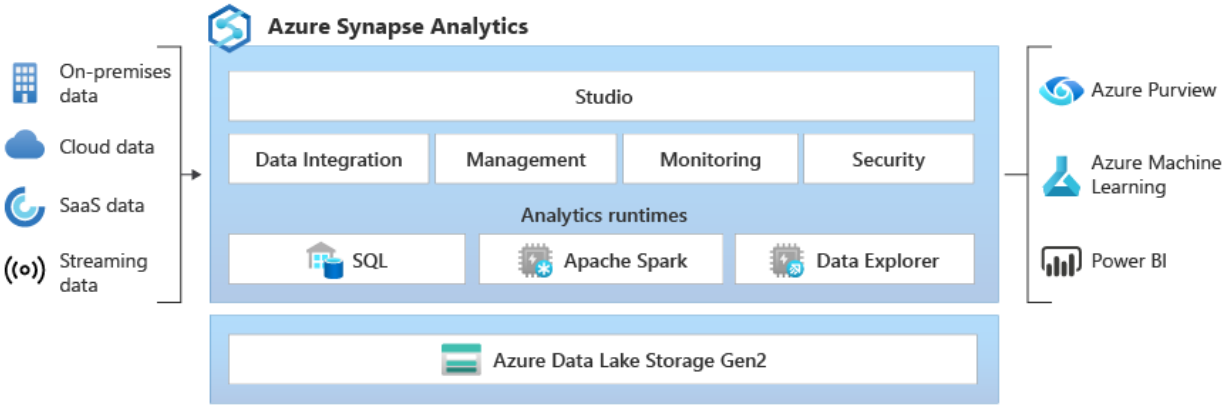
Or look at some of these other Azure Synapse resources.

- Search Blogs
- Submit a Feature requests
- Create a support ticket
- Search Microsoft Q&A question page
- Search Stack Overflow forum

What is Azure Synapse Analytics?

8/19/2022 • 3 minutes to read • Edit Online

Azure Synapse is an enterprise analytics service that accelerates time to insight across data warehouses and big data systems. Azure Synapse brings together the best of **SQL** technologies used in enterprise data warehousing, **Spark** technologies used for big data, **Data Explorer** for log and time series analytics, **Pipelines** for data integration and ETL/ELT, and deep integration with other Azure services such as **Power BI**, **CosmosDB**, and **AzureML**.



Industry-leading SQL

Synapse SQL is a distributed query system for T-SQL that enables data warehousing and data virtualization scenarios and extends T-SQL to address streaming and machine learning scenarios.

- Synapse SQL offers both **serverless** and **dedicated** resource models. For predictable performance and cost, create dedicated SQL pools to reserve processing power for data stored in SQL tables. For unplanned or bursty workloads, use the always-available, serverless SQL endpoint.
- Use built-in **streaming** capabilities to land data from cloud data sources into SQL tables
- Integrate AI with SQL by using **machine learning** models to score data using the T-SQL PREDICT function

Industry-standard Apache Spark

Apache Spark for Azure Synapse deeply and seamlessly integrates Apache Spark--the most popular open source big data engine used for data preparation, data engineering, ETL, and machine learning.

- ML models with SparkML algorithms and AzureML integration for Apache Spark 3.1 with built-in support for Linux Foundation Delta Lake.
- Simplified resource model that frees you from having to worry about managing clusters.
- Fast Spark start-up and aggressive autoscaling.
- Built-in support for .NET for Spark allowing you to reuse your C# expertise and existing .NET code within a Spark application.

Working with your Data Lake

Azure Synapse removes the traditional technology barriers between using SQL and Spark together. You can seamlessly mix and match based on your needs and expertise.

- Tables defined on files in the data lake are seamlessly consumed by either Spark or Hive.

- SQL and Spark can directly explore and analyze Parquet, CSV, TSV, and JSON files stored in the data lake.
- Fast, scalable data loading between SQL and Spark databases

Built-in data integration

Azure Synapse contains the same Data Integration engine and experiences as Azure Data Factory, allowing you to create rich at-scale ETL pipelines without leaving Azure Synapse Analytics.

- Ingest data from 90+ data sources
- Code-Free ETL with Data flow activities
- Orchestrate notebooks, Spark jobs, stored procedures, SQL scripts, and more

Data Explorer (Preview)

Azure Synapse Data Explorer provides customers with an interactive query experience to unlock insights from log and telemetry data. To complement existing SQL and Apache Spark analytics runtime engines, Data Explorer analytics runtime is optimized for efficient log analytics using powerful indexing technology to automatically index free-text and semi-structured data commonly found in the telemetry data.

Use Data Explorer as a data platform for building near real-time log analytics and IoT analytics solutions to:

- Consolidate and correlate your logs and events data across on-premises, cloud, third-party data sources.
- Accelerate your AI Ops journey (pattern recognition, anomaly detection, forecasting, and more)
- Replace infrastructure-based log search solutions to save cost and increase productivity.
- Build IoT Analytics solution for your IoT data.
- Build Analytical SaaS solutions to offer services to your internal and external customers.

Unified experience

Synapse Studio provides a single way for enterprises to build solutions, maintain, and secure all in a single user experience

- Perform key tasks: ingest, explore, prepare, orchestrate, visualize
- Monitor resources, usage, and users across SQL, Spark, and Data Explorer
- Use Role-based access control to simplify access to analytics resources
- Write SQL, Spark or KQL code and integrate with enterprise CI/CD processes

Engage with the Synapse community

- Microsoft Q&A: Ask technical questions.
- Stack Overflow: Ask development questions.

Next steps

- Get started with Azure Synapse Analytics
- Create a workspace
- Use serverless SQL pool
- Create a Data Explorer pool using Synapse Studio (Preview)

What's new in Azure Synapse Analytics?

8/19/2022 • 5 minutes to read • Edit Online

This article lists updates to Azure Synapse Analytics that are published in April 2022. Each update links to the Azure Synapse Analytics blog and an article that provides more information. For previous months releases, check out Azure Synapse Analytics - updates archive.

General

- **Azure Orbital analytics with Synapse Analytics** - We now offer an Azure Orbital analytics sample solution showing an end-to-end implementation of extracting, loading, transforming, and analyzing spaceborne data by using geospatial libraries and AI models withAzure Synapse Analytics. The sample solution also demonstrates how to integrate geospatial-specificAzure Cognitive Servicesmodels, AI models from partners, and bring-your-own-data models.

- **Azure Synapse success by design** - Project success is no accident and requires careful planning and execution. The Synapse Analytics' Success by Design playbooks are now available. The Azure Synapse proof of concept playbook provides a guide to scope, design, execute, and evaluate a proof of concept for SQL or Spark workloads. These guides contain best practices from the most challenging and complex solution implementations incorporating Azure Synapse. To learn more about the Azure Synapse proof of concept playbook, read Success by Design.

SQL

Result set size limit increase - We know that you turn to Azure Synapse Analytics to work with large amounts of data. With that in mind, the maximum size of query result sets in Serverless SQL pools has been increased from 200GB to 400GB. This limit is shared between concurrent queries. To learn more about this size limit increase and other constraints, read Self-help for serverless SQL pool.

Synapse data explorer

- **Web Explorer new homepage** - The new Synapse Web Explorer homepage makes it even easier to get started with Synapse Web Explorer. The Web Explorer homepage now includes the following sections:
    - Get started – Sample gallery offering example queries and dashboards for popular Synapse Data Explorer use cases.
    - Recommended – Popular learning modules designed to help you master Synapse Web Explorer and KQL.
    - Documentation – Synapse Web Explorer basic and advanced documentation.
- **Web Explorer sample gallery** - A great way to learn about a product is to see how it is being used by others. The Web Explorer sample gallery provides end-to-end samples of how customers leverage Synapse Data Explorer popular use cases such as Logs Data, Metrics Data, IoT data and Basic big data examples. Each sample includes the dataset, well-documented queries, and a sample dashboard. To learn more about the sample gallery, read Azure Data Explorer in 60 minutes with the new samples gallery.

- **Web Explorer dashboards drill through capabilities** - You can now add drill through capabilities to your Synapse Web Explorer dashboards. The new drill through capabilities allow you to easily jump back and forth between dashboard pages. This is made possible by using a contextual filter to connect your dashboards. Defining these contextual drill throughs is done by editing the visual interactions of the selected tile in your dashboard. To learn more about drill through capabilities, read Use drillthroughs as

dashboard parameters.

- **Time Zone settings for Web Explorer** - Being able to display data in different time zones is very powerful. You can now decide to view the data in UTC time, your local time zone, or the time zone of the monitored device/machine. The Time Zone settings of the Web Explorer now apply to both the Query results and to the Dashboard. By changing the time zone, the dashboards will be automatically refreshed to present the data with the selected time zone. For more information on time zone settings, read Change datetime to specific time zone.

Data integration

- **Fuzzy Join option in Join Transformation** - Fuzzy matching with a sliding similarity score option has been added to the Join transformation in Mapping Data Flows. You can create inner and outer joins on data values that are similar rather than exact matches! Previously, you would have had to use an exact match. The sliding scale value goes from 60% to 100%, making it easy to adjust the similarity threshold of the match. For learn more about fuzzy joins, read Join transformation in mapping data flow.

- **Map Data [Generally Available]** - We're excited to announce that the Map Data tool is now Generally Available. The Map Data tool is a guided process to help you create ETL mappings and mapping data flows from your source data to Synapse without writing code. To learn more about Map Data, read Map Data in Azure Synapse Analytics.

- **Rerun pipeline with new parameters** - You can now change pipeline parameters when re-running a pipeline from the Monitoring page without having to return to the pipeline editor. After running a pipeline with new parameters, you can easily monitor the new run against the old ones without having to toggle between pages. To learn more about rerunning pipelines with new parameters, read Rerun pipelines and activities.

- **User Defined Functions [Generally Available]** - We're excited to announce that user defined functions (UDFs) are now Generally Available. With user-defined functions, you can create customized expressions that can be reused across multiple mapping data flows. You no longer have to use the same string manipulation, math calculations, or other complex logic several times. User-defined functions will be grouped in libraries to help developers group common sets of functions. To learn more about user defined functions, read User defined functions in mapping data flows.

Machine learning

Distributed Deep Neural Network Training with Horovod and Petastorm [Public Preview] - To simplify the process for creating and managing GPU-accelerated pools, Azure Synapse takes care of pre-installing low-level libraries and setting up all the complex networking requirements between compute nodes. This integration allows users to get started with GPU- accelerated pools within just a few minutes.

Now, Azure Synapse Analytics provides built-in support for deep learning infrastructure. The Azure Synapse Analytics runtime for Apache Spark 3.1 and 3.2 now includes support for the most common deep learning libraries like TensorFlow and PyTorch. The Azure Synapse runtime also includes supporting libraries like Petastorm and Horovod, which are commonly used for distributed training. This feature is currently available in Public Preview.

To learn more about how to leverage these libraries within your Azure Synapse Analytics GPU-accelerated pools, read the Deep learning tutorials.

Next steps

Get started with Azure Synapse Analytics

Azure Synapse Analytics terminology

8/19/2022 • 3 minutes to read • Edit Online

This document guides you through the basic concepts of Azure Synapse Analytics.

Basics

A **Synapse workspace** is a securable collaboration boundary for doing cloud-based enterprise analytics in Azure. A workspace is deployed in a specific region and has an associated ADLS Gen2 account and file system (for storing temporary data). A workspace is under a resource group.

A workspace allows you to perform analytics with SQL and Apache spark. Resources available for SQL and Spark analytics are organized into SQL and Spark **pools**.

Linked services

A workspace can contain any number of **Linked service**, essentially connection strings that define the connection information needed for the workspace to connect to external resources.

Synapse SQL

Synapse SQL is the ability to do T-SQL based analytics in Synapse workspace. Synapse SQL has two consumption models: dedicated and serverless. For the dedicated model, use **dedicated SQL pools**. A workspace can have any number of these pools. To use the serverless model, use the **serverless SQL pools**. Every workspace has one of these pools.

Inside Synapse Studio, you can work with SQL pools by running **SQL scripts**.

> **NOTE**
> Dedicated SQL pools in Azure Synapse is different from the dedicated SQL pool (formerly SQL DW). Not all features of the dedicated SQL pool in Azure Synapse workspaces apply to dedicated SQL pool (formerly SQL DW), and vice versa. To enable workspace features for an existing dedicated SQL pool (formerly SQL DW), see How to enable a workspace for your dedicated SQL pool (formerly SQL DW).

Apache Spark for Synapse

To use Spark analytics, create and use **serverless Apache Spark pools** in your Synapse workspace. When you start using a Spark pool, the workspaces creates a **spark session** to handle the resources associated with that session.

There are two ways within Synapse to use Spark:

- **Spark Notebooks** for doing data Data Science and Engineering use Scala, PySpark, C#, and SparkSQL
- **Spark job definitions** for running batch Spark jobs using jar files.

Pipelines

Pipelines are how Azure Synapse provides Data Integration - allowing you to move data between services and orchestrate activities.

- **Pipeline** are logical grouping of activities that perform a task together.
- **Activities** defines actions within a Pipeline to perform on data such as copying data, running a Notebook or a SQL script.
- **Data flows** are a specific kind of activity that provide a no-code experience for doing data transformation that uses Synapse Spark under-the-covers.
- **Trigger** - Executes a pipeline. It can be run manually or automatically (schedule, tumbling window or event-based)
- **Integration dataset** - Named view of data that simply points or references the data to be used in an activity as input and output. It belongs to a Linked Service.

Data Explorer (Preview)

Azure Synapse Data Explorer provides customers with an interactive query experience to unlock insights from log and telemetry data.

- **Data Explorer pools** are dedicated clusters that includes two or more compute nodes with local SSD storage (hot cache) for optimized query performance and multiple blob storage (cold cache) for persistence.
- **Data Explorer databases** are hosted on Data Explorer pools and are logical entities made up of collections of tables and other database objects. You can have more than one database per pool.
- **Tables** are database objects that contain data that is organized using a traditional relational data model. Data is stored in records that adhere to Data Explorer's well-defined table schema that defines an ordered list of columns, each column having a name and scalar data type. Scalar data types can be structured (*int*, *real*, *datetime*, or *timespan*), semi-structured (*dynamic*), or free text (*string*). The dynamic type is similar to JSON in that it can hold a single scalar value, an array, or a dictionary of such values.
- **External Tables** are tables that reference a storage or SQL data source outside the Data Explorer database. Similar to tables, an external table has a well-defined schema (an ordered list of column name and data type pairs). Unlike Data Explorer tables where data is ingested into Data Explorer pools, external tables operate on data stored and managed outside pools. External tables don't persist any data and are used to query or export data to an external data store.

Next steps

- Get started with Azure Synapse Analytics
- Create a workspace
- Use serverless SQL pool
- Create a Data Explorer pool using Synapse Studio

Get Started with Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

This tutorial is a step-by-step guide through the major feature areas of Azure Synapse Analytics. The tutorial is the ideal starting point for someone who wants a guided tour through the key scenarios of Azure Synapse Analytics. After following the steps in the tutorial, you will have a Synapse workspace. This tutorial also includes steps to enable a workspace for your dedicated SQL pool (formerly SQL DW). Once your workspace is created, you can start analyzing data using dedicated SQL pool, serverless SQL pool, or serverless Apache Spark pool.

Follow the steps *in order* as shown below and you'll take a tour through many of the capabilities and learn how to exercise its core features.

- STEP 1 - Create and setup a Synapse workspace
- STEP 2 - Analyze using a serverless SQL pool
- STEP 3 - Analyze using a Data Explorer pool
- STEP 4 - Analyze using Apache Spark
- STEP 5 - Analyze using a dedicated SQL pool
- STEP 6 - Analyze data in a storage account
- STEP 7 - Orchestrate with pipelines
- STEP 8 - Visualize data with Power BI
- STEP 9 - Monitor activities
- STEP 10 - Explore the Knowledge center
- STEP 11 - Add an administrator

Creating a Synapse workspace

8/19/2022 • 3 minutes to read • Edit Online

In this tutorial, you'll learn how to create a Synapse workspace, a dedicated SQL pool, and a serverless Apache Spark pool.

Prerequisites

To complete this tutorial's steps, you need to have access to a resource group for which you are assigned the **Owner** role. Create the Synapse workspace in this resource group.

Create a Synapse workspace in the Azure portal

Start the process

1. Open the Azure portal, in the search bar enter **Synapse** without hitting enter.
2. In the search results, under **Services**, select **Azure Synapse Analytics**.
3. Select **Create** to create a workspace.

Basics tab > Project Details

Fill in the following fields:

1. **Subscription** - Pick any subscription.
2. **Resource group** - Use any resource group.
3. **Managed Resource group** - Leave this blank.

Basics tab > Workspace details

Fill in the following fields:

1. **Workspace name** - Pick any globally unique name. In this tutorial, we'll use **myworkspace**.
2. **Region** - Pick the region where you have placed your client applications/services (for example, Azure VM, Power BI, Azure Analysis Service) and storages that contain data (for example Azure Data Lake storage, Azure Cosmos DB analytical storage).

> **NOTE**
> A workspace that is not co-located with the client applications or storage can be the root cause of many performance issues. If your data or the clients are placed in multiple regions, you can create separate workspaces in different regions co-located with your data and clients.

Under **Select Data Lake Storage Gen 2**:

1. By **Account name**, select **Create New** and name the new storage account **contosolake** or similar as the name must be unique.
2. By **File system name**, select **Create New** and name it **users**. This will create a storage container called **users**. The workspace will use this storage account as the "primary" storage account to Spark tables and Spark application logs.
3. Check the "Assign myself the Storage Blob Data Contributor role on the Data Lake Storage Gen2 account"

box.

Completing the process

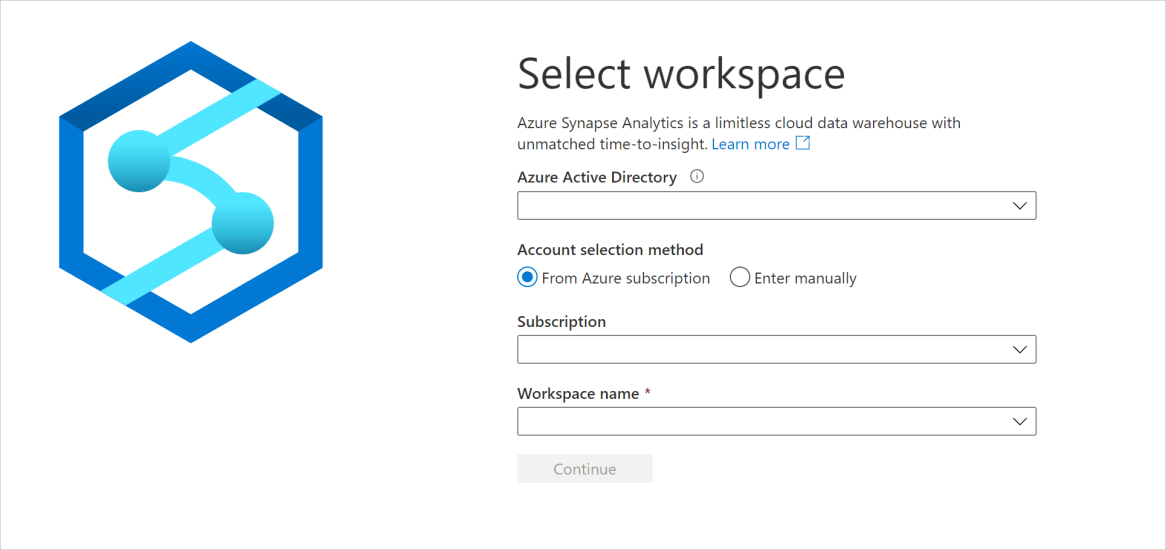
Select **Review + create** > **Create**. Your workspace is ready in a few minutes.

> **NOTE**
> To enable workspace features from an existing dedicated SQL pool (formerly SQL DW) refer to How to enable a workspace for your dedicated SQL pool (formerly SQL DW).

Open Synapse Studio

After your Azure Synapse workspace is created, you have two ways to open Synapse Studio:

1. Open your Synapse workspace in the Azure portal, in the **Overview** section of the Synapse workspace, select **Open** in the Open Synapse Studio box.
2. Go to the `https://web.azuresynapse.net` and sign in to your workspace.



> **NOTE**
> To sign into your workspace, there are two **Account selection methods**. One is from **Azure subscription**, the other is from **Enter manually**. If you have the Synapse Azure role or higher level Azure roles, you can use both methods to log into the workspace. If you don't have the related Azure roles, and you were granted as the Synapse RBAC role, **Enter manually** is the only way to log into the workspace. To learn more about the Synapse RBAC, refer to What is Synapse role-based access control (RBAC).

Place sample data into the primary storage account

We are going to use a small 100K row sample dataset of NYX Taxi Cab data for many examples in this getting started guide. We begin by placing it in the primary storage account you created for the workspace.

- Download this file to your computer: https://azuresynapsestorage.blob.core.windows.net/sampledata/NYCTaxiSmall/NYCTripSmall.parquet
- In Synapse Studio, navigate to the Data Hub.
- Select **Linked**.
- Under the category **Azure Data Lake Storage Gen2** you'll see an item with a name like **myworkspace (**

Primary - contosolake).

- Select the container named **users (Primary)**.
- Select **Upload** and select the `NYCTripSmall.parquet` file you downloaded.

Once the parquet file is uploaded it is available through two equivalent URIs:

- `https://contosolake.dfs.core.windows.net/users/NYCTripSmall.parquet`
- `abfss://users@contosolake.dfs.core.windows.net/NYCTripSmall.parquet`

In the examples that follow in this tutorial, make sure to replace **contosolake** in the UI with the name of the primary storage account that you selected for your workspace.

Next steps

Analyze using a serverless SQL pool

Analyze data with a serverless SQL pool

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, you'll learn how to analyze data with serverless SQL pool.

The Built-in serverless SQL pool

Serverless SQL pools let you use SQL without having to reserve capacity. Billing for a serverless SQL pool is based on the amount of data processed to run the query and not the number of nodes used to run the query.

Every workspace comes with a pre-configured serverless SQL pool called **Built-in**.

Analyze NYC Taxi data with a serverless SQL pool

> **NOTE**
> Make sure you have placed the sample data into the primary storage account

1. In Synapse Studio, go to the **Develop** hub
2. Create a new SQL script.
3. Paste the following code into the script.

```
SELECT
    TOP 100 *
FROM
    OPENROWSET(
        BULK 'https://contosolake.dfs.core.windows.net/users/NYCTripSmall.parquet',
        FORMAT='PARQUET'
    ) AS [result]
```

4. Click **Run**.

Data exploration is just a simplified scenario where you can understand the basic characteristics of your data. Learn more about data exploration and analysis in this tutorial.

Create data exploration database

You can browse the content of the files directly via `master` database. For some simple data exploration scenarios, you don't need to create a separate database. However, as you continue data exploration, you might want to create some utility objects, such as:

- External data sources that represent the named references for storage accounts.
- Database scoped credentials that enable you to specify how to authenticate to external data source.
- Database users with the permissions to access some data sources or database objects.
- Utility views, procedures, and functions that you can use in the queries.

1. Use the `master` database to create a separate database for custom database objects. Custom database objects, cannot be created in the `master` database.

```
CREATE DATABASE DataExplorationDB
               COLLATE Latin1_General_100_BIN2_UTF8
```

> **IMPORTANT**
> Use a collation with `_UTF8` suffix to ensure that UTF-8 text is properly converted to `VARCHAR` columns. `Latin1_General_100_BIN2_UTF8` provides the best performance in the queries that read data from Parquet files and cosmos Db containers.

2. Switch from master to `DataExplorationDB` using the following command. You can also use the UI control **use database** to switch your current database:

```
USE DataExplorationDB
```

3. From the 'DataExplorationDB', create utility objects such as credentials and data sources.

```
CREATE EXTERNAL DATA SOURCE ContosoLake
WITH ( LOCATION = 'https://contosolake.dfs.core.windows.net')
```

> **NOTE**
> An external data source can be created without a credential. If a credential does not exist, the caller's identity will be used to access the external data source.

4. Optionally, use the newly created 'DataExplorationDB' database to create a login for a user in DataExplorationDB that will access external data:

```
CREATE LOGIN data_explorer WITH PASSWORD = 'My Very Strong Password 1234!';
```

   Next create a database user in 'DataExplorationDB' for the above login and grant the `ADMINISTER DATABASE BULK OPERATIONS` permission.

```
CREATE USER data_explorer FOR LOGIN data_explorer;
GO
GRANT ADMINISTER DATABASE BULK OPERATIONS TO data_explorer;
GO
```

5. Explore the content of the file using the relative path and the data source:

```
SELECT
    TOP 100 *
FROM
    OPENROWSET(
            BULK '/users/NYCTripSmall.parquet',
            DATA_SOURCE = 'ContosoLake',
            FORMAT='PARQUET'
    ) AS [result]
```

6. **Publish** your changes to the workspace.

Data exploration database is just a simple placeholder where you can store your utility objects. Synapse SQL

pool enables you to do much more and create a Logical Data Warehouse - a relational layer built on top of Azure data sources. Learn more about building Logical Data Warehouse in this tutorial.

Next steps

Analyze data with a serverless Spark pool

Quickstart: Analyze with Data Explorer (Preview)

8/19/2022 • 2 minutes to read • Edit Online

In this article, you'll learn the basic steps to load and analyze data with Data Explorer for Azure Synapse.

Create a Data Explorer pool

1. In Synapse Studio, on the left-side pane, select **Manage** > **Data Explorer pools**.
2. Select **New**, and then enter the following details on the **Basics** tab:

SETTING	SUGGESTED VALUE	DESCRIPTION
Data Explorer pool name	contosodataexplorer	This is the name that the Data Explorer pool will have.
Workload	Compute optimized	This workload provides a higher CPU to SSD storage ratio.
Node size	Small (4 cores)	Set this to the smallest size to reduce costs for this quickstart

> **IMPORTANT**
> Note that there are specific limitations for the names that Data Explorer pools can use. Names must contain lowercase letters and numbers only, must be between 4 and 15 characters, and must start with a letter.

3. Select **Review + create** > **Create**. Your Data Explorer pool will start the provisioning process.

Create a Data Explorer database

1. In Synapse Studio, on the left-side pane, select **Data**.
2. Select **+** (Add new resource) > **Data Explorer database**, and paste the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Pool name	*contosodataexplorer*	The name of the Data Explorer pool to use
Name	*TestDatabase*	The database name must be unique within the cluster.
Default retention period	*365*	The time span (in days) for which it's guaranteed that the data is kept available to query. The time span is measured from the time that data is ingested.

SETTING	SUGGESTED VALUE	DESCRIPTION
Default cache period	*31*	The time span (in days) for which to keep frequently queried data available in SSD storage or RAM, rather than in longer-term storage.

3. Select **Create** to create the database. Creation typically takes less than a minute.

Ingest sample data and analyze with a simple query

1. In Synapse Studio, on the left-side pane, select **Develop**.
2. Under **KQL scripts**, Select **+** (Add new resource) > **KQL script**. On the right-side pane, you can name your script.
3. In the **Connect to** menu, select *contosodataexplorer*.
4. In the **Use database** menu, select *TestDatabase*.
5. Paste in the following command, and select **Run** to create a StormEvents table.

```
.create table StormEvents (StartTime: datetime, EndTime: datetime, EpisodeId: int, EventId: int,
State: string, EventType: string, InjuriesDirect: int, InjuriesIndirect: int, DeathsDirect: int,
DeathsIndirect: int, DamageProperty: int, DamageCrops: int, Source: string, BeginLocation: string,
EndLocation: string, BeginLat: real, BeginLon: real, EndLat: real, EndLon: real, EpisodeNarrative:
string, EventNarrative: string, StormSummary: dynamic)
```

> **TIP**
> Verify that the table was successfully created. On the left-side pane, select **Data**, select the *contosodataexplorer* more menu, and then select **Refresh**. Under *contosodataexplorer*, expand **Tables** and make sure that the *StormEvents* table appears in the list.

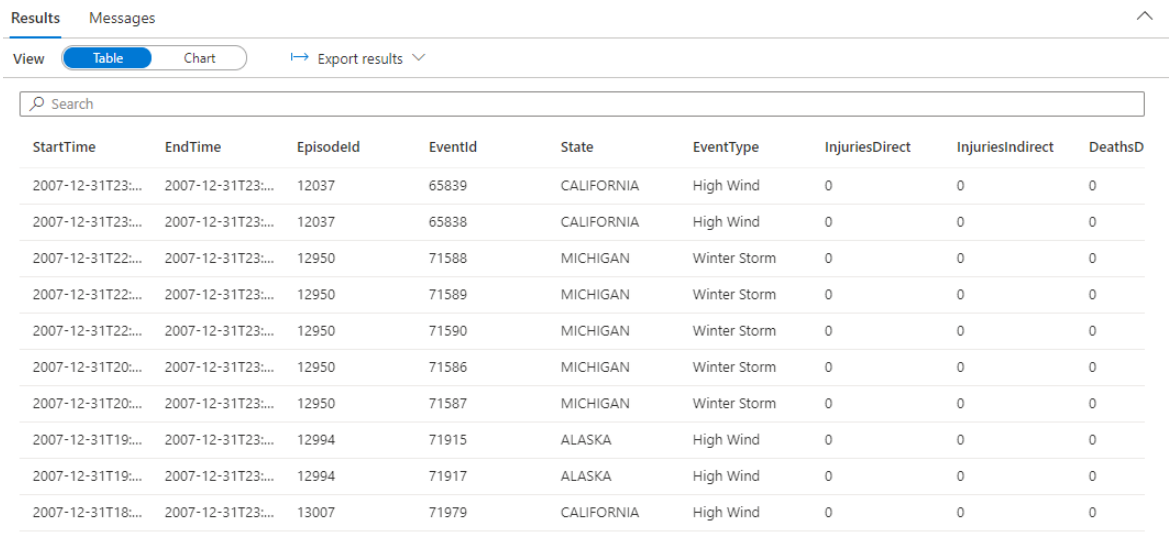
6. Paste in the following command, and select **Run** to ingest data into StormEvents table.

```
.ingest into table StormEvents
'https://kustosamplefiles.blob.core.windows.net/samplefiles/StormEvents.csv?sv=2019-12-
12&ss=b&srt=o&sp=r&se=2022-09-05T02:23:52Z&st=2020-09-
04T18:23:52Z&spr=https&sig=VrOfQMT1gUrHltJ8uhjYcCequEcfhjyyMX%2FSc3xsCy4%3D' with
(ignoreFirstRecord=true)
```

7. After ingestion completes, paste in the following query, select the query in the window, and select **Run**.

```
StormEvents
| sort by StartTime desc
| take 10
```

The query returns the following results from the ingested sample data.



StartTime	EndTime	EpisodeId	EventId	State	EventType	InjuriesDirect	InjuriesIndirect	DeathsD
2007-12-31T23:...	2007-12-31T23:...	12037	65839	CALIFORNIA	High Wind	0	0	0
2007-12-31T23:...	2007-12-31T23:...	12037	65838	CALIFORNIA	High Wind	0	0	0
2007-12-31T22:...	2007-12-31T23:...	12950	71588	MICHIGAN	Winter Storm	0	0	0
2007-12-31T22:...	2007-12-31T23:...	12950	71589	MICHIGAN	Winter Storm	0	0	0
2007-12-31T22:...	2007-12-31T23:...	12950	71590	MICHIGAN	Winter Storm	0	0	0
2007-12-31T20:...	2007-12-31T23:...	12950	71586	MICHIGAN	Winter Storm	0	0	0
2007-12-31T20:...	2007-12-31T23:...	12950	71587	MICHIGAN	Winter Storm	0	0	0
2007-12-31T19:...	2007-12-31T23:...	12994	71915	ALASKA	High Wind	0	0	0
2007-12-31T19:...	2007-12-31T23:...	12994	71917	ALASKA	High Wind	0	0	0
2007-12-31T18:...	2007-12-31T23:...	13007	71979	CALIFORNIA	High Wind	0	0	0

Next steps

- Tutorial: Use KQL queries
- Monitor Data Explorer pools

Analyze with Apache Spark

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, you'll learn the basic steps to load and analyze data with Apache Spark for Azure Synapse.

Create a serverless Apache Spark pool

1. In Synapse Studio, on the left-side pane, select **Manage** > **Apache Spark pools**.
2. Select **New**
3. For **Apache Spark pool name** enter **Spark1**.
4. For **Node size** enter **Small**.
5. For **Number of nodes** Set the minimum to 3 and the maximum to 3
6. Select **Review + create** > **Create**. Your Apache Spark pool will be ready in a few seconds.

Understanding serverless Apache Spark pools

A serverless Spark pool is a way of indicating how a user wants to work with Spark. When you start using a pool, a Spark session is created if needed. The pool controls how many Spark resources will be used by that session and how long the session will last before it automatically pauses. You pay for spark resources used during that session not for the pool itself. In this way a Spark pool lets you work with Spark, without having to worry managing clusters. This is similar to how a serverless SQL pool works.

Analyze NYC Taxi data with a Spark pool

> **NOTE**
>
> Make sure you have placed the sample data in the primary storage account.

1. In Synapse Studio, go to the **Develop** hub.
2. Create a new notebook.
3. Create a new code cell and paste the following code in that cell:

```
%%pyspark
df = spark.read.load('abfss://users@contosolake.dfs.core.windows.net/NYCTripSmall.parquet',
format='parquet')
display(df.limit(10))
```

4. Modify the load URI, so it references the sample file in your storage account according to the abfss URI scheme.
5. In the notebook, in the **Attach to** menu, choose the **Spark1** serverless Spark pool that we created earlier.
6. Select **Run** on the cell. Synapse will start a new Spark session to run this cell if needed. If a new Spark session is needed, initially it will take about two seconds to be created.
7. If you just want to see the schema of the dataframe run a cell with the following code:

```
%%pyspark
df.printSchema()
```

Load the NYC Taxi data into the Spark nyctaxi database

Data is available via the dataframe named **df**. Load it into a Spark database named **nyctaxi**.

1. Add a new code cell to the notebook, and then enter the following code:

```
%%pyspark
spark.sql("CREATE DATABASE IF NOT EXISTS nyctaxi")
df.write.mode("overwrite").saveAsTable("nyctaxi.trip")
```

Analyze the NYC Taxi data using Spark and notebooks

1. Create a new code cell and enter the following code.

```
%%pyspark
df = spark.sql("SELECT * FROM nyctaxi.trip")
display(df)
```

2. Run the cell to show the NYC Taxi data we loaded into the **nyctaxi** Spark database.

3. Create a new code cell and enter the following code. We will analyze this data and save the results into a table called **nyctaxi.passengercountstats**.

```
%%pyspark
df = spark.sql("""
   SELECT PassengerCount,
       SUM(TripDistanceMiles) as SumTripDistance,
       AVG(TripDistanceMiles) as AvgTripDistance
   FROM nyctaxi.trip
   WHERE TripDistanceMiles > 0 AND PassengerCount > 0
   GROUP BY PassengerCount
   ORDER BY PassengerCount
""")
display(df)
df.write.saveAsTable("nyctaxi.passengercountstats")
```

4. In the cell results, select **Chart** to see the data visualized.

Next steps

Analyze data with dedicated SQL pool

Analyze data with dedicated SQL pools

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, you'll use the NYC Taxi data to explore a dedicated SQL pool's capabilities.

Create a dedicated SQL pool

1. In Synapse Studio, on the left-side pane, select **Manage** > **SQL pools** under **Analytics pools**.
2. Select **New**
3. For **Dedicated SQL pool name** select **SQLPOOL1**
4. For **Performance level** choose **DW100C**
5. Select **Review + create** > **Create**. Your dedicated SQL pool will be ready in a few minutes.

Your dedicated SQL pool is associated with a SQL database that's also called **SQLPOOL1**.

1. Navigate to **Data** > **Workspace**.
2. You should see a database named **SQLPOOL1**. If you do not see it, click **Refresh**.

A dedicated SQL pool consumes billable resources as long as it's active. You can pause the pool later to reduce costs.

> **NOTE**
> When creating a new dedicated SQL pool (formerly SQL DW) in your workspace, the dedicated SQL pool provisioning page will open. Provisioning will take place on the logical SQL server.

Load the NYC Taxi Data into SQLPOOL1

1. In Synapse Studio, navigate to the **Develop** hub, click the **+** button to add new resource, then create new SQL script.
2. Select the pool 'SQLPOOL1' (pool created in STEP 1 of this tutorial) in **Connect to** drop down list above the script.
3. Enter the following code:

```
IF NOT EXISTS (SELECT * FROM sys.objects O JOIN sys.schemas S ON O.schema_id = S.schema_id WHERE
O.NAME = 'NYCTaxiTripSmall' AND O.TYPE = 'U' AND S.NAME = 'dbo')
CREATE TABLE dbo.NYCTaxiTripSmall
    (
     [DateID] int,
     [MedallionID] int,
     [HackneyLicenseID] int,
     [PickupTimeID] int,
     [DropoffTimeID] int,
     [PickupGeographyID] int,
     [DropoffGeographyID] int,
     [PickupLatitude] float,
     [PickupLongitude] float,
     [PickupLatLong] nvarchar(4000),
     [DropoffLatitude] float,
     [DropoffLongitude] float,
     [DropoffLatLong] nvarchar(4000),
     [PassengerCount] int,
     [TripDurationSeconds] int,
     [TripDistanceMiles] float,
     [PaymentType] nvarchar(4000),
     [FareAmount] numeric(19,4),
     [SurchargeAmount] numeric(19,4),
     [TaxAmount] numeric(19,4),
     [TipAmount] numeric(19,4),
     [TollsAmount] numeric(19,4),
     [TotalAmount] numeric(19,4)
    )
WITH
    (
    DISTRIBUTION = ROUND_ROBIN,
     CLUSTERED COLUMNSTORE INDEX
     -- HEAP
    )
GO

COPY INTO dbo.NYCTaxiTripSmall
(DateID 1, MedallionID 2, HackneyLicenseID 3, PickupTimeID 4, DropoffTimeID 5,
PickupGeographyID 6, DropoffGeographyID 7, PickupLatitude 8, PickupLongitude 9,
PickupLatLong 10, DropoffLatitude 11, DropoffLongitude 12, DropoffLatLong 13,
PassengerCount 14, TripDurationSeconds 15, TripDistanceMiles 16, PaymentType 17,
FareAmount 18, SurchargeAmount 19, TaxAmount 20, TipAmount 21, TollsAmount 22,
TotalAmount 23)
FROM 'https://contosolake.dfs.core.windows.net/users/NYCTripSmall.parquet'
WITH
(
    FILE_TYPE = 'PARQUET'
    ,MAXERRORS = 0
    ,IDENTITY_INSERT = 'OFF'
)
```

4. Click the **Run** button to execute the script.
5. This script will finish in less than 60 seconds. It loads 2 million rows of NYC Taxi data into a table called `dbo.NYCTaxiTripSmall`.

Explore the NYC Taxi data in the dedicated SQL pool

1. In Synapse Studio, go to the **Data** hub.
2. Go to **SQLPOOL1** > **Tables**.
3. Right-click the **dbo.NYCTaxiTripSmall** table and select **New SQL Script** > **Select TOP 100 Rows**.
4. Wait while a new SQL script is created and runs.

5. Notice that at the top of the SQL script **Connect to** is automatically set to the SQL pool called **SQLPOOL1**.

6. Replace the text of the SQL script with this code and run it.

```
SELECT PassengerCount,
      SUM(TripDistanceMiles) as SumTripDistance,
      AVG(TripDistanceMiles) as AvgTripDistance
FROM  dbo.NYCTaxiTripSmall
WHERE TripDistanceMiles > 0 AND PassengerCount > 0
GROUP BY PassengerCount
ORDER BY PassengerCount;
```

   This query shows how the total trip distances and average trip distance relate to the number of passengers.

7. In the SQL script result window, change the **View** to **Chart** to see a visualization of the results as a line chart.

Next steps

Analyze data in an Azure Storage account

Analyze data in a storage account

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, you'll learn how to analyze data located in a storage account.

Overview

So far, we've covered scenarios where data resides in databases in the workspace. Now we'll show you how to work with files in storage accounts. In this scenario, we'll use the primary storage account of the workspace and container that we specified when creating the workspace.

- The name of the storage account: **contosolake**
- The name of the container in the storage account: **users**

Create CSV and Parquet files in your storage account

Run the following code in a notebook in a new code cell. It creates a CSV file and a parquet file in the storage account.

```
%%pyspark
df = spark.sql("SELECT * FROM nyctaxi.passengercountstats")
df = df.repartition(1) # This ensures we'll get a single file during write()
df.write.mode("overwrite").csv("/NYCTaxi/PassengerCountStats_csvformat")
df.write.mode("overwrite").parquet("/NYCTaxi/PassengerCountStats_parquetformat")
```

Analyze data in a storage account

You can analyze the data in your workspace default ADLS Gen2 account or you can link an ADLS Gen2 or Blob storage account to your workspace through "**Manage**" > "**Linked Services**" > "**New**" (The steps below will refer to the primary ADLS Gen2 account).

1. In Synapse Studio, go to the **Data** hub, and then select **Linked**.
2. Go to **Azure Data Lake Storage Gen2** > **myworkspace (Primary - contosolake)**.
3. Select **users (Primary)**. You should see the **NYCTaxi** folder. Inside you should see two folders called **PassengerCountStats_csvformat** and **PassengerCountStats_parquetformat**.
4. Open the **PassengerCountStats_parquetformat** folder. Inside, you'll see a parquet file with a name like `part-00000-2638e00c-0790-496b-a523-578da9a15019-c000.snappy.parquet`.
5. Right-click **.parquet**, then select **New notebook**, then select **Load to DataFrame**. A new notebook is created with a cell like this:

   ```
   %%pyspark
   abspath =
   'abfss://users@contosolake.dfs.core.windows.net/NYCTaxi/PassengerCountStats_parquetformat/part-00000-
   1f251a58-d8ac-4972-9215-8d528d490690-c000.snappy.parquet'
   df = spark.read.load(abspath, format='parquet')
   display(df.limit(10))
   ```

6. Attach to the Spark pool named **Spark1**. Run the cell.
7. Select back to the **users** folder. Right-click the **.parquet** file again, and then select **New SQL script** > **SELECT TOP 100 rows**. It creates a SQL script like this:

```
SELECT
    TOP 100 *
FROM OPENROWSET(
    BULK
'https://contosolake.dfs.core.windows.net/users/NYCTaxi/PassengerCountStats_parquetformat/part-00000-
1f251a58-d8ac-4972-9215-8d528d490690-c000.snappy.parquet',
    FORMAT='PARQUET'
) AS [result]
```

In the script window, make sure the **Connect to** field is set to the **Built-in** serverless SQL pool.

8. Run the script.

Next steps

Orchestrate activities with pipelines

Integrate with pipelines

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, you'll learn how to integrate pipelines and activities using Synapse Studio.

Create a pipeline and add a notebook activity

1. In Synapse Studio, go to the **Integrate** hub.
2. Select **+** > **Pipeline** to create a new pipeline. Click on the new pipeline object to open the Pipeline designer.
3. Under **Activities**, expand the **Synapse** folder, and drag a **Notebook** object into the designer.
4. Select the **Settings** tab of the Notebook activity properties. Use the drop-down list to select a notebook from your current Synapse workspace.

Schedule the pipeline to run every hour

1. In the pipeline, select **Add trigger** > **New/edit**.
2. In **Choose trigger**, select **New**, and set the **Recurrence** to "every 1 hour".
3. Select **OK**.
4. Select **Publish All**.

Forcing a pipeline to run immediately

Once the pipeline is published, you may want to run it immediately without waiting for an hour to pass.

1. Open the pipeline.
2. Click **Add trigger** > **Trigger now**.
3. Select **OK**.

Monitor pipeline execution

1. Go to the **Monitor** hub.
2. Select **Pipeline runs** to monitor pipeline execution progress.
3. In this view you can switch between tabular **List** display a graphical **Gantt** chart.
4. Click on a pipeline name to see the status of activities in that pipeline.

Next steps

Visualize data with Power BI

Visualize data with Power BI

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, you'll learn how to create a Power BI workspace, link your Azure Synapse workspace, and create a Power BI data set that utilizes data in your Azure Synapse workspace.

Prerequisites

To complete this tutorial, install Power BI Desktop.

Overview

From the NYC Taxi data, we created aggregated datasets in two tables:

- **nyctaxi.passengercountstats**
- **SQLDB1.dbo.PassengerCountStats**

You can link a Power BI workspace to your Azure Synapse workspace. This capability allows you to easily get data into your Power BI workspace. You can edit your Power BI reports directly in your Azure Synapse workspace.

Create a Power BI workspace

1. Sign in to powerbi.microsoft.com.
2. Click on **Workspaces**, then select **Create a workspace**. Create a new Power BI workspace named **NYCTaxiWorkspace1** or similar, since this name must be unique.

Link your Azure Synapse workspace to your new Power BI workspace

1. In Synapse Studio, go to **Manage** > **Linked Services**.
2. Select **New** > **Connect to Power BI**.
3. Set **Name** to **NYCTaxiWorkspace1**.
4. Set **Workspace name** to the Power BI workspace you created above, similar to **NYCTaxiWorkspace1**.
5. Select **Create**.

Create a Power BI dataset that uses data in your Azure Synapse workspace

1. In Synapse Studio, go to **Develop** > **Power BI**.
2. Go to **NYCTaxiWorkspace1** > **Power BI datasets** and select **New Power BI dataset**. Click **Start**.
3. Select the **SQLPOOL1** data source, click **Continue**.
4. Click **Download** to download the .pbids file for your **NYCTaxiWorkspace1SQLPOOL1.pbids** file. Click **Continue**.
5. Open the downloaded **.pbids** file. Power BI Desktop opens and automatically connects to **SQLDB1** in your Azure Synapse workspace.
6. If you see a dialog box appear called **SQL Server database**:
   a. Select **Microsoft account**.
   b. Select **Sign in** and sign in to your account.
   c. Select **Connect**.
7. After the **Navigator** dialog box opens, check the **PassengerCountStats** table and select **Load**.
8. After the **Connection settings** dialog box appears, select **DirectQuery** > **OK**.
9. Select the **Report** button on the left side.

10. Under **Visualizations**, click to the line chart icon to add a **Line chart** to your report.
    a. Under **Fields**, drag the **PassengerCount** column to **Visualizations** > **Axis**.
    b. Drag the **SumTripDistance** and **AvgTripDistance** columns to **Visualizations** > **Values**.
11. On the **Home** tab, select **Publish**.
12. Select **Save** to save your changes.
13. Choose the file name **PassengerAnalysis.pbix**, and then select **Save**.
14. In the **Publish to Power BI** window, under **Select a destination**, choose your **NYCTaxiWorkspace1**, and then click **Select**.
15. Wait for publishing to finish.

Configure authentication for your dataset

1. Open powerbi.microsoft.com and **Sign in**.
2. On the left side, under **Workspaces**, select the **NYCTaxiWorkspace1** workspace.
3. Inside that workspace, locate a dataset called **Passenger Analysis** and a report called **Passenger Analysis**.
4. Hover over the **PassengerAnalysis** dataset, select the ellipsis (...) button, and then select **Settings**.
5. In **Data source credentials**, click **Edit**, set the **Authentication method** to **OAuth2**, and then select **Sign in**.

Edit a report in Synapse Studio

1. Go back to Synapse Studio and select **Close and refresh**.
2. Go to the **Develop** hub.
3. To the right of the **Power BI** layer, ellipsis (...) button, and click **refresh** to refresh the **Power BI reports** node.
4. Under **Power BI** you should see:
    - In **NYCTaxiWorkspace1** > **Power BI datasets**, a new dataset called **PassengerAnalysis**.
    - Under **NYCTaxiWorkspace1** > **Power BI reports**, a new report called **PassengerAnalysis**.
5. Select the **PassengerAnalysis** report. The report opens and you can edit it directly within Synapse Studio.

Next steps

Monitor

Monitor your Synapse Workspace

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, you'll learn how to monitor activities in your Synapse workspace. You can monitor current and historical activities for SQL, Apache Spark, and Pipelines.

Introduction to the Monitor Hub

Open Synapse Studio and navigate to the **Monitor** hub. Here, you can see a history of all the activities taking place in the workspace and which ones are active now.

- Under **Integration**, you can monitor pipelines, triggers, and integration runtimes.
- Under **Activities**, you can monitor Spark and SQL activities.

Integration

1. Navigate to **Integration** > **Pipeline runs**. In this view, you can see every time a pipeline has run in your workspace.
2. Find the pipeline that you ran in the previous step and click on its **Pipeline name** to view the details.
3. Click **Breadcrumb bar** near the top of Synapse Studio, click **All pipeline runs** to return to the previous view.

Data Explorer activities

1. Navigate to **Activities** > **KQL requests**.
2. In this view you can see KQL requests.
3. Select a **Pool** to monitor from the **Pool** filter. Now you can see all KQL requests that are running or have run in your workspace in that pool.
4. Find a specific KQL request and click on the **More** link to see the full text of the KQL request.

Apache Spark activities

1. Navigate to **Activities** > **Apache Spark applications**. Now you can see all the Spark applications that are running or have run in your workspace.
2. Find an application that is no longer running and click on its **Application name**. Now you can see the details of the spark application.
3. If you are familiar with Apache Spark, you can find the standard Apache Spark history server UI by clicking on **Spark history server**.

SQL activities

1. Navigate to **Activities** > **SQL requests**.
2. In this view you can see SQL requests.
3. Select a **Pool** to monitor from the **Pool** filter. Now you can see all SQL requests that are running or have run in your workspace in that pool.
4. Find a specific SQL request and click on the **More** link to see the full text of the SQL request.

> **NOTE**
>
> SQL requests submitted via the Synapse Studio in a workspace enabled dedicated SQL pool (formerly SQL DW) can be viewed in the Monitor hub. For all other monitoring activities, you can go to Azure portal dedicated SQL pool (formerly SQL DW) monitoring.

Next steps

Explore the Knowledge center

Explore the Synapse Knowledge center

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, you'll learn how to use the Synapse Studio **Knowledge center**.

Finding the Knowledge center

There are two ways of finding the **Knowledge center** in Synapse Studio:

1. In the Home hub, near the top-right of the page click on **Learn**.
2. In the menu bar at the top, click **?** and then **Knowledge center**.

Pick either method and open the **Knowledge center**.

Exploring the Knowledge center

Once it is visible, you will see that the **Knowledge center** allows you to do three things:

- **Use samples immediately**. If you want a quick example of how Synapse works, choose this option.
- **Browse gallery**. This option lets you link sample data sets and add sample code in the form of SQL scripts, notebooks, and pipelines.
- **Tour Synapse Studio**. This option takes you on a brief tour of the basic parts of Synapse Studio. This is useful if you have never used Synapse Studio before.

Use samples immediately: Three samples to help you get started fast

There are three items in this section:

- Explore sample data with Spark
- Query data with SQL
- Create external table with SQL

1. In the **Knowledge center**, click **Use samples immediately**.
2. Select **Query data with SQL**.
3. Click **Use sample**.
4. A new sample SQL script will open.
5. Scroll to the first query (lines 28 to 32) and select the query text.
6. Click Run. It will run only code you have selected.

Gallery: A collection of sample datasets and sample code

1. Go to the **Knowledge center**, click **Browse gallery**.
2. Select the **SQL scripts** tab at the top.
3. Select **Load the New York Taxicab dataset** Data ingestion sample, click **Continue**.
4. Under **SQL pool**, choose **Select an existing pool** and select **SQLPOOL1**, and select the **SQLPOOL1** database you created earlier.
5. Click **Open Script**.

6. A new sample SQL script will open.
7. Click **Run**
8. This will create several tables for all of the NYC Taxi data and load them using the T-SQL COPY command. If you had created these tables in the previous quick start steps, select and execute only code to CREATE and COPY for tables that do not exist.

> **NOTE**
> When using the sample gallery for SQL script with a dedicated SQL pool (formerly SQL DW), you will only be able to use an existing dedicated SQL pool (formerly SQL DW).

Next steps

Add an administrator

Add an administrator to your Synapse workspace

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, you'll learn how to add an administrator to your Synapse workspace. This user will have full control over the workspace.

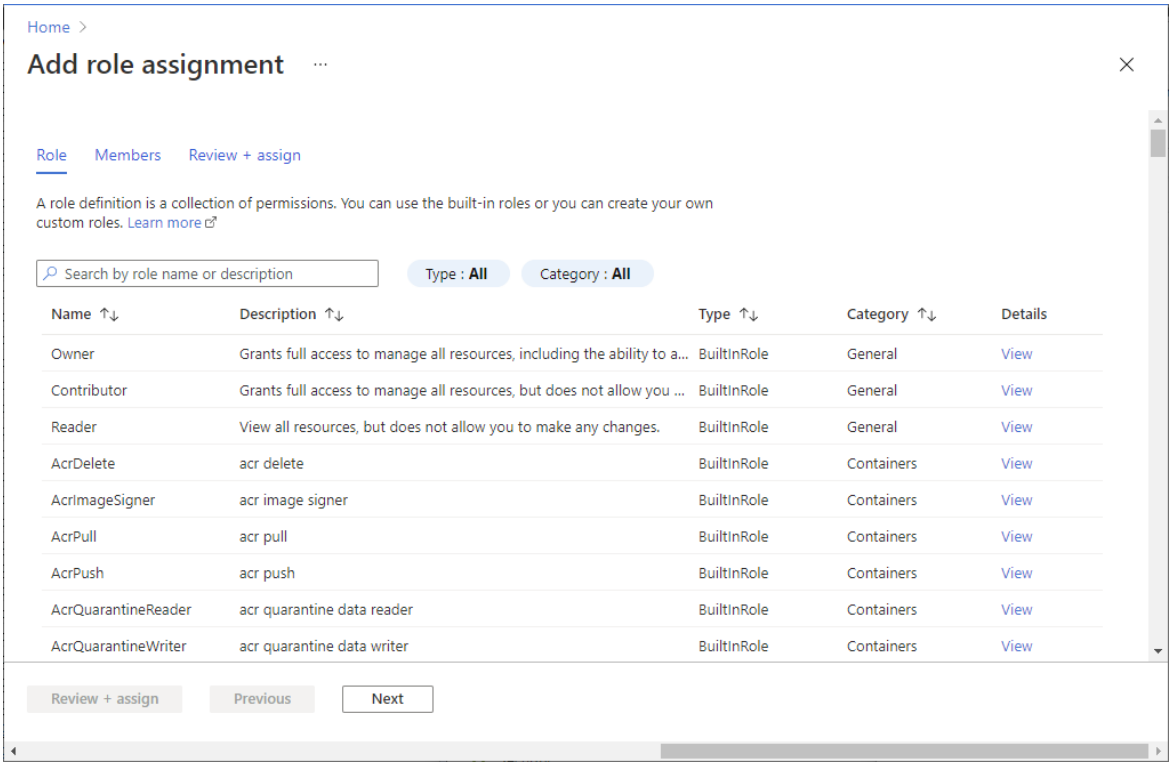
Overview

So far in the get started guide, we've focused on activities *you* do in the workspace. Because you created the workspace in STEP 1, you are an administrator of the Synapse workspace. Now, we will make another user Ryan (`ryan@contoso.com`) an administrator. When we are done, Ryan will be able to do everything you can do in the workspace.

Azure RBAC: Owner role for the workspace

1. Open the Azure portal and open your Synapse workspace.
2. On the left side, select **Access control (IAM)**.
3. Select **Add** > **Add role assignment** to open the Add role assignment page.
4. Assign the following role. For detailed steps, see Assign Azure roles using the Azure portal.

SETTING	VALUE
Role	Owner
Assign access to	USER
Member	ryan@contoso.com



5. Select **Save**.

Synapse RBAC: Synapse Administrator role for the workspace

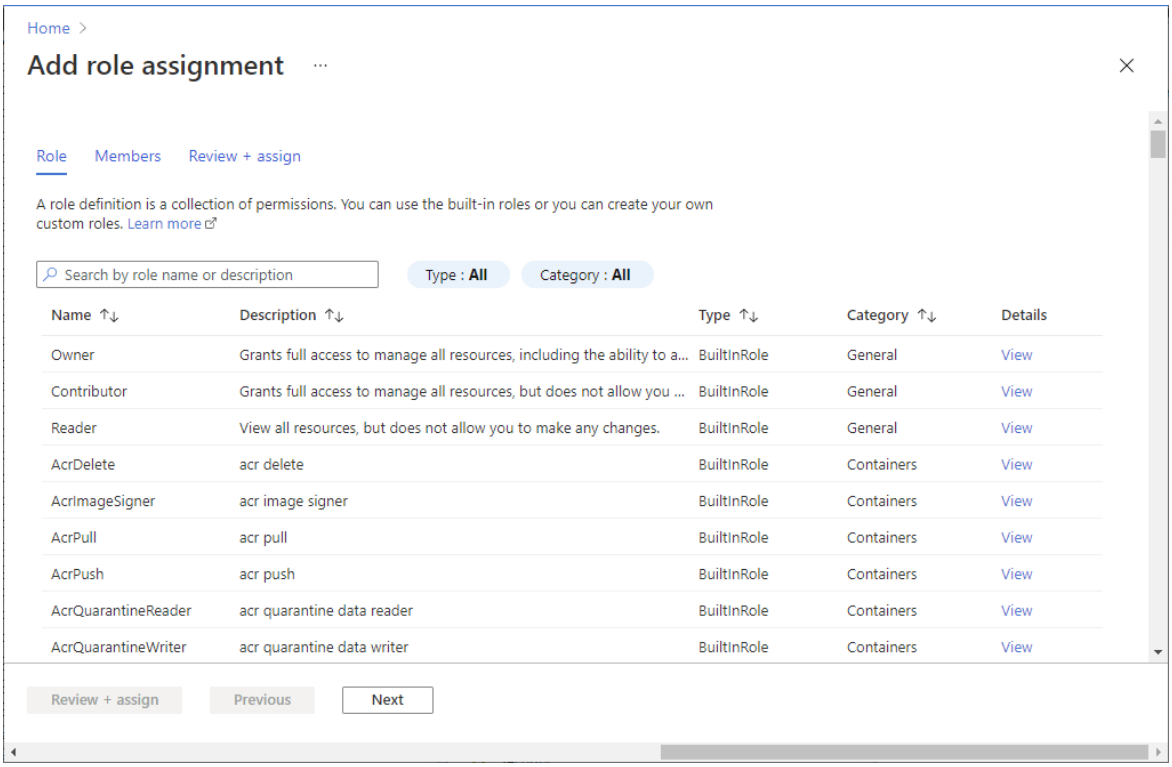
Assign to `ryan@contoso.com` to Synapse RBAC **Synapse Administrator** role on the workspace.

1. Open your workspace in Synapse Studio.
2. On the left side, select **Manage** to open the Manage hub.
3. Under **Security**, select **Access control**.
4. Select **Add**.
5. Leave **Scope** set to **Workspace**.
6. Add `ryan@contoso.com` to the **Synapse Administrator** role.
7. Then select **Apply**.

Azure RBAC: Role assignments on the workspace's primary storage account

1. Open the workspace's primary storage account in the Azure portal.
2. On the left side, select **Access control (IAM)**.
3. Select **Add** > **Add role assignment** to open the Add role assignment page.
4. Assign the following role. For detailed steps, see Assign Azure roles using the Azure portal.

SETTING	VALUE
Role 1	Owner
Role 2	Azure Storage Blob Data Contributor
Assign access to	USER
Member	ryan@contoso.com



Dedicated SQL pools: db_owner role

Assign `ryan@contoso.com` to the **db_owner** on each dedicated SQL pool in the workspace.

```
CREATE USER [ryan@contoso.com] FROM EXTERNAL PROVIDER;
EXEC sp_addrolemember 'db_owner', 'ryan@contoso.com'
```

Next steps

- Get started with Azure Synapse Analytics
- Create a workspace
- Use serverless SQL pool

Quickstart: Create a Synapse workspace

8/19/2022 • 2 minutes to read • Edit Online

This quickstart describes the steps to create an Azure Synapse workspace by using the Azure portal.

Create a Synapse workspace

1. Open the Azure portal, and at the top search for **Synapse**.
2. In the search results, under **Services**, select **Azure Synapse Analytics**.
3. Select **Add** to create a workspace.
4. In the **Basics** tab, give the workspace a unique name. We'll use **mysworkspace** in this document
5. You need an ADLSGEN2 account to create a workspace. The simplest choice is to create a new one. If you want to re-use an existing one you'll need to perform some additional configuration.
6. OPTION 1 Creating a new ADLSGEN2 account
   a. Under **Select Data Lake Storage Gen 2**, click **Create New** and name it **contosolake**.
   b. Under **Select Data Lake Storage Gen 2**, click **File System** and name it **users**.
7. OPTION 2 See the **Prepare a Storage Account** instructions at the bottom of this document.
8. Your Azure Synapse workspace will use this storage account as the "primary" storage account and the container to store workspace data. The workspace stores data in Apache Spark tables. It stores Spark application logs under a folder called **/synapse/workspacename**.
9. Select **Review + create** > **Create**. Your workspace is ready in a few minutes.

> **NOTE**
> After creating your Azure Synapse workspace, you will not be able to move the workspace to another Azure Active Directory tenant. If you do so through subscription migration or other actions, you may lose access to the artifacts within the workspace.

Open Synapse Studio

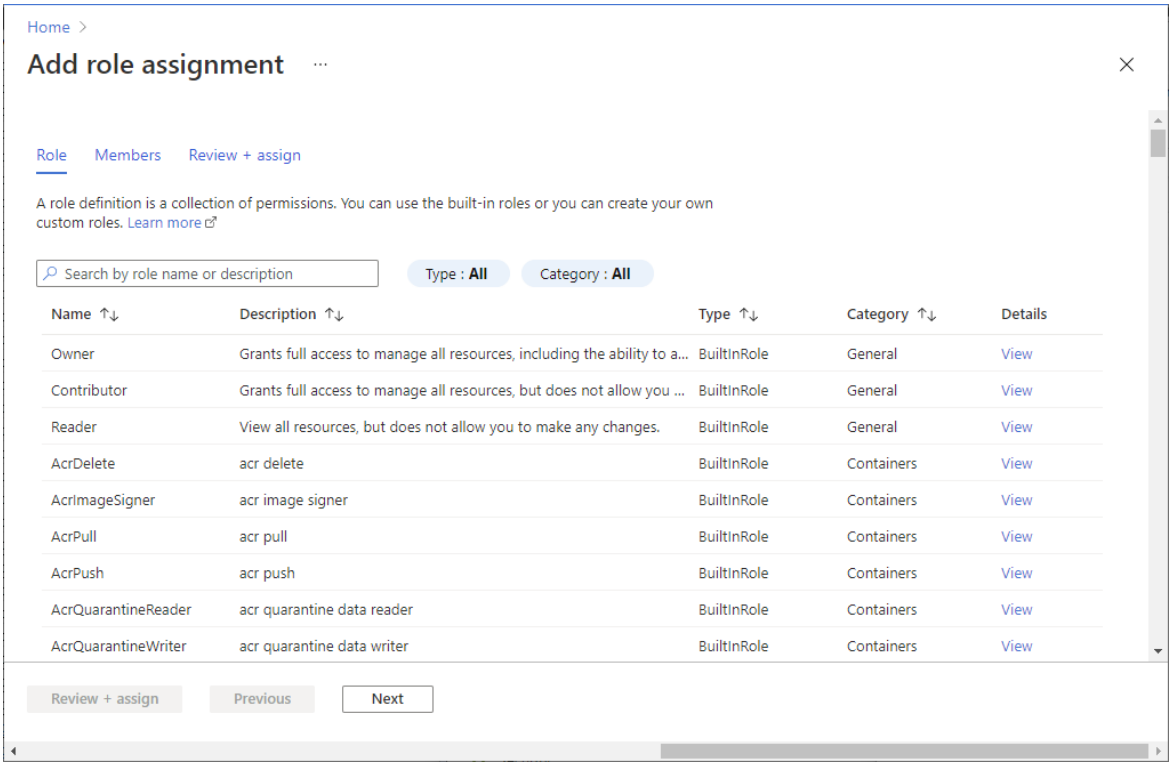
After your Azure Synapse workspace is created, you have two ways to open Synapse Studio:

- Open your Synapse workspace in the Azure portal. On the top of the **Overview** section, select **Launch Synapse Studio**.
- Go to the `https://web.azuresynapse.net` and sign in to your workspace.

Prepare an existing storage account for use with Azure Synapse Analytics

1. Open the Azure portal.
2. Navigate to an existing ADLSGEN2 storage account
3. Select **Access control (IAM)**.
4. Select **Add** > **Add role assignment** to open the Add role assignment page.
5. Assign the following role. For detailed steps, see Assign Azure roles using the Azure portal.

SETTING	VALUE
Role	Owner and Storage Blob Data Owner
Assign access to	USER
Members	your user name



6. On the left pane, select **Containers** and create a container.
7. You can give the container any name. In this document, we'll name the container **users**.
8. Accept the default setting **Public access level**, and then select **Create**.

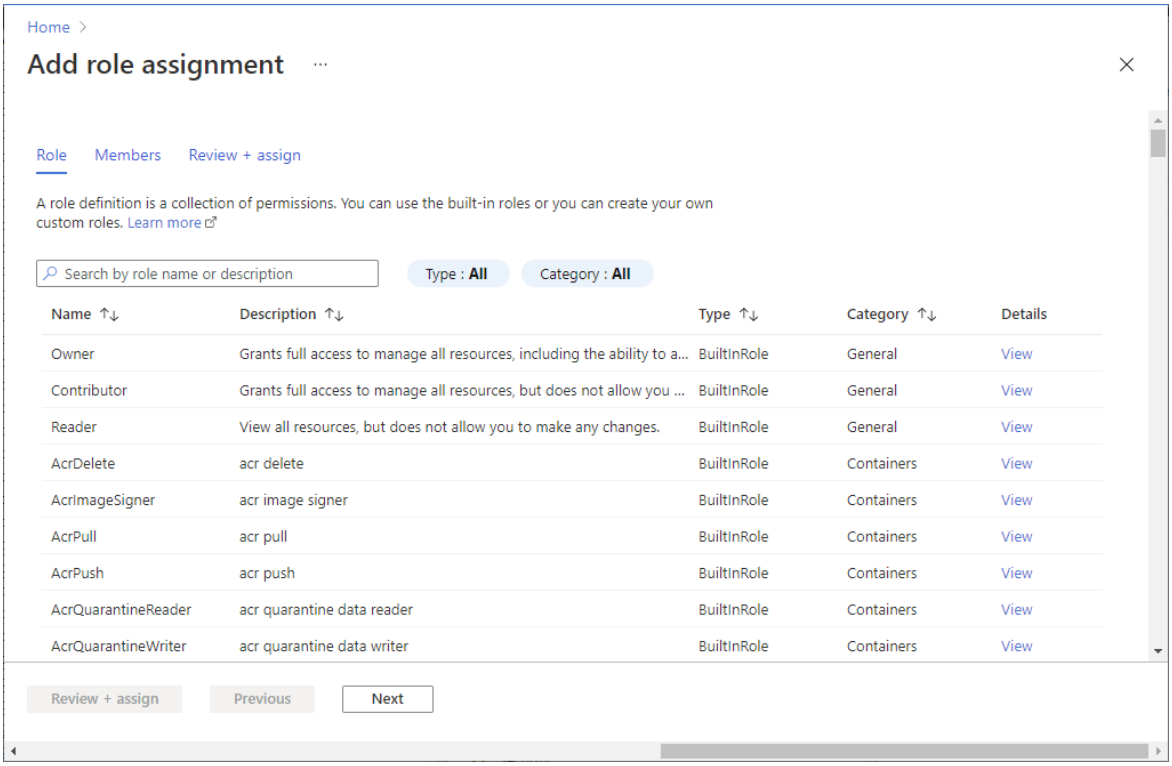
Configure access to the storage account from your workspace

Managed identities for your Azure Synapse workspace might already have access to the storage account. Follow these steps to make sure:

1. Open the Azure portal and the primary storage account chosen for your workspace.
2. Select **Access control (IAM)**.
3. Select **Add** > **Add role assignment** to open the Add role assignment page.
4. Assign the following role. For detailed steps, see Assign Azure roles using the Azure portal.

SETTING	VALUE
Role	Storage Blob Data Contributor
Assign access to	MANAGEDIDENTITY
Members	myworkspace

> **NOTE**
>
> The managed identity name is also the workspace name.



5. Select **Save**.

Next steps

- Create a dedicated SQL pool
- Create a serverless Apache Spark pool
- Use serverless SQL pool

Quickstart: Create an Azure synapse workspace with Azure CLI

8/19/2022 • 3 minutes to read • Edit Online

The Azure CLI is Azure's command-line experience for managing Azure resources. You can use it in your browser with Azure Cloud Shell. You can also install it on macOS, Linux, or Windows and run it from the command line.

In this quickstart, you learn to create a Synapse workspace by using the Azure CLI.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Download and install jq, a lightweight and flexible command-line JSON processor
- Azure Data Lake Storage Gen2 storage account

  > **IMPORTANT**
  > The Azure Synapse workspace needs to be able to read and write to the selected ADLS Gen2 account. In addition, for any storage account that you link as the primary storage account, you must have enabled **hierarchical namespace** at the creation of the storage account, as described on the Create a Storage Account page.

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Create an Azure Synapse workspace using the Azure CLI

1. Define necessary environment variables to create resources for Azure Synapse workspace.

ENVIRONMENT VARIABLE NAME	DESCRIPTION
StorageAccountName	Name for your existing ADLS Gen2 storage account.

ENVIRONMENT VARIABLE NAME	DESCRIPTION
StorageAccountResourceGroup	Name of your existing ADLS Gen2 storage account resource group.
FileShareName	Name of your existing storage file system.
SynapseResourceGroup	Choose a new name for your Azure Synapse resource group.
Region	Choose one of the Azure regions.
SynapseWorkspaceName	Choose a unique name for your new Azure Synapse Workspace.
SqlUser	Choose a value for a new username.
SqlPassword	Choose a secure password.

2. Create a resource group as a container for your Azure Synapse workspace:

```
az group create --name $SynapseResourceGroup --location $Region
```

3. Create an Azure Synapse Workspace:

```
az synapse workspace create \
  --name $SynapseWorkspaceName \
  --resource-group $SynapseResourceGroup \
  --storage-account $StorageAccountName \
  --file-system $FileShareName \
  --sql-admin-login-user $SqlUser \
  --sql-admin-login-password $SqlPassword \
  --location $Region
```

4. Get Web and Dev URL for Azure Synapse Workspace:

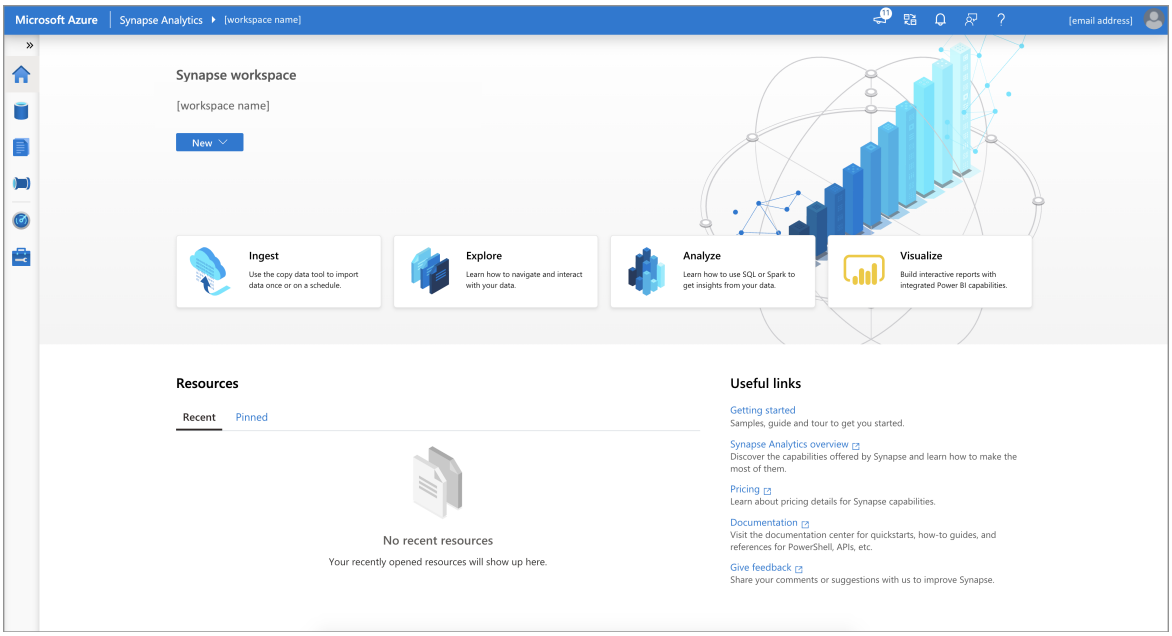
```
WorkspaceWeb=$(az synapse workspace show --name $SynapseWorkspaceName --resource-group
$SynapseResourceGroup | jq -r '.connectivityEndpoints | .web')

WorkspaceDev=$(az synapse workspace show --name $SynapseWorkspaceName --resource-group
$SynapseResourceGroup | jq -r '.connectivityEndpoints | .dev')
```

5. Create a Firewall Rule to allow your access to Azure Synapse Workspace from your machine:

```
ClientIP=$(curl -sb -H "Accept: application/json" "$WorkspaceDev" | jq -r '.message')
ClientIP=${ClientIP##'Client Ip address : '}
echo "Creating a firewall rule to enable access for IP address: $ClientIP"

az synapse workspace firewall-rule create --end-ip-address $ClientIP --start-ip-address $ClientIP --
name "Allow Client IP" --resource-group $SynapseResourceGroup --workspace-name $SynapseWorkspaceName
```

6. Open the Azure Synapse Workspace Web URL address stored in environment variable `WorkspaceWeb` to access your workspace:

```
echo "Open your Azure Synapse Workspace Web URL in the browser: $WorkspaceWeb"
```



7. Once deployed, additional permissions are required.

- In the Azure portal, assign other users of the workspace to the **Contributor** role in the workspace. For detailed steps, see Assign Azure roles using the Azure portal.
- Assign other users the appropriate **Synapse RBAC roles** using Synapse Studio.
- A member of the **Owner** role of the Azure Storage account must assign the **Storage Blob Data Contributor** role to the Azure Synapse workspace MSI and other users.

Clean up resources

Follow the steps below to delete the Azure Synapse workspace.

> **WARNING**
>
> Deleting an Azure Synapse workspace will remove the analytics engines and the data stored in the database of the contained SQL pools and workspace metadata. It will no longer be possible to connect to the SQL or Apache Spark endpoints. All code artifacts will be deleted (queries, notebooks, job definitions and pipelines).
>
> Deleting the workspace will **not** affect the data in the Data Lake Store Gen2 linked to the workspace.

If you want to delete the Azure Synapse workspace, complete the following command:

```
az synapse workspace delete --name $SynapseWorkspaceName --resource-group $SynapseResourceGroup
```

Next steps

Next, you can create SQL pools or create Apache Spark pools to start analyzing and exploring your data.

Quickstart: Create an Azure synapse workspace with Azure PowerShell

8/19/2022 • 3 minutes to read • Edit Online

Azure PowerShell is a set of cmdlets for managing Azure resources directly from PowerShell. You can use it in your browser with Azure Cloud Shell. You can also install it on macOS, Linux, or Windows.

In this quickstart, you learn to create a Synapse workspace using Azure PowerShell.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Data Lake Storage Gen2 storage account

> **IMPORTANT**
> The Azure Synapse workspace needs to be able to read and write to the selected ADLS Gen2 account. For any storage account that you link as the primary storage account, you must enable **hierarchical namespace** at the creation of the storage account as described in Create a storage account.

If you choose to use Cloud Shell, see Overview of Azure Cloud Shell for more information.

Install the Azure PowerShell module locally

If you choose to use PowerShell locally, this article requires that you install the Az PowerShell module and connect to your Azure account using the Connect-AzAccount cmdlet. For more information about installing the Az PowerShell module, see Install Azure PowerShell.

For more information about authentication with Azure PowerShell, see Sign in with Azure PowerShell.

Install the Azure Synapse PowerShell module

> **IMPORTANT**
> While the **Az.Synapse** PowerShell module is in preview, you must install it separately using the `Install-Module` cmdlet. After this PowerShell module becomes generally available, it will be part of future Az PowerShell module releases and available by default from within Azure Cloud Shell.

```
Install-Module -Name Az.Synapse
```

Create an Azure Synapse workspace using Azure PowerShell

1. Define necessary environment variables to create resources for Azure Synapse workspace.

VARIABLE NAME	DESCRIPTION
StorageAccountName	Name for your existing ADLS Gen2 storage account.

VARIABLE NAME	DESCRIPTION
StorageAccountResourceGroup	Name of your existing ADLS Gen2 storage account resource group.
FileShareName	Name of your existing storage file system.
SynapseResourceGroup	Choose a new name for your Azure Synapse resource group.
Region	Choose one of the Azure regions.
SynapseWorkspaceName	Choose a unique name for your new Azure Synapse Workspace.
SqlUser	Choose a value for a new username.
SqlPassword	Choose a secure password.
ClientIP	Public IP Address of the system you're running PowerShell from.

2. Create a resource group as a container for your Azure Synapse workspace:

```
New-AzResourceGroup -Name $SynapseResourceGroup -Location $Region
```

3. Create an Azure Synapse Workspace:

```
$Cred = New-Object -TypeName System.Management.Automation.PSCredential ($SqlUser, (ConvertTo-
SecureString $SqlPassword -AsPlainText -Force))

$WorkspaceParams = @{
  Name = $SynapseWorkspaceName
  ResourceGroupName = $SynapseResourceGroup
  DefaultDataLakeStorageAccountName = $StorageAccountName
  DefaultDataLakeStorageFilesystem = $FileShareName
  SqlAdministratorLoginCredential = $Cred
  Location = $Region
}
New-AzSynapseWorkspace @WorkspaceParams
```

4. Get Web and Dev URL for Azure Synapse Workspace:

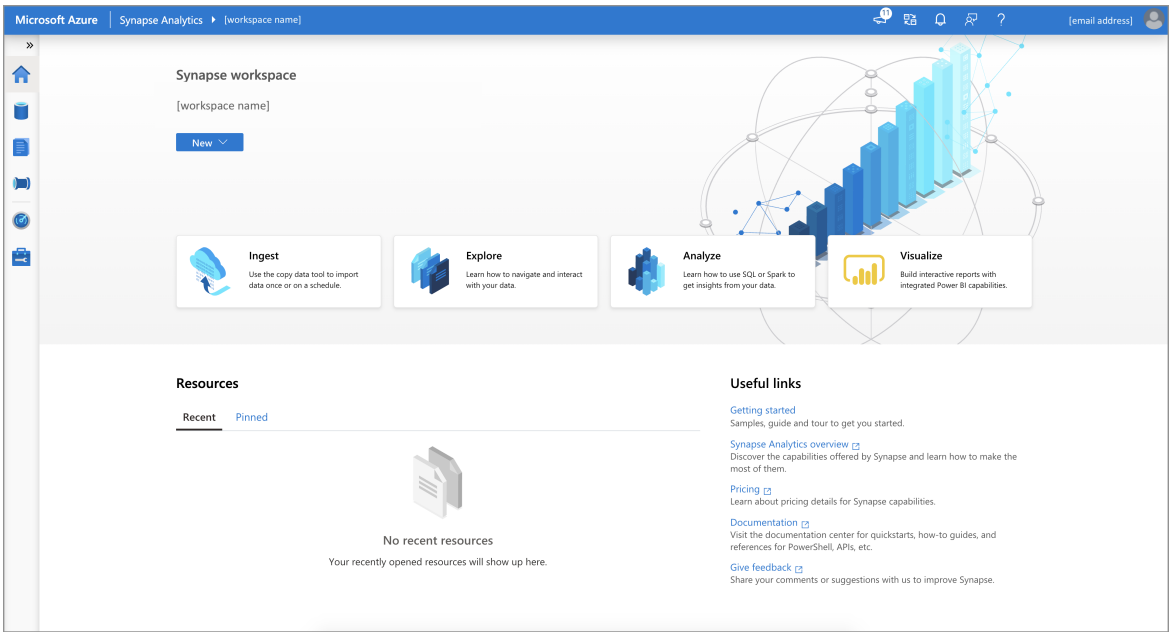
```
$WorkspaceWeb = (Get-AzSynapseWorkspace -Name $SynapseWorkspaceName -ResourceGroupName
$StorageAccountResourceGroup).ConnectivityEndpoints.web
$WorkspaceDev = (Get-AzSynapseWorkspace -Name $SynapseWorkspaceName -ResourceGroupName
$StorageAccountResourceGroup).ConnectivityEndpoints.dev
```

5. Create a Firewall Rule to allow your access to Azure Synapse Workspace from your machine:

```
$FirewallParams = @{
  WorkspaceName = $SynapseWorkspaceName
  Name = 'Allow Client IP'
  ResourceGroupName = $StorageAccountResourceGroup
  StartIpAddress = $ClientIP
  EndIpAddress = $ClientIP
}
New-AzSynapseFirewallRule @FirewallParams
```

6. Open the Azure Synapse Workspace Web URL address stored in environment variable `WorkspaceWeb` to access your workspace:

```
Start-Process $WorkspaceWeb
```



7. Once deployed, additional permissions are required.

- In the Azure portal, assign other users of the workspace to the **Contributor** role in the workspace. For detailed steps, see Assign Azure roles using the Azure portal.
- Assign other users the appropriate **Synapse RBAC roles** using Synapse Studio.
- A member of the **Owner** role of the Azure Storage account must assign the **Storage Blob Data Contributor** role to the Azure Synapse workspace MSI and other users.

Clean up resources

Follow the steps below to delete the Azure Synapse workspace.

> **WARNING**
> Deleting an Azure Synapse workspace will remove the analytics engines and the data stored in the database of the contained SQL pools and workspace metadata. It will no longer be possible to connect to the SQL or Apache Spark endpoints. All code artifacts will be deleted (queries, notebooks, job definitions and pipelines). Deleting the workspace will **not** affect the data in the Data Lake Store Gen2 linked to the workspace.

If the Azure Synapse workspace created in this article isn't needed, you can delete it by running the following example.

```
Remove-AzSynapseWorkspace -Name $SynapseWorkspaceNam -ResourceGroupName $SynapseResourceGroup
```

Next steps

Next, you can create SQL pools or create Apache Spark pools to start analyzing and exploring your data.

Quickstart: Create an Azure Synapse workspace using an ARM template

8/19/2022 • 2 minutes to read • Edit Online

This Azure Resource Manager (ARM) template will create an Azure Synapse workspace with underlying Data Lake Storage. The Azure Synapse workspace is a securable collaboration boundary for analytics processes in Azure Synapse Analytics.

An ARM template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

To create an Azure Synapse workspace, a user must have **Azure Contributor** role and **User Access Administrator** permissions, or the **Owner** role in the subscription. For detailed steps, see Assign Azure roles using the Azure portal.

Review the template

You can review the template by selecting the **Visualize** link. Then select **Edit template**.

The template defines two resources:

- Storage account
- Workspace

Deploy the template

1. Select the following image to sign in to Azure and open the template. This template creates a Synapse workspace.

2. Enter or update the following values:

   - **Subscription**: Select an Azure subscription.
   - **Resource group**: Select **Create new** and enter a unique name for the resource group and select **OK**. A new resource group will facilitate resource clean up.
   - **Region**: Select a region. For example, **Central US**.
   - **Name**: Enter a name for your workspace.

- **SQL Administrator login**: Enter the administrator username for the SQL Server.
- **SQL Administrator password**: Enter the administrator password for the SQL Server.
- **Tag Values**: Accept the default.
- **Review and Create**: Select.
- **Create**: Select.

3. Once deployed, additional permissions are required.

- In the Azure portal, assign other users of the workspace to the **Contributor** role in the workspace. For detailed steps, see Assign Azure roles using the Azure portal.
- Assign other users the appropriate **Synapse RBAC roles** using Synapse Studio.
- A member of the **Owner** role of the Azure Storage account must assign the **Storage Blob Data Contributor** role to the Azure Synapse workspace MSI and other users.

Next steps

To learn more about Azure Synapse Analytics and Azure Resource Manager,

- Read an Overview of Azure Synapse Analytics
- Learn more about Azure Resource Manager
- Create and deploy your first ARM template

Next, you can create SQL pools or create Apache Spark pools to start analyzing and exploring your data.

Quickstart: Linking a Power BI workspace to a Synapse workspace

8/19/2022 • 2 minutes to read • Edit Online

In this quickstart, you will learn how to connect a Power BI workspace to an Azure Synapse Analytics workspace to create new Power BI reports and datasets from Synapse Studio.

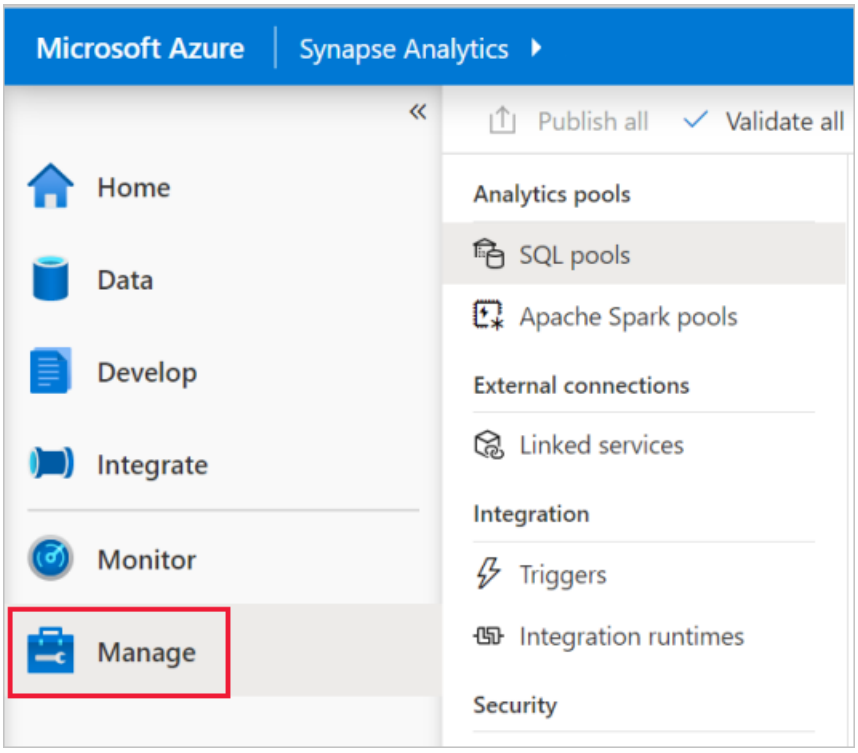
If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

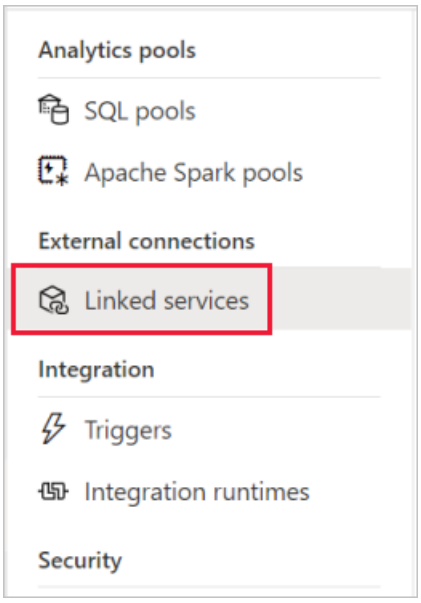
- Create an Azure Synapse workspace and associated storage account
- A Power BI Professional or Premium workspace

Link Power BI workspace to your Synapse workspace

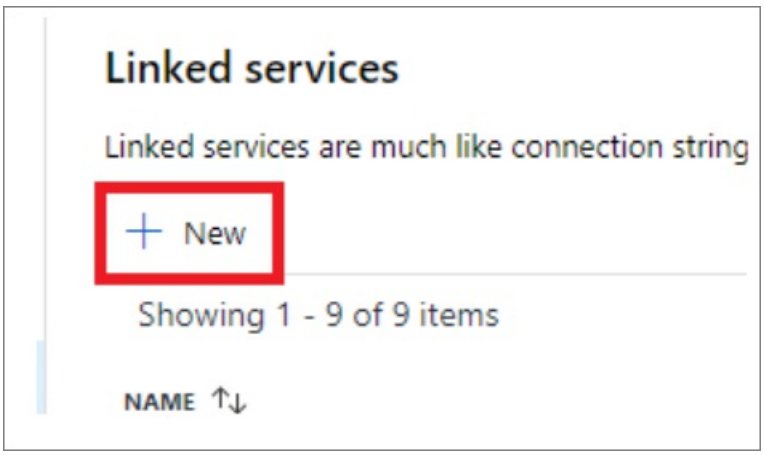
1. Starting from Synapse Studio, click **Manage**.



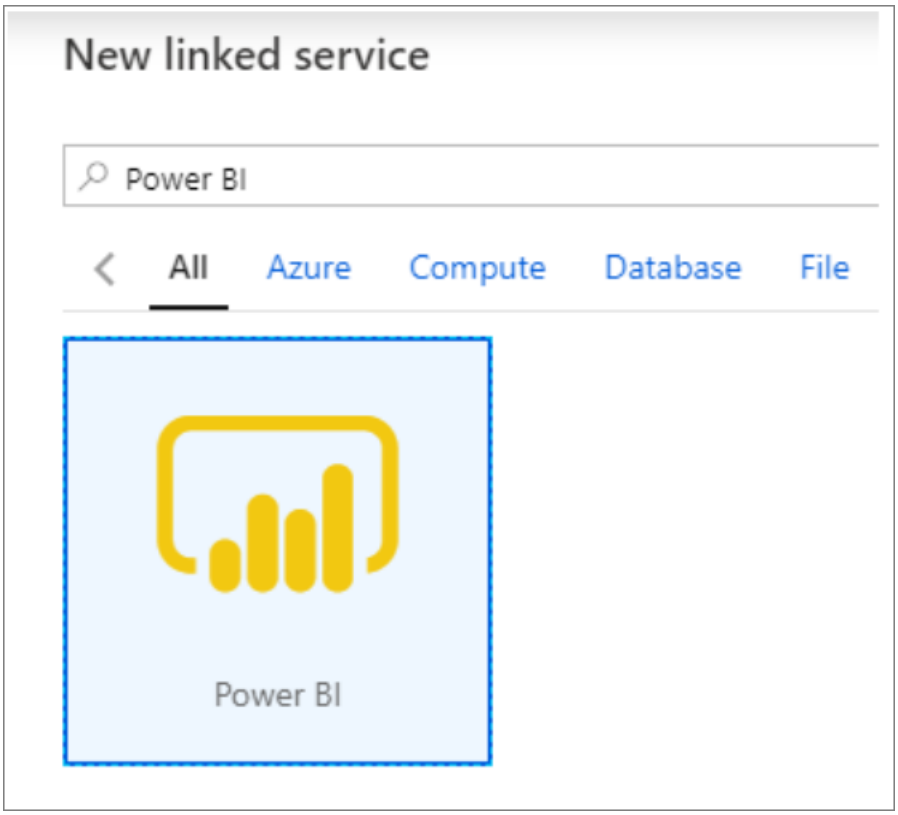
2. Under **External Connections**, click **Linked services**.



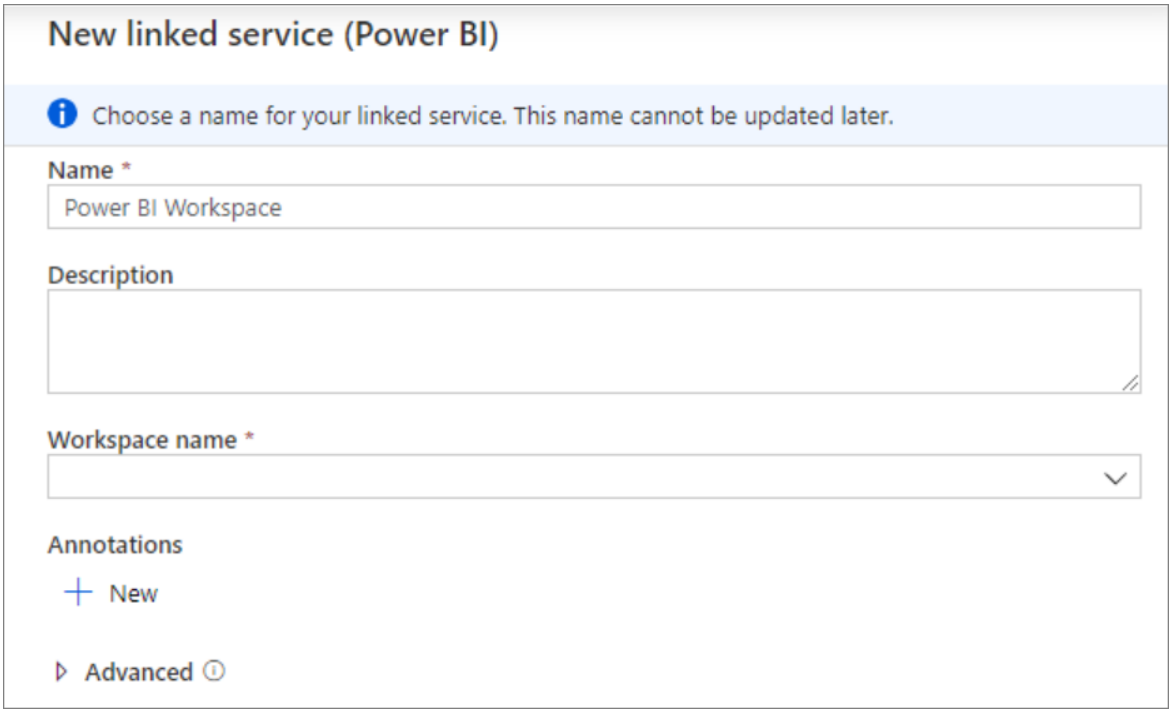
3. Click + **New**.



4. Click **Power BI** and click **Continue**.



5. Enter a name for the linked service and select a workspace from the dropdown list.

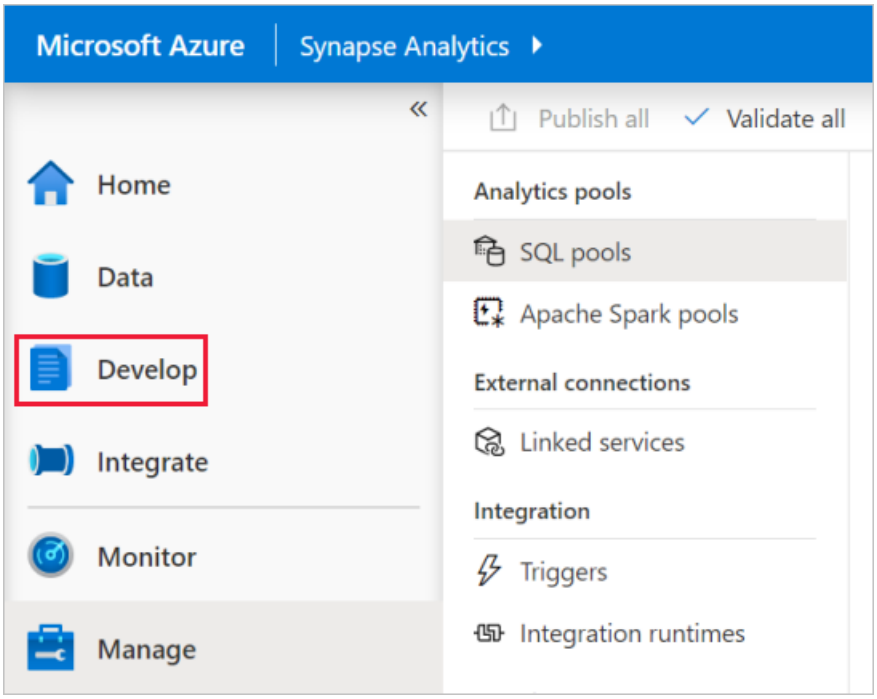


6. Click **Create**.

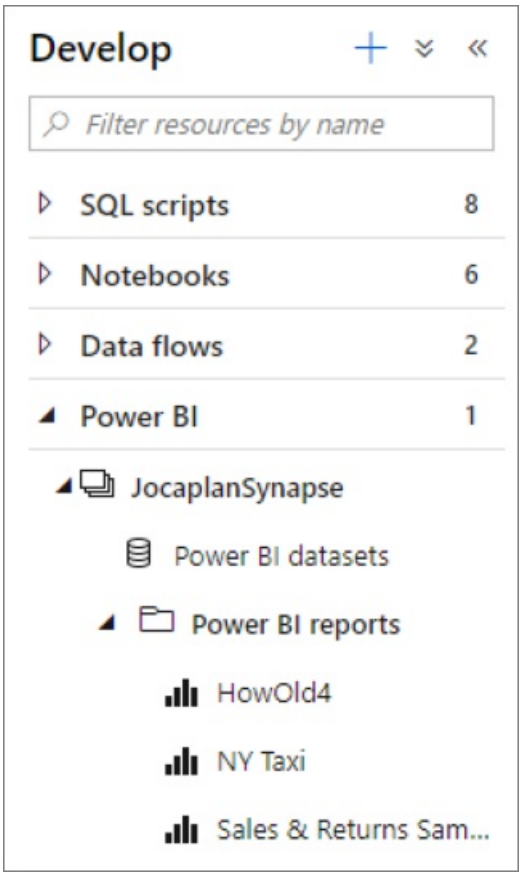
View Power BI workspace in Synapse Studio

Once your workspaces are linked, you can browse your Power BI datasets, edit/create new Power BI Reports from Synapse Studio.

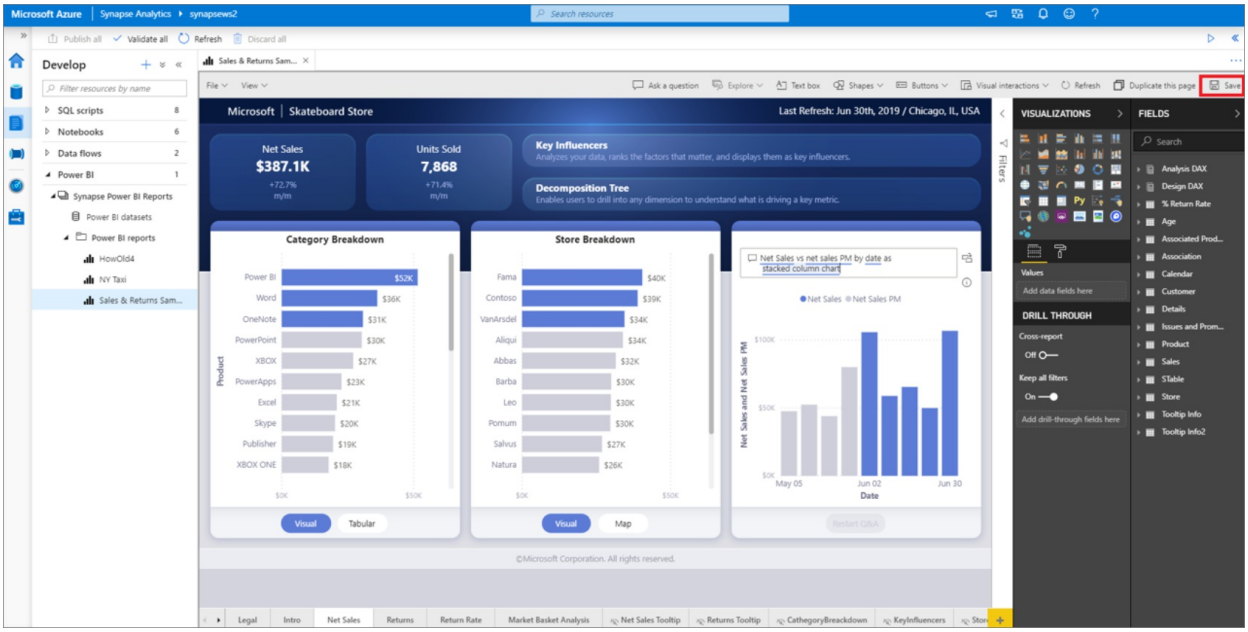
1. Click **Develop**.



2. Expand Power BI and the workspace you wish to use.



New reports can be created clicking **+** at the top of the **Develop** tab. Existing reports can be edited by clicking on the report name. Any saved changes will be written back to the Power BI workspace.



Next steps

Learn more about creating Power BI report on files stored on Azure Storage.

Quickstart: Read data from ADLS Gen2 to Pandas dataframe in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

In this quickstart, you'll learn how to easily use Python to read data from an Azure Data Lake Storage (ADLS) Gen2 into a Pandas dataframe in Azure Synapse Analytics.

From a Synapse Studio notebook, you'll:

- Connect to a container in Azure Data Lake Storage (ADLS) Gen2 that is linked to your Azure Synapse Analytics workspace.
- Read the data from a PySpark Notebook using `spark.read.load`.
- Convert the data to a Pandas dataframe using `.toPandas()`.

Prerequisites

- Azure subscription - Create one for free.
- Synapse Analytics workspace with ADLS Gen2 configured as the default storage - You need to be the **Storage Blob Data Contributor** of the ADLS Gen2 filesystem that you work with. For details on how to create a workspace, see Creating a Synapse workspace.
- Apache Spark pool in your workspace - See Create a serverless Apache Spark pool.

Sign in to the Azure portal

Sign in to the Azure portal.

Upload sample data to ADLS Gen2

1. In the Azure portal, create a container in the same ADLS Gen2 used by Synapse Studio. You can skip this step if you want to use the default linked storage account in your Azure Synapse Analytics workspace.
2. In Synapse Studio, select **Data**, select the **Linked** tab, and select the container under **Azure Data Lake Storage Gen2**.
3. Download the sample file RetailSales.csv and upload it to the container.
4. Select the uploaded file, select **Properties**, and copy the **ABFSS Path** value.

Read data from ADLS Gen2 into a Pandas dataframe

1. In the left pane, select **Develop**.
2. Select **+** and select "Notebook" to create a new notebook.
3. In **Attach to**, select your Apache Spark Pool. If you don't have one, select **Create Apache Spark pool**.
4. In the notebook code cell, paste the following Python code, inserting the ABFSS path you copied earlier:

```
%%pyspark
data_path = spark.read.load('<ABFSS Path to RetailSales.csv>', format='csv', header=True)
data_path.show(10)

print('Converting to Pandas.')

pdf = data_path.toPandas()
print(pdf)
```

5. Run the cell.

After a few minutes, the text displayed should look similar to the following.

```
Command executed in 25s 324ms by gary on 03-23-2021 17:40:23.481 -07:00

Job execution Succeeded Spark 2 executors 8 cores

+-------+-----------+--------+-----------+-----------+-----+------------+--------------------+
|storeId|productCode|quantity|logQuantity|advertising|price|weekStarting|                  id|
+-------+-----------+--------+-----------+-----------+-----+------------+--------------------+
|      2| surface.go|     105|9.264828557|          1|  159|   6/15/2017|d6bd47a7-2ad6-4f0...|
|      2| surface.go|      80|8.987196821|          0|  269|   7/27/2017|64cc74c2-c7da-4e1...|
|      2| surface.go|      68|8.831711918|          1|  209|    8/3/2017|9a2d164b-5e44-44d...|
|      2| surface.go|      28|7.965545573|          0|  209|   8/10/2017|b8cd9987-1d5a-4f4...|
|      2| surface.go|      16|7.377758908|          0|  209|   8/24/2017|ac0ec099-e102-4bf...|
|      2| surface.go|     253| 10.1402973|          1|  189|   8/31/2017|3d22c002-b04c-409...|
|      2| surface.go|     107|9.282847063|          0|  189|    9/7/2017|b6e19699-d684-449...|
|      2| surface.go|      66|8.803273983|          0|  189|   9/14/2017|e89a5838-fb8f-413...|
|      2| surface.go|      65|8.793612072|          0|  179|   9/21/2017|c3278682-16c0-483...|
|      2| surface.go|      17|7.454719949|          0|  269|  10/12/2017|f40190c1-b2ed-46f...|
+-------+-----------+--------+-----------+-----------+-----+------------+--------------------+
only showing top 10 rows

Converting to Pandas.
      storeId  ...                                    id
0           2  ...  d6bd47a7-2ad6-4f0a-b8de-ed1386cae5ea
1           2  ...  64cc74c2-c7da-4e12-af64-c95bdf429934
2           2  ...  9a2d164b-5e44-44d7-9837-cf9ae6566c99
3           2  ...  b8cd9987-1d5a-4f4f-9346-719d73b1f7f0
4           2  ...  ac0ec099-e102-4bfc-9775-983b151dcd03
...       ...  ...                                   ...
28942     137  ...  6af00133-7015-415d-831b-ddf05bb5828c
28943     137  ...  1e0d3a21-ab43-49c4-89e2-49d202821807
28944     137  ...  5cc7e50a-6aa4-419b-a933-905a667aa2df
28945     137  ...  650ca506-7a4f-46f8-b2e1-e52ceffadf16
28946     137  ...  9bb216f6-04ec-4b61-9e68-34772b814c44

[28947 rows x 8 columns]
```

Next steps

- What is Azure Synapse Analytics?
- Get Started with Azure Synapse Analytics
- Create a serverless Apache Spark pool
- How to use file mount/unmount API in Synapse
- Azure Architecture Center: Explore data in Azure Blob storage with the pandas Python package
- Tutorial: Use Pandas to read/write Azure Data Lake Storage Gen2 data in serverless Apache Spark pool in Synapse Analytics

Connect to Azure Data Explorer using Apache Spark for Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

This article describes how to access an Azure Data Explorer database from Synapse Studio with Apache Spark for Azure Synapse Analytics.

Prerequisites

- Create an Azure Data Explorer cluster and database.
- Have an existing Azure Synapse Analytics workspace, or create a new workspace by following the steps in Quickstart: Create an Azure Synapse workspace.
- Have an existing Apache Spark pool, or create a new pool by following the steps in Quickstart: Create an Apache Spark pool using the Azure portal.
- Create an Azure Active Directory (Azure AD) app by provisioning an Azure AD application.
- Grant your Azure AD app access to your database by following the steps in Manage Azure Data Explorer database permissions.

Go to Synapse Studio

From an Azure Synapse workspace, select **Launch Synapse Studio**. On the Synapse Studio home page, select **Data** to go to **Data Object Explorer**.

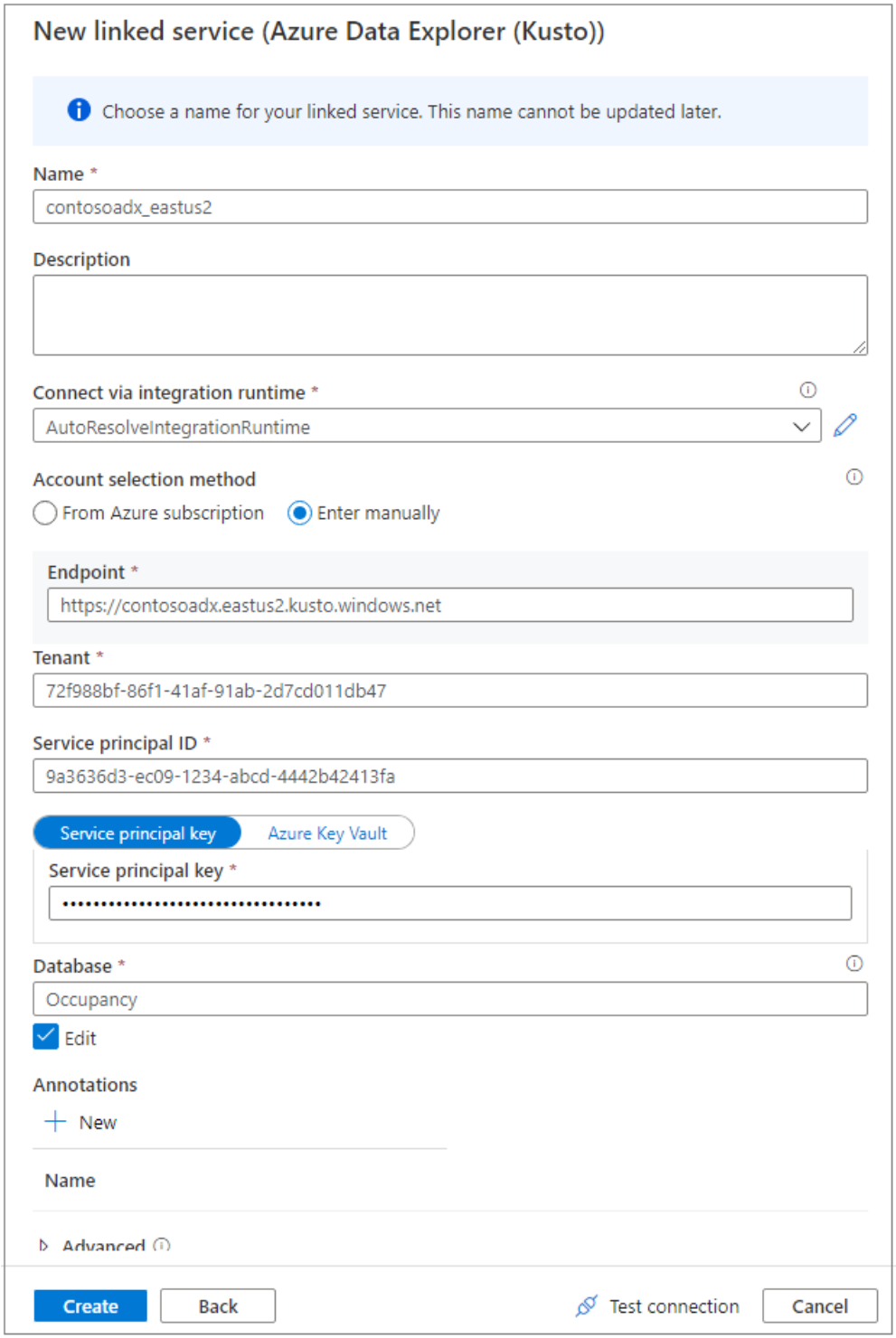
Connect an Azure Data Explorer database to an Azure Synapse workspace

Connecting an Azure Data Explorer database to a workspace is done through a linked service. With an Azure Data Explorer linked service, you can browse and explore data, read, and write from Apache Spark for Azure Synapse. You can also run integration jobs in a pipeline.

From the Data Object Explorer, follow these steps to directly connect an Azure Data Explorer cluster:

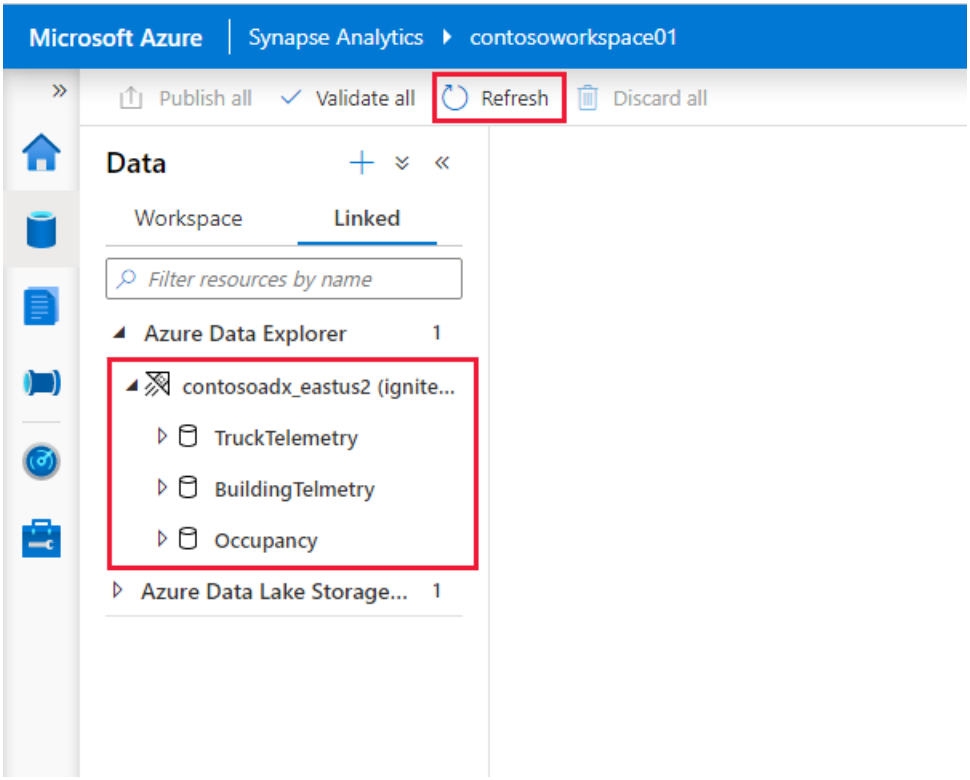
1. Select the **+** icon near **Data**.
2. Select **Connect** to connect to external data.
3. Select **Azure Data Explorer (Kusto)**.
4. Select **Continue**.
5. Use a friendly name to name the linked service. The name will appear in the Data Object Explorer and is used by Azure Synapse runtimes to connect to the database.
6. Select the Azure Data Explorer cluster from your subscription, or enter the URI.
7. Enter the **Service principal ID** and **Service principal key**. Ensure this service principal has view access on the database for read operation and ingestor access for ingesting data.
8. Enter the Azure Data Explorer database name.
9. Select **Test connection** to ensure you have the right permissions.

10. Select **Create**.



> **NOTE**
>
> (Optional) **Test connection** doesn't validate write access. Ensure your service principal ID has write access to the Azure Data Explorer database.

11. Azure Data Explorer clusters and databases appear on the **Linked** tab under the **Azure Data Explorer** section.

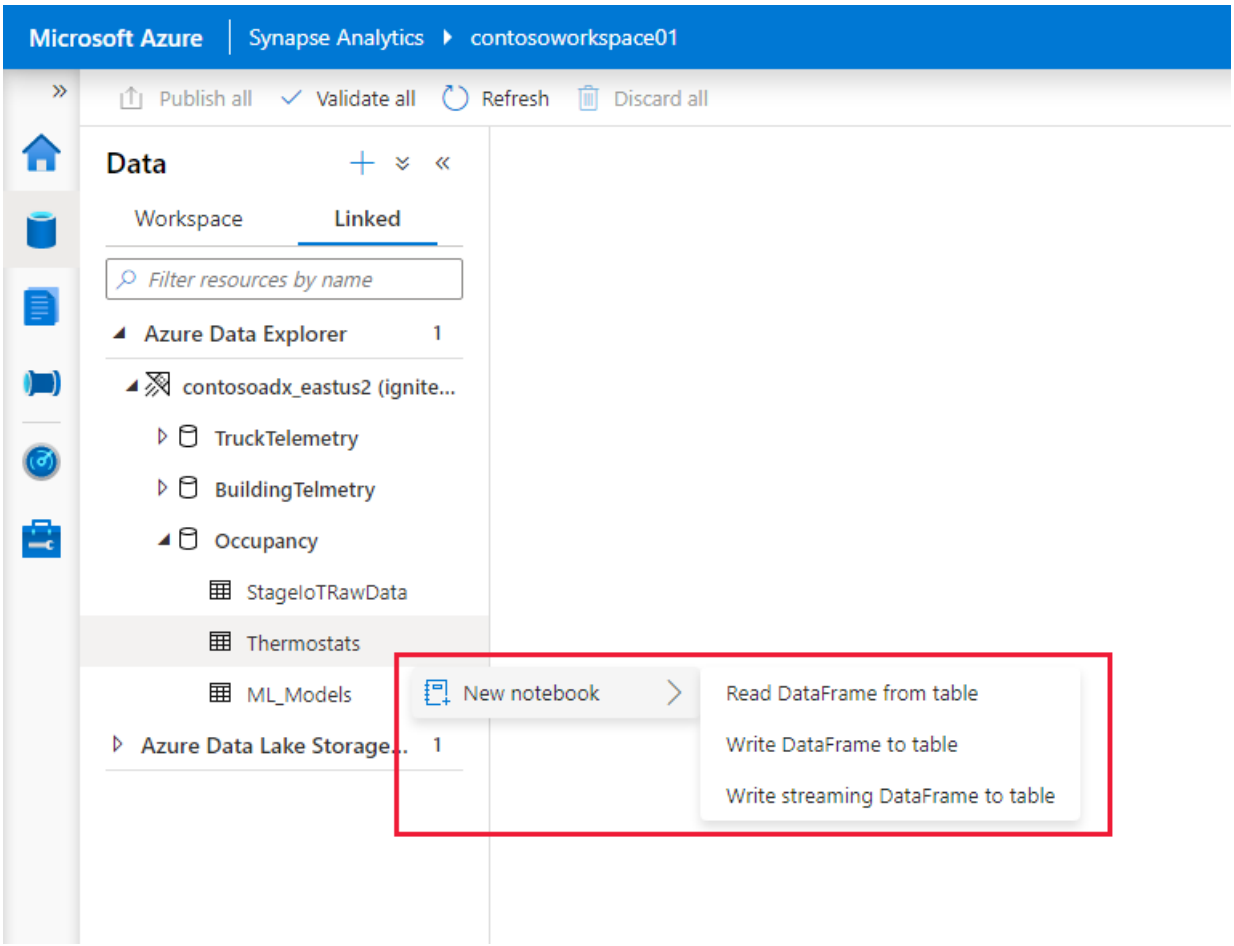


12. Before you can interact with the linked service from a notebook, it must be published to the Workspace. Click **Publish** in the toolbar, review the pending changes and click **OK**.

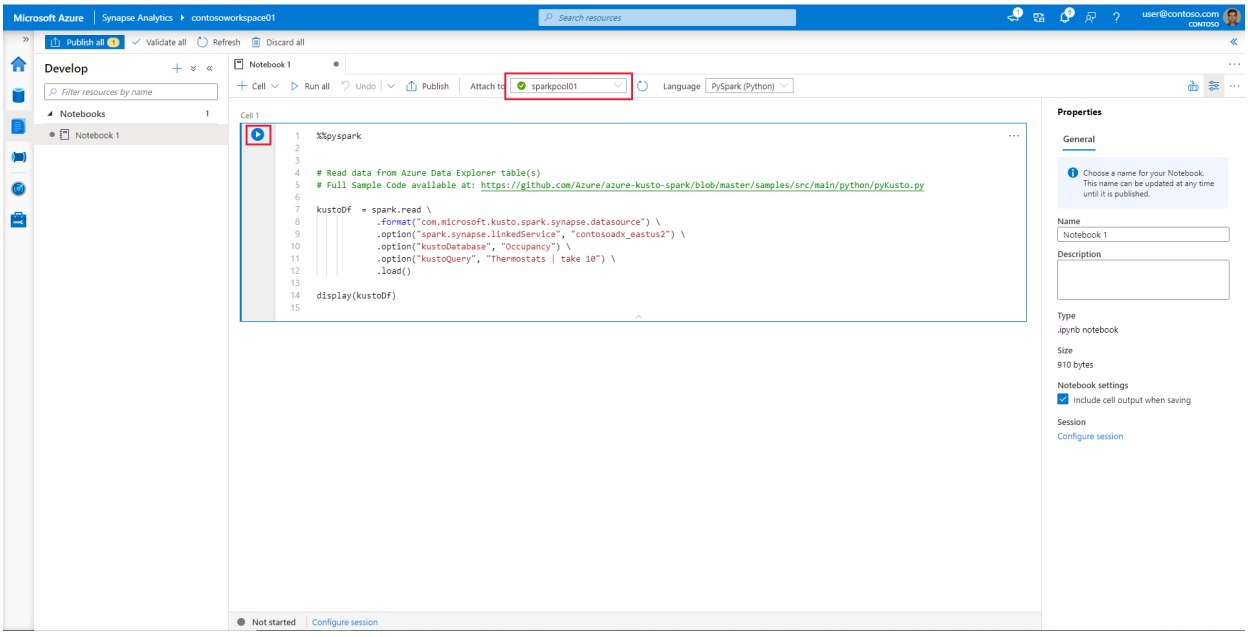
> **NOTE**
>
> In the current release, the database objects are populated based on your Azure AD account permissions on the Azure Data Explorer databases. When you run the Apache Spark notebooks or integration jobs, the credential in the link service will be used (for example, service principal).

Quickly interact with code-generated actions

When you right-click a database or table, a list of sample Spark notebooks appears. Select an option to read, write, or stream data to Azure Data Explorer.



Here's an example of reading data. Attach the notebook to your Spark pool, and run the cell.



> **NOTE**
>
> First-time execution might take more than three minutes to initiate the Spark session. Subsequent executions will be significantly faster.

Limitations

The Azure Data Explorer connector is currently not supported with Azure Synapse managed virtual networks.

Next steps

- Sample code with advanced options
- Azure Data Explorer (Kusto) Spark connector

Discover partner solutions through Synapse Studio

8/19/2022 • 2 minutes to read • Edit Online

Synapse Studio allows the discovery of solution partners that extend the capabilities of Azure Synapse. From data connectors, through data wrangling tools, orchestration engines, and other workloads, the **browse partners** page serves as a hub for discovering third-party ISV applications and solutions verified to work with Azure Synapse Analytics. Synapse Studio simplifies getting started with these partners, in many cases with automated setup of the initial connection to the partner platform.

Participating partners

The following table lists partner solutions that are currently supported. Make sure you check back often as we add new partners to this list.

PARTNER	SOLUTION NAME
	Incorta Intelligent Ingest for Azure Synapse
	InformaticaIntelligent DataManagement Cloud
	Qlik Data Integration (formerly Attunity)

Requirements

When you chose a partner application, Azure Synapse Studio provisions a sandbox environment you can use for this trial, ensuring you can experiment with partner solutions quickly before you decide to use it with your production data. The following objects are created:

OBJECT	DETAILS
A dedicated SQL pool named **Partner_[PartnerName]_pool**	DW100c performance level.
A SQL login named **Partner_[PartnerName]_login**	Created on your `master` database. The password for this SQL login is specified by you at the creation of your trial.
A database user	A new database user, mapped to the new SQL login. This user is added to the db_owner role for the newly created database.

In all cases, **[PartnerName]** is the name of the third-party ISV who offers the trial.

Security

After the required objects are created, Synapse Studio sends information about your new sandbox environment to the partner application, allowing a customized trial experience. The following information is sent to our

partners:

- First name
- Last name
- E-mail address
- Details about the Synapse environment required to establish a connection:
  - DNS name of your Synapse Workspace (server name)
  - Name of the SQL pool (database)
  - SQL login (username only)

We never share any passwords with the partner application, including the password of the newly created SQL login. You'll be required to type your password in the partner application.

Costs

The dedicated SQL pool that is created for your partner trial incurs ongoing costs, which are based on the number of DWU blocks and hours running. Make sure you pause the SQL pool created for this partner trial when it isn't in use, to avoid unnecessary charges.

Starting a new partner trial

1. On the Synapse Studio home page, under **Discover more**, select **browse partners**.
2. The Browse partners page shows all partners currently offering trials that allow direct connectivity with Azure Synapse. Choose a partner solution.
3. The partner details page shows you relevant information about this application and links to learn more about their solution. When you're ready to start a trial, select **Connect to partner**.
4. In the **Connect to [PartnerName] Solution** page, note requirements of this partner connection. Change the SQL pool name and SQL login parameters if wanted (or accept the defaults), type the password of your new SQL login, and select **Connect**.

The required objects will be created for your partner trial. You'll then be forwarded to a partner page to provide additional information (if needed) and to start your trial.

> **NOTE**
> Microsoft doesn't control the partner trial experience. Partners offer product trials on their own terms and the experience, trial availability, and features may vary depending on the partner. Microsoft does not offer support to third-party applications offered in Synapse Studio.

Next steps

To learn more about some of our other partners, see Data integration partners, Data management partners, and Machine Learning and AI partners.

Azure Synapse Analytics shared metadata

8/19/2022 • 2 minutes to read • Edit Online

Azure Synapse Analytics allows the different workspace computational engines to share databases and tables between Apache Spark pools and serverless SQL pool.

The sharing supports the so-called modern data warehouse pattern and gives the workspace SQL engines access to databases and tables created with Spark. It also allows the SQL engines to create their own objects that aren't being shared with the other engines.

Support the modern data warehouse

The shared metadata model supports the modern data warehouse pattern in the following way:

1. Data from the data lake is prepared and structured efficiently with Spark by storing the prepared data in (possibly partitioned) Parquet-backed tables contained in possibly several databases.
2. The Spark created databases and all their tables become visible in any of the Azure Synapse workspace Spark pool instances and can be used from any of the Spark jobs. This capability is subject to the permissions since all Spark pools in a workspace share the same underlying catalog meta store.
3. The Spark created databases and their Parquet-backed or CSV-backed tables become visible in the workspace serverless SQL pool. Databases are created automatically in the serverless SQL pool metadata, and both the external and managed tables created by a Spark job are made accessible as external tables in the serverless SQL pool metadata in the `dbo` schema of the corresponding database.

Object synchronization occurs asynchronously. Objects will have a slight delay of a few seconds until they appear in the SQL context. Once they appear, they can be queried, but not updated nor changed by the SQL engines that have access to them.

Shared metadata objects

Spark allows you to create databases, external tables, managed tables, and views. Since Spark views require a Spark engine to process the defining Spark SQL statement, and cannot be processed by a SQL engine, only databases and their contained external and managed tables that use the Parquet or CSV storage format are shared with the workspace SQL engine. Spark views are only shared among the Spark pool instances.

Security model at a glance

The Spark databases and tables, along with their synchronized representations in the SQL engine, are secured at the underlying storage level. When the table is queried by any of the engines that the query submitter has the right to use, the query submitter's security principal is being passed through to the underlying files. Permissions are checked at the file system level.

For more information, see Azure Synapse Analytics shared database.

Change maintenance

If a metadata object is deleted or changed with Spark, the changes are picked up and propagated to the serverless SQL pool. Synchronization is asynchronous and changes are reflected in the SQL engine after a short delay.

Next steps

- Learn more about Azure Synapse Analytics' shared metadata Databases
- Learn more about Azure Synapse Analytics' shared metadata Tables

Access lake databases using serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

The Azure Synapse Analytics workspace enables you to create two types of databases on top of a Spark data lake:

- **Lake databases** where you can define tables on top of lake data using Apache Spark notebooks, database templates, or Microsoft Dataverse (previously Common Data Service). These tables will be available for querying using T-SQL (Transact-SQL) language using the serverless SQL pool.
- **SQL databases** where you can define your own databases and tables directly using the serverless SQL pools. You can use T-SQL CREATE DATABASE, CREATE EXTERNAL TABLE to define the objects and add additional SQL views, procedures, and inline-table-value functions on top of the tables.

This article focuses on lake databases in a serverless SQL pool in Azure Synapse Analytics.

Azure Synapse Analytics allows you to create lake databases and tables using Spark or database designer, and then analyze data in the lake databases using the serverless SQL pool. The lake databases and the tables (parquet or CSV-backed) that are created on the Apache Spark pools, database templates, or Dataverse are automatically available for querying with the serverless SQL pool engine. The lake databases and tables that are modified will be available in serverless SQL pool after some time. There will be a delay until the changes made in Spark or Database designed appear in serverless.

Manage lake database

To manage Spark created lake databases, you can use Apache Spark pools or Database designer. For example, create or delete a lake database through a Spark pool job. You can't create a lake database or the objects in the lake databases using the serverless SQL pool.

The Spark default database is available in the serverless SQL pool context as a lake database called `default`.

> **NOTE**
> You cannot create a lake and a SQL database in the serverless SQL pool with the same name.

Tables in the lake databases cannot be modified from a serverless SQL pool. Use the Database designer or Apache Spark pools to modify a lake database. The serverless SQL pool enables you to make the following changes in a lake database using Transact-SQL commands:

- Adding, altering, and dropping views, procedures, inline table-value functions in a lake database.
- Adding and removing database-scoped Azure AD users.
- Add or remove Azure AD database users to the **db_datareader** role. Azure AD database users in the **db_datareader** role have permission to read all tables in the lake database, but cannot read data from other databases.

Security model

The lake databases and tables are secured at two levels:

- The underlying storage layer by assigning to Azure AD users one of the following:

    - Azure role-based access control (Azure RBAC)
    - Azure attribute-based access control (Azure ABAC) role
    - ACL permissions
- The SQL layer where you can define an Azure AD user and grant SQL permissions to SELECT data from tables referencing the lake data.

Lake security model

Access to lake database files is controlled using the lake permissions on storage layer. Only Azure AD users can use tables in the lake databases, and they can access the data in the lake using their own identities.

You can grant access to the underlying data used for external tables to a security principal, such as: a user, an Azure AD application with assigned service principal, or a security group. For data access, grant both of the following permissions:

- Grant `read (R)` permission on files (such as the table's underlying data files).
- Grant `execute (X)` permission on the folder where the files are stored and on every parent folder up to the root. You can read more about these permissions on Access control lists(ACLs) page.

For example, in

```
https://<storage-
name>.dfs.core.windows.net/<fs>/synapse/workspaces/<synapse_ws>/warehouse/mytestdb.db/myparquettable/
```

, security principals need:

- `execute (X)` permissions on all the folders starting at the `<fs>` to the `myparquettable`.
- `read (R)` permissions on `myparquettable` and files inside that folder, to be able to read a table in a database (synchronized or original one).

If a security principal requires the ability to create objects or drop objects in a database, additional `write (W)` permissions are required on the folders and files in the `warehouse` folder. Modifying objects in a database is not possible from serverless SQL pool, only from Spark pools or the database designer.

SQL security model

The Azure Synapse workspace provides a T-SQL endpoint that enables you to query the lake database using the serverless SQL pool. In addition to the data access, SQL interface enables you to control who can access the tables. You need to enable a user to access the shared lake databases using the serverless SQL pool. There are two types of users who can access the lake databases:

- Administrators: Assign the **Synapse SQL Administrator** workspace role or **sysadmin** server-level role inside the serverless SQL pool. This role has full control over all databases. The **Synapse Administrator** and **Synapse SQL Administrator** roles also have all permissions on all objects in a serverless SQL pool, by default.
- Workspace readers: Grant the server-level permissions **GRANT CONNECT ANY DATABASE** and **GRANT SELECT ALL USER SECURABLES** on serverless SQL pool to a login that will enable the login to access and read any database. This might be a good choice for assigning reader/non-admin access to a user.
- Database readers: Create database users from Azure AD in your lake database and add them to **db_datareader** role, which will enable them to read data in the lake database.

Learn more about setting access control on shared databases here.

Custom SQL objects in lake databases

Lake databases allow creation of custom T-SQL objects, such as schemas, procedures, views, and the inline table-value functions (iTVFs). In order to create custom SQL objects, you **MUST** create a schema where you will place the objects. Custom SQL objects cannot be placed in `dbo` schema because it is reserved for the lake tables

that are defined in Spark, database designer, or Dataverse.

> **IMPORTANT**
> You must create custom SQL schema where you will place your SQL objects. The custom SQL objects cannot be placed in the `dbo` schema. The `dbo` schema is reserved for the lake tables that are originally created in Spark or database designer.

Examples

Create SQL database reader in lake database

In this example, we are adding an Azure AD user in the lake database who can read data via shared tables. The users are added in the lake database via the serverless SQL pool. Then, assign the user to the **db_datareader** role so they can read data.

```
CREATE USER [customuser@contoso.com] FROM EXTERNAL PROVIDER;
GO
ALTER ROLE db_datareader
ADD MEMBER [customuser@contoso.com];
```

Create workspace-level data reader

A login with `GRANT CONNECT ANY DATABASE` and `GRANT SELECT ALL USER SECURABLES` permissions is able to read all tables using the serverless SQL pool, but not able to create SQL databases or modify the objects in them.

```
CREATE LOGIN [wsdatareader@contoso.com] FROM EXTERNAL PROVIDER
GRANT CONNECT ANY DATABASE TO [wsdatareader@contoso.com]
GRANT SELECT ALL USER SECURABLES TO [wsdatareader@contoso.com]
```

This script enables you to create users without admin privileges who can read any table in Lake databases.

Create and connect to Spark database with serverless SQL pool

First, create a new Spark database named `mytestdb` using a Spark cluster you have already created in your workspace. You can achieve that, for example, using a Spark C# Notebook with the following .NET for Spark statement:

```
spark.Sql("CREATE DATABASE mytestlakedb")
```

After a short delay, you can see the lake database from serverless SQL pool. For example, run the following statement from serverless SQL pool.

```
SELECT * FROM sys.databases;
```

Verify that `mytestlakedb` is included in the results.

Create custom SQL objects in lake database

The following example shows how to create a custom view, procedure, and inline table-value function (iTVF) in the `reports` schema:

```
CREATE SCHEMA reports
GO

CREATE OR ALTER VIEW reports.GreenReport
AS SELECT puYear, puMonth,
            fareAmount = SUM(fareAmount),
            tipAmount = SUM(tipAmount),
            mtaTax = SUM(mtaTax)
FROM dbo.green
GROUP BY puYear, puMonth
GO

CREATE OR ALTER PROCEDURE reports.GreenReportSummary
AS BEGIN
SELECT puYear, puMonth,
            fareAmount = SUM(fareAmount),
            tipAmount = SUM(tipAmount),
            mtaTax = SUM(mtaTax)
FROM dbo.green
GROUP BY puYear, puMonth
END
GO

CREATE OR ALTER FUNCTION reports.GreenDataReportMonthly(@year int)
RETURNS TABLE
RETURN ( SELECT puYear = @year, puMonth,
                fareAmount = SUM(fareAmount),
                tipAmount = SUM(tipAmount),
                mtaTax = SUM(mtaTax)
        FROM dbo.green
        WHERE puYear = @year
        GROUP BY puMonth )
GO
```

Next steps

- Learn more about Azure Synapse Analytics' shared metadata
- Learn more about Azure Synapse Analytics' shared metadata Tables
- Quickstart: Create a new lake database leveraging database templates
- Tutorial: Use serverless SQL pool with Power BI Desktop & create a report
- Synchronize Apache Spark for Azure Synapse external table definitions in serverless SQL pool
- Tutorial: Explore and Analyze data lakes with serverless SQL pool

Azure Synapse Analytics shared metadata tables

8/19/2022 • 7 minutes to read • Edit Online

Azure Synapse Analytics allows the different workspace computational engines to share databases and tables between its Apache Spark pools and serverless SQL pool.

Once a database has been created by a Spark job, you can create tables in it with Spark that use Parquet, Delta, or CSV as the storage format. Table names will be converted to lower case and need to be queried using the lower case name. These tables will immediately become available for querying by any of the Azure Synapse workspace Spark pools. They can also be used from any of the Spark jobs subject to permissions.

The Spark created, managed, and external tables are also made available as external tables with the same name in the corresponding synchronized database in serverless SQL pool. Exposing a Spark table in SQL provides more detail on the table synchronization.

Since the tables are synchronized to serverless SQL pool asynchronously, there will be a small delay until they appear.

Manage a Spark created table

Use Spark to manage Spark created databases. For example, delete it through a serverless Apache Spark pool job, and create tables in it from Spark.

Objects in synchronized databases cannot be modified from serverless SQL pool.

Expose a Spark table in SQL

Shared Spark tables

Spark provides two types of tables that Azure Synapse exposes in SQL automatically:

- Managed tables

  Spark provides many options for how to store data in managed tables, such as TEXT, CSV, JSON, JDBC, PARQUET, ORC, HIVE, DELTA, and LIBSVM. These files are normally stored in the `warehouse` directory where managed table data is stored.

- External tables

  Spark also provides ways to create external tables over existing data, either by providing the `LOCATION` option or using the Hive format. Such external tables can be over a variety of data formats, including Parquet.

Azure Synapse currently only shares managed and external Spark tables that store their data in Parquet, DELTA, or CSV format with the SQL engines. Tables backed by other formats are not automatically synced. You may be able to sync such tables explicitly yourself as an external table in your own SQL database if the SQL engine supports the table's underlying format.

> **NOTE**
>
> Currently, only Parquet and CSV formats are fully supported in serverless SQL pool. Spark Delta tables are also available in the serverless SQL pool, but this feature is in **public preview**. External tables created in Spark are not available in dedicated SQL pool databases.

Share Spark tables

The shareable managed and external Spark tables exposed in the SQL engine as external tables with the following properties:

- The SQL external table's data source is the data source representing the Spark table's location folder.
- The SQL external table's file format is Parquet, Delta, or CSV.
- The SQL external table's access credential is pass-through.

Since all Spark table names are valid SQL table names and all Spark column names are valid SQL column names, the Spark table and column names will be used for the SQL external table.

Spark tables provide different data types than the Synapse SQL engines. The following table maps Spark table data types map to the SQL types:

SPARK DATA TYPE	SQL DATA TYPE	COMMENTS
`LongType`, `long`, `bigint`	`bigint`	**Spark**: *LongType* represents 8-byte signed integer numbers. **SQL**: See int, bigint, smallint, and tinyint.
`BooleanType`, `boolean`	`bit` (Parquet), `varchar(6)` (CSV)	**Spark**: Boolean. **SQL**: See [/sql/t-sql/data-types/bit-transact-sql).
`DecimalType`, `decimal`, `dec`, `numeric`	`decimal`	**Spark**: *DecimalType* represents arbitrary-precision signed decimal numbers. Backed internally by java.math.BigDecimal. A BigDecimal consists of an arbitrary precision integer unscaled value and a 32-bit integer scale. **SQL**: Fixed precision and scale numbers. When maximum precision is used, valid values are from - 10^38 +1 through 10^38 - 1. The ISO synonyms for decimal are dec and dec(p, s). numeric is functionally identical to decimal. See decimal and numeric.
`IntegerType`, `Integer`, `int`	`int`	**Spark** *IntegerType* represents 4-byte signed integer numbers. **SQL**: See int, bigint, smallint, and tinyint.
`ByteType`, `Byte`, `tinyint`	`smallint`	**Spark**: *ByteType* represents 1-byte signed integer numbers [-128 to 127] and ShortType represents 2-byte signed integer numbers [-32768 to 32767]. **SQL**: Tinyint represents 1-byte signed integer numbers [0, 255] and smallint represents 2-byte signed integer numbers [-32768, 32767]. See int, bigint, smallint, and tinyint.
`ShortType`, `Short`, `smallint`	`smallint`	Same as above.

SPARK DATA TYPE	SQL DATA TYPE	COMMENTS
`DoubleType`, `Double`	`float`	**Spark**: *DoubleType* represents 8-byte double-precision floating point numbers. **SQL**: See float and real.
`FloatType`, `float`, `real`	`real`	**Spark**: *FloatType* represents 4-byte double-precision floating point numbers. **SQL**: See float and real.
`DateType`, `date`	`date`	**Spark**: *DateType* represents values comprising values of fields year, month and day, without a time-zone. **SQL**: See date.
`TimestampType`, `timestamp`	`datetime2`	**Spark**: *TimestampType* represents values comprising values of fields year, month, day, hour, minute, and second, with the session local time-zone. The timestamp value represents an absolute point in time. **SQL**: See datetime2.
`char`	`char`	
`StringType`, `String`, `varchar`	`Varchar(n)`	**Spark**: *StringType* represents character string values. *VarcharType(n)* is a variant of StringType which has a length limitation. Data writing will fail if the input string exceeds the length limitation. This type can only be used in table schema, not functions/operators. *CharType(n)* is a variant of *VarcharType(n)* which is fixed length. Reading column of type *CharType(n)* always returns string values of length n. *CharType(n)* column comparison will pad the short one to the longer length. **SQL**: If there's a length provided from Spark, n in *varchar(n)* will be set to that length. If it is partitioned column, n can be max 2048. Otherwise, it will be *varchar(max)*. See char and varchar. Use it with collation `Latin1_General_100_BIN2_UTF8`.
`BinaryType`, `binary`	`varbinary(n)`	**SQL**: If there's a length provided from Spark, `n` in *Varbinary(n)* will be set to that length. If it is partitioned column, n can be max 2048. Otherwise, it will be *Varbinary(max)*. See binary and varbinary.
`array`, `map`, `struct`	`varchar(max)`	**SQL**: Serializes into JSON with collation `Latin1_General_100_BIN2_UTF8`. See JSON Data.

> **NOTE**
> Database level collation is `Latin1_General_100_CI_AS_SC_UTF8`.

Security model

The Spark databases and tables, as well as their synchronized representations in the SQL engine will be secured at the underlying storage level. Since they do not currently have permissions on the objects themselves, the objects can be seen in the object explorer.

The security principal who creates a managed table is considered the owner of that table and has all the rights to the table as well as the underlying folders and files. In addition, the owner of the database will automatically become co-owner of the table.

If you create a Spark or SQL external table with authentication pass-through, the data is only secured at the folder and file levels. If someone queries this type of external table, the security identity of the query submitter is passed down to the file system, which will check for access rights.

For more information on how to set permissions on the folders and files, see Azure Synapse Analytics shared database.

Examples

Create a managed table in Spark and query from serverless SQL pool

In this scenario, you have a Spark database named `mytestdb`. See Create and connect to a Spark database with serverless SQL pool.

Create a managed Spark table with SparkSQL by running the following command:

```
CREATE TABLE mytestdb.myparquettable(id int, name string, birthdate date) USING Parquet
```

This command creates the table `myparquettable` in the database `mytestdb`. Table names will be converted to lowercase. After a short delay, you can see the table in your serverless SQL pool. For example, run the following statement from your serverless SQL pool.

```
USE mytestdb;
SELECT * FROM sys.tables;
```

Verify that `myparquettable` is included in the results.

> **NOTE**
> A table that is not using Delta, Parquet or CSV as its storage format will not be synchronized.

Next, insert some values into the table from Spark, for example with the following C# Spark statements in a C# notebook:

```
using Microsoft.Spark.Sql.Types;

var data = new List<GenericRow>();

data.Add(new GenericRow(new object[] { 1, "Alice", new Date(2010, 1, 1)}));
data.Add(new GenericRow(new object[] { 2, "Bob", new Date(1990, 1, 1)}));

var schema = new StructType
    (new List<StructField>()
        {
            new StructField("id", new IntegerType()),
            new StructField("name", new StringType()),
            new StructField("birthdate", new DateType())
        }
    );

var df = spark.CreateDataFrame(data, schema);
df.Write().Mode(SaveMode.Append).InsertInto("mytestdb.myparquettable");
```

Now you can read the data from your serverless SQL pool as follows:

```
SELECT * FROM mytestdb.dbo.myparquettable WHERE name = 'Alice';
```

You should get the following row as result:

```
id | name | birthdate
---+-------+-----------
1 | Alice | 2010-01-01
```

Create an external table in Spark and query from serverless SQL pool

In this example, we will create an external Spark table over the Parquet data files that got created in the previous example for the managed table.

For example, with SparkSQL run:

```
CREATE TABLE mytestdb.myexternalparquettable
    USING Parquet
    LOCATION "abfss://<storage-
name>.dfs.core.windows.net/<fs>/synapse/workspaces/<synapse_ws>/warehouse/mytestdb.db/myparquettable/"
```

Replace the placeholder `<storage-name>` with the ADLS Gen2 storage account name that you are using, `<fs>` with the file system name you're using and the placeholder `<synapse_ws>` with the name of the Azure Synapse workspace you're using to run this example.

The previous example creates the table `myextneralparquettable` in the database `mytestdb`. After a short delay, you can see the table in your serverless SQL pool. For example, run the following statement from your serverless SQL pool.

```
USE mytestdb;
SELECT * FROM sys.tables;
```

Verify that `myexternalparquettable` is included in the results.

Now you can read the data from your serverless SQL pool as follows:

```
SELECT * FROM mytestdb.dbo.myexternalparquettable WHERE name = 'Alice';
```

You should get the following row as result:

```
id | name | birthdate
---+-------+-----------
1 | Alice | 2010-01-01
```

Next steps

- Learn more about Azure Synapse Analytics' shared metadata
- Learn more about Azure Synapse Analytics' shared metadata database

Create, develop, and maintain Synapse notebooks in Azure Synapse Analytics

8/19/2022 • 19 minutes to read • Edit Online

A Synapse notebook is a web interface for you to create files that contain live code, visualizations, and narrative text. Notebooks are a good place to validate ideas and use quick experiments to get insights from your data. Notebooks are also widely used in data preparation, data visualization, machine learning, and other Big Data scenarios.

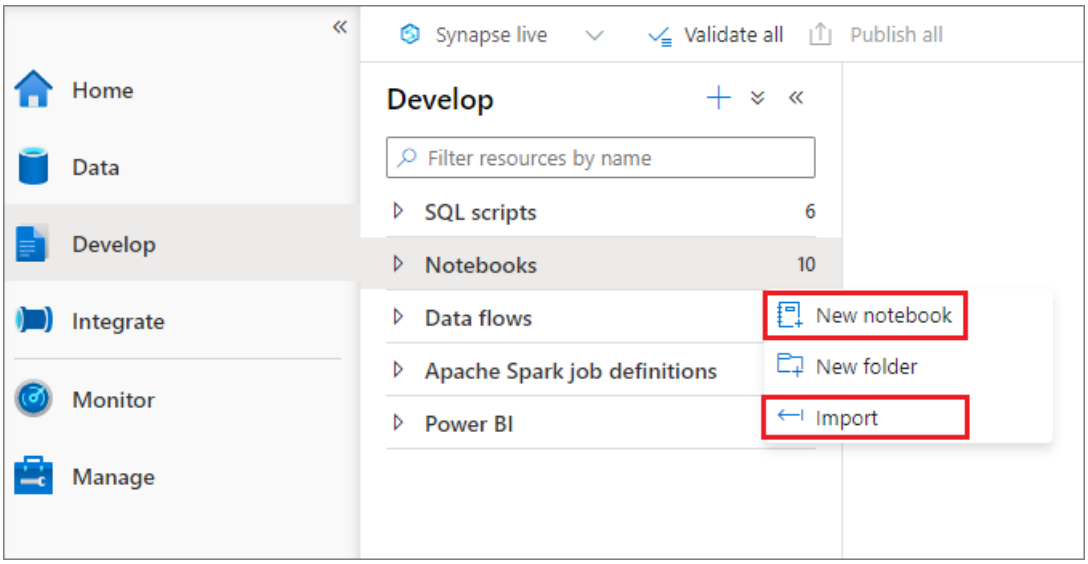
With a Synapse notebook, you can:

- Get started with zero setup effort.
- Keep data secure with built-in enterprise security features.
- Analyze data across raw formats (CSV, txt, JSON, etc.), processed file formats (parquet, Delta Lake, ORC, etc.), and SQL tabular data files against Spark and SQL.
- Be productive with enhanced authoring capabilities and built-in data visualization.

This article describes how to use notebooks in Synapse Studio.

Create a notebook

There are two ways to create a notebook. You can create a new notebook or import an existing notebook to a Synapse workspace from the **Object Explorer**. Synapse notebooks recognize standard Jupyter Notebook IPYNB files.



Develop notebooks

Notebooks consist of cells, which are individual blocks of code or text that can be run independently or as a group.

We provide rich operations to develop notebooks:

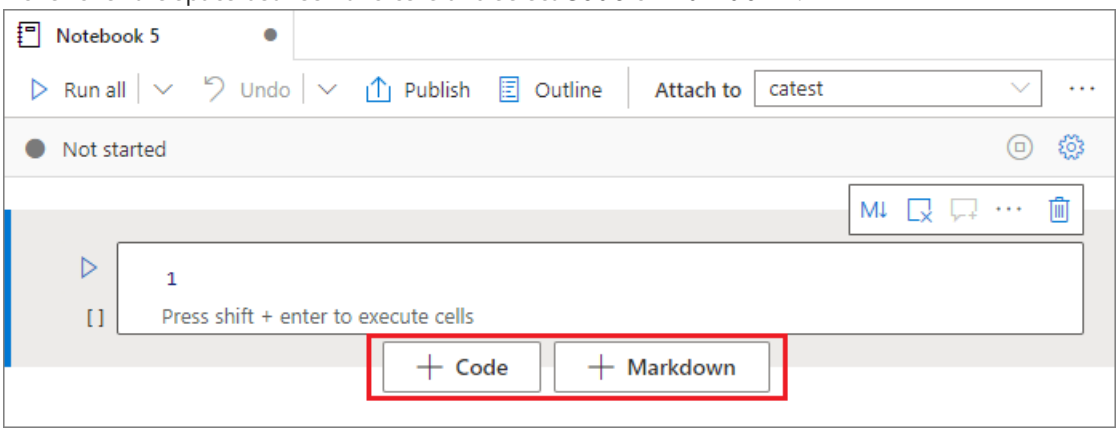
- Add a cell
- Set a primary language
- Use multiple languages

- Use temp tables to reference data across languages
- IDE-style IntelliSense
- Code Snippets
- Format text cell with toolbar buttons
- Undo/Redo cell operation
- Code cell commenting
- Move a cell
- Delete a cell
- Collapse a cell input
- Collapse a cell output
- Notebook outline

Add a cell

There are multiple ways to add a new cell to your notebook.

1. Hover over the space between two cells and select **Code** or **Markdown**.



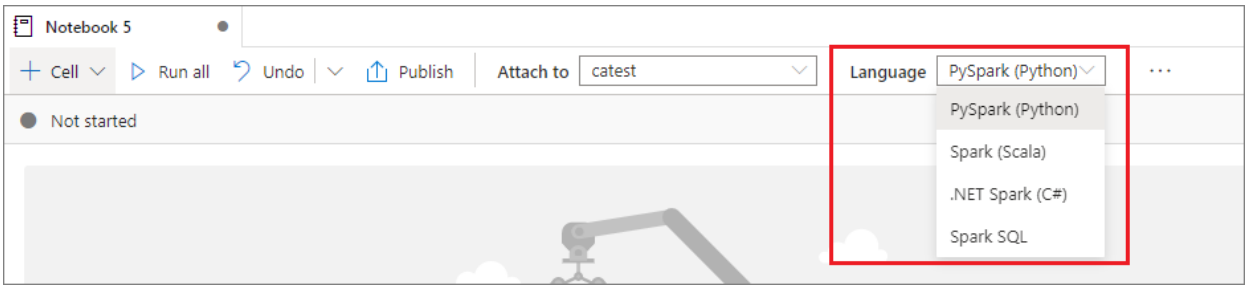
2. Use aznb Shortcut keys under command mode. Press **A** to insert a cell above the current cell. Press **B** to insert a cell below the current cell.

Set a primary language

Synapse notebooks support four Apache Spark languages:

- PySpark (Python)
- Spark (Scala)
- Spark SQL
- .NET Spark (C#)

You can set the primary language for new added cells from the dropdown list in the top command bar.

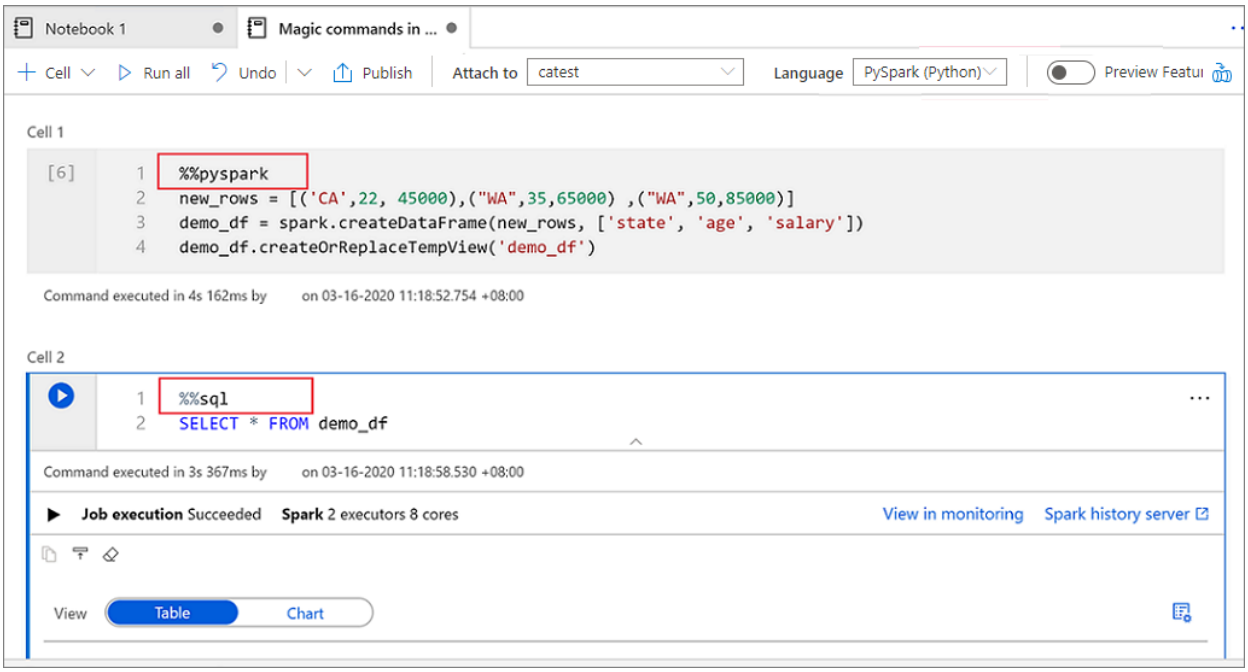


Use multiple languages

You can use multiple languages in one notebook by specifying the correct language magic command at the beginning of a cell. The following table lists the magic commands to switch cell languages.

MAGIC COMMAND	LANGUAGE	DESCRIPTION
%%pyspark	Python	Execute a **Python** query against Spark Context.
%%spark	Scala	Execute a **Scala** query against Spark Context.
%%sql	SparkSQL	Execute a **SparkSQL** query against Spark Context.
%%csharp	.NET for Spark C#	Execute a **.NET for Spark C#** query against Spark Context.

The following image is an example of how you can write a PySpark query using the **%%pyspark** magic command or a SparkSQL query with the **%%sql** magic command in a **Spark(Scala)** notebook. Notice that the primary language for the notebook is set to pySpark.



Use temp tables to reference data across languages

You cannot reference data or variables directly across different languages in a Synapse notebook. In Spark, a temporary table can be referenced across languages. Here is an example of how to read a `Scala` DataFrame in `PySpark` and `SparkSQL` using a Spark temp table as a workaround.

1. In Cell 1, read a DataFrame from a SQL pool connector using Scala and create a temporary table.

```
%%spark
val scalaDataFrame = spark.read.sqlanalytics("mySQLPoolDatabase.dbo.mySQLPoolTable")
scalaDataFrame.createOrReplaceTempView( "mydataframetable" )
```

2. In Cell 2, query the data using Spark SQL.

```
%%sql
SELECT * FROM mydataframetable
```

3. In Cell 3, use the data in PySpark.

```
%%pyspark
myNewPythonDataFrame = spark.sql("SELECT * FROM mydataframetable")
```

IDE-style IntelliSense

Synapse notebooks are integrated with the Monaco editor to bring IDE-style IntelliSense to the cell editor. Syntax highlight, error marker, and automatic code completions help you to write code and identify issues quicker.

The IntelliSense features are at different levels of maturity for different languages. Use the following table to see what's supported.

LANGUAGES	SYNTAX HIGHLIGHT	SYNTAX ERROR MARKER	SYNTAX CODE COMPLETION	VARIABLE CODE COMPLETION	SYSTEM FUNCTION CODE COMPLETION	USER FUNCTION CODE COMPLETION	SMART INDENT	CODE FOLDING
PySpark (Python)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Spark (Scala)	Yes	Yes	Yes	Yes	Yes	Yes	-	Yes
SparkSQL	Yes	Yes	Yes	Yes	Yes	-	-	-
.NET for Spark (C#)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

> **NOTE**
> An active Spark session is required to benefit the Variable Code Completion, System Function Code Completion, User Function Code Completion for .NET for Spark (C#).

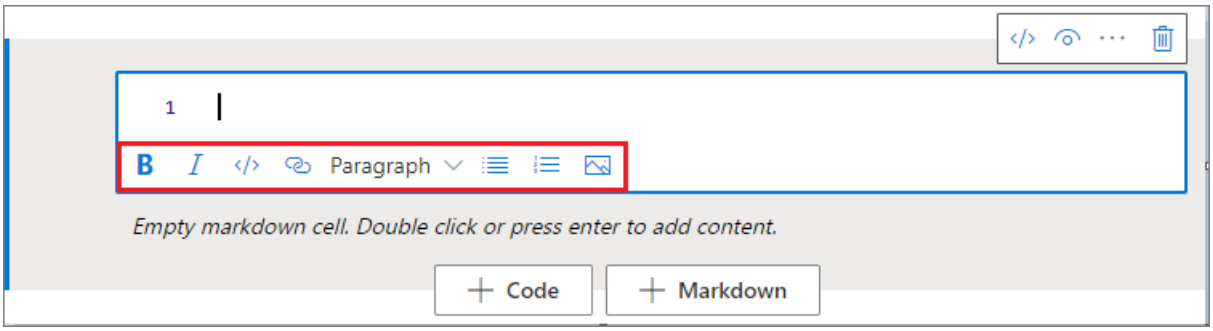
Code Snippets

Synapse notebooks provide code snippets that make it easier to enter common used code patterns, such as configuring your Spark session, reading data as a Spark DataFrame, or drawing charts with matplotlib etc.

Snippets appear in Shortcut keys of IDE style IntelliSense mixed with other suggestions. The code snippets contents align with the code cell language. You can see available snippets by typing **Snippet** or any keywords appear in the snippet title in the code cell editor. For example, by typing **read** you can see the list of snippets to read data from various data sources.

Format text cell with toolbar buttons

You can use the format buttons in the text cells toolbar to do common markdown actions. It includes bolding text, italicizing text, paragraph/headers through a dropdown, inserting code, inserting unordered list, inserting ordered list, inserting hyperlink and inserting image from URL.



Undo/Redo cell operation

Select the **Undo** / **Redo** button or press **Z** / **Shift+Z** to revoke the most recent cell operations. Now you can undo/redo up to the latest 10 historical cell operations.

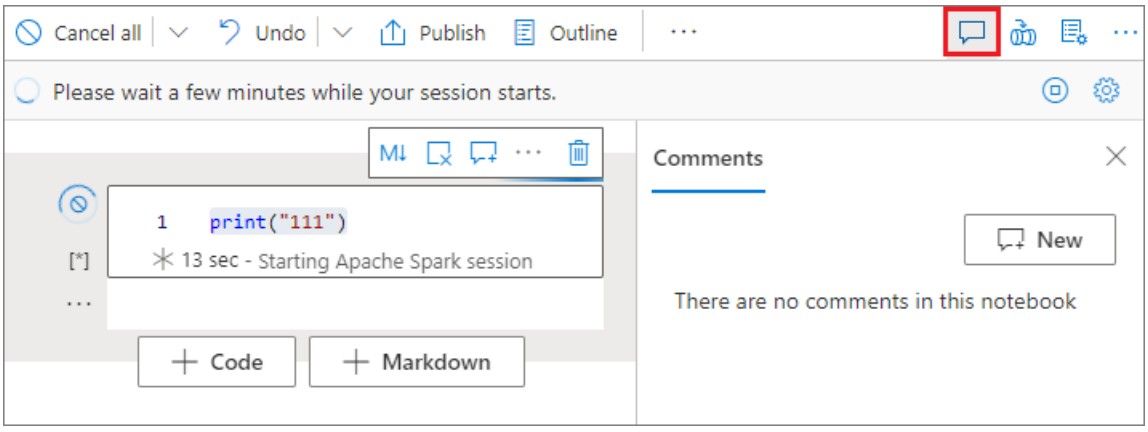
Supported undo cell operations:

- Insert/Delete cell: You could revoke the delete operations by selecting **Undo**, the text content will be kept along with the cell.
- Reorder cell.
- Toggle parameter.
- Convert between Code cell and Markdown cell.

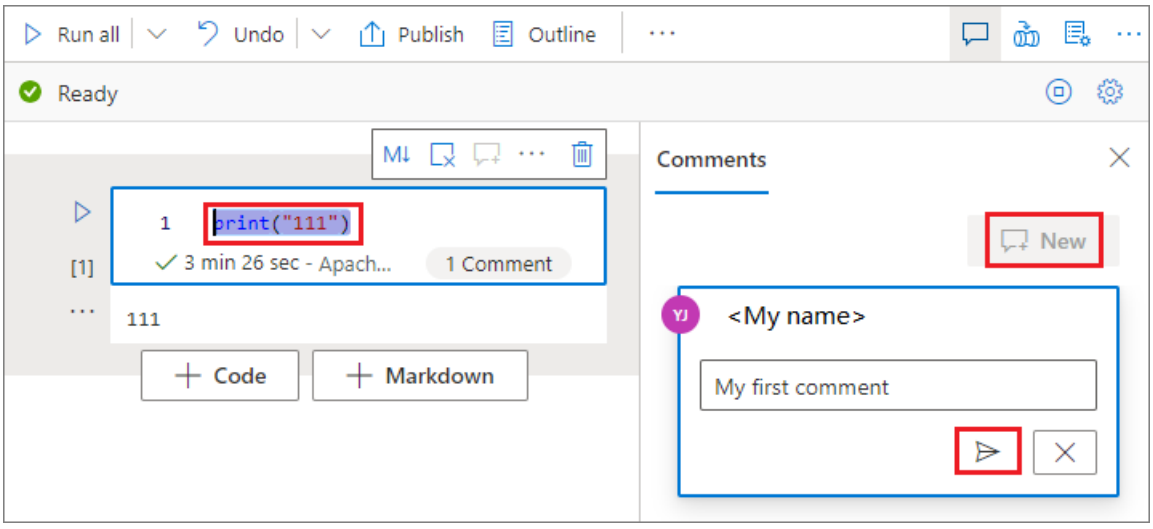
> **NOTE**
> In-cell text operations and code cell commenting operations are not undoable. Now you can undo/redo up to the latest 10 historical cell operations.

Code cell commenting

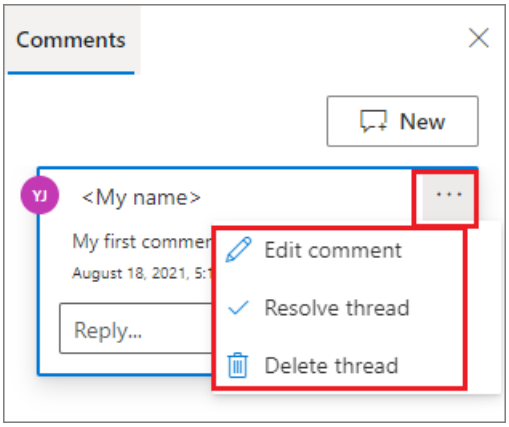
1. Select **Comments** button on the notebook toolbar to open **Comments** pane.



2. Select code in the code cell, click **New** in the **Comments** pane, add comments then click **Post comment** button to save.

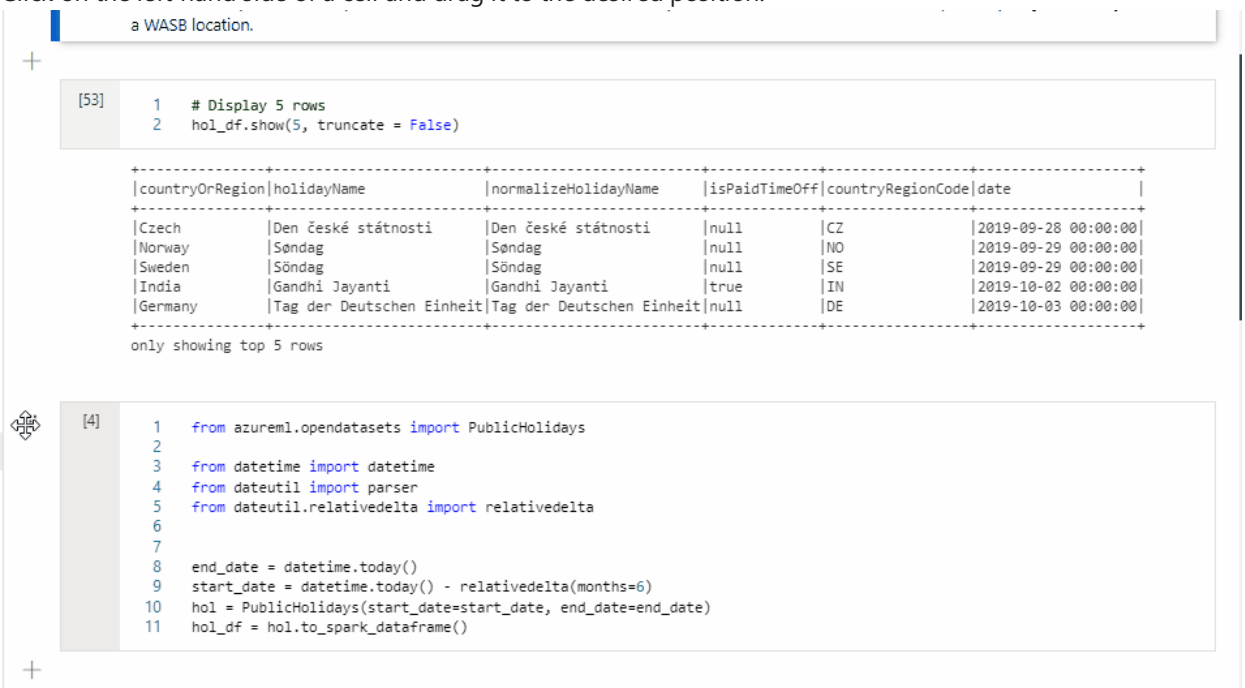


3. You could perform **Edit comment**, **Resolve thread**, or **Delete thread** by clicking the **More** button besides your comment.



Move a cell

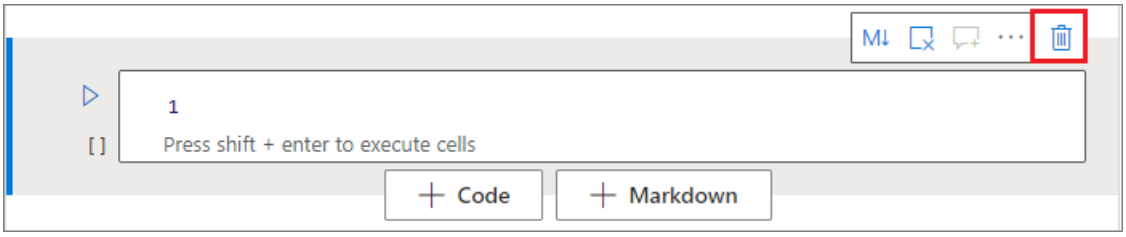
Click on the left-hand side of a cell and drag it to the desired position.



Delete a cell

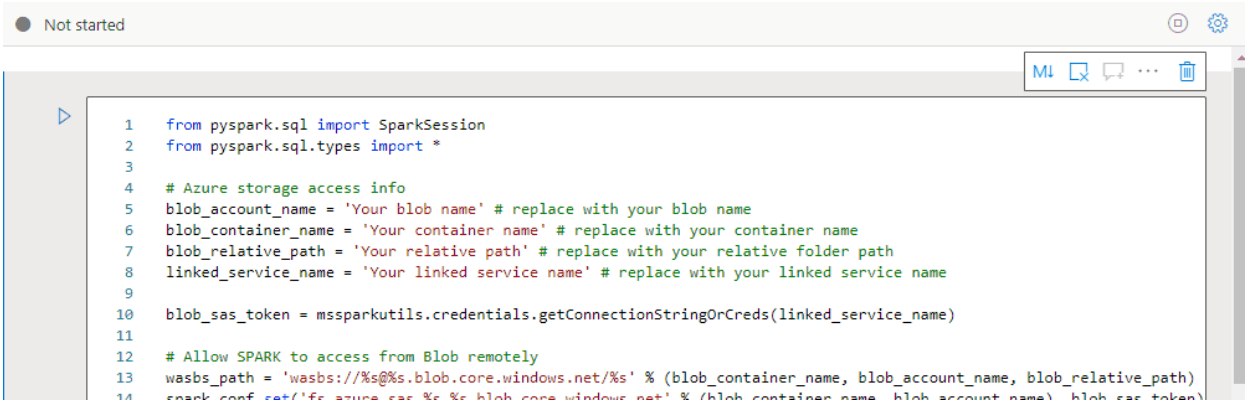
To delete a cell, select the delete button at the right hand of the cell.

You can also use shortcut keys under command mode. Press **Shift+D** to delete the current cell.



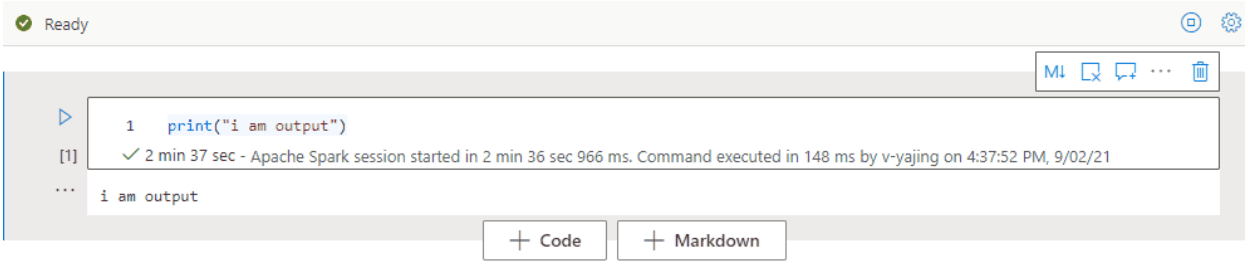
Collapse a cell input

Select the **More commands** ellipses (...) on the cell toolbar and **Hide input** to collapse current cell's input. To expand it, Select the **Show input** while the cell is collapsed.



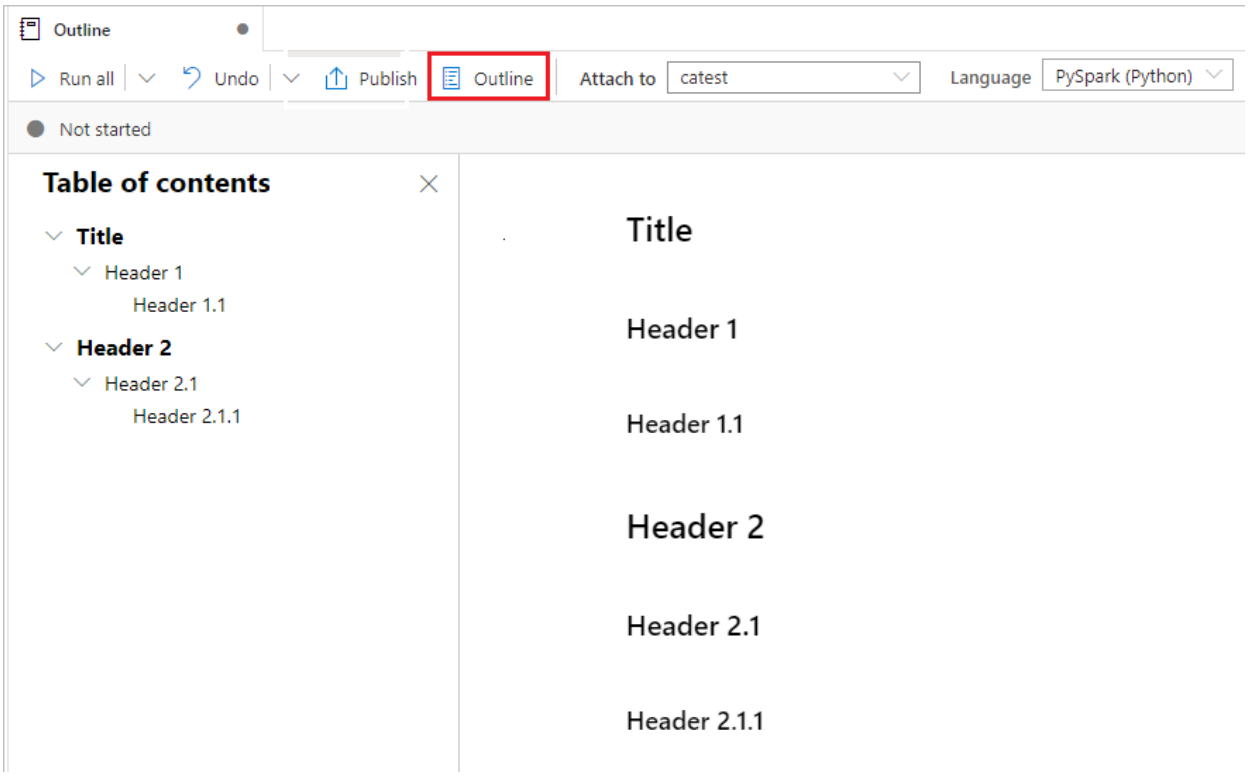
Collapse a cell output

Select the **More commands** ellipses (...) on the cell toolbar and **Hide output** to collapse current cell's output. To expand it, select the **Show output** while the cell's output is hidden.



Notebook outline

The Outlines (Table of Contents) presents the first markdown header of any markdown cell in a sidebar window for quick navigation. The Outlines sidebar is resizable and collapsible to fit the screen in the best ways possible. You can select the **Outline** button on the notebook command bar to open or hide sidebar



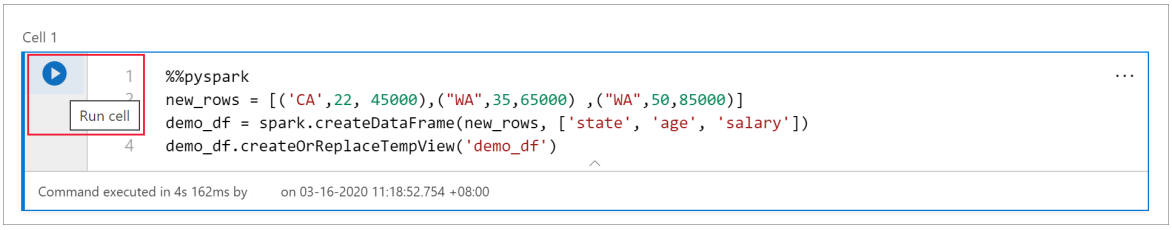
Run notebooks

You can run the code cells in your notebook individually or all at once. The status and progress of each cell is represented in the notebook.

Run a cell

There are several ways to run the code in a cell.

1. Hover on the cell you want to run and select the **Run Cell** button or press **Ctrl+Enter**.



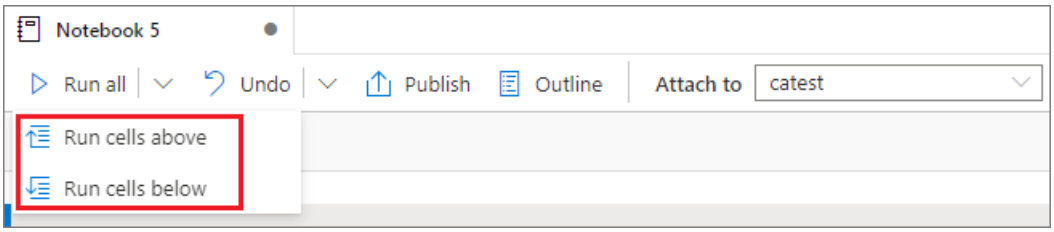
2. Use Shortcut keys under command mode. Press **Shift+Enter** to run the current cell and select the cell below. Press **Alt+Enter** to run the current cell and insert a new cell below.

Run all cells

Select the **Run All** button to run all the cells in current notebook in sequence.

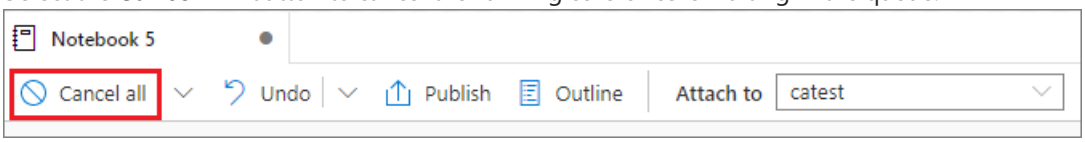
Run all cells above or below

Expand the dropdown list from **Run all** button, then select **Run cells above** to run all the cells above the current in sequence. Select **Run cells below** to run all the cells below the current in sequence.



Cancel all running cells

Select the **Cancel All** button to cancel the running cells or cells waiting in the queue.



Notebook reference

You can use `%run <notebook path>` magic command to reference another notebook within current notebook's context. All the variables defined in the reference notebook are available in the current notebook. `%run` magic command supports nested calls but not support recursive calls. You will receive an exception if the statement depth is larger than **five**.

Example:

```
%run /<path>/Notebook1 { "parameterInt": 1, "parameterFloat": 2.5, "parameterBool": true, "parameterString": "abc" }
```

.

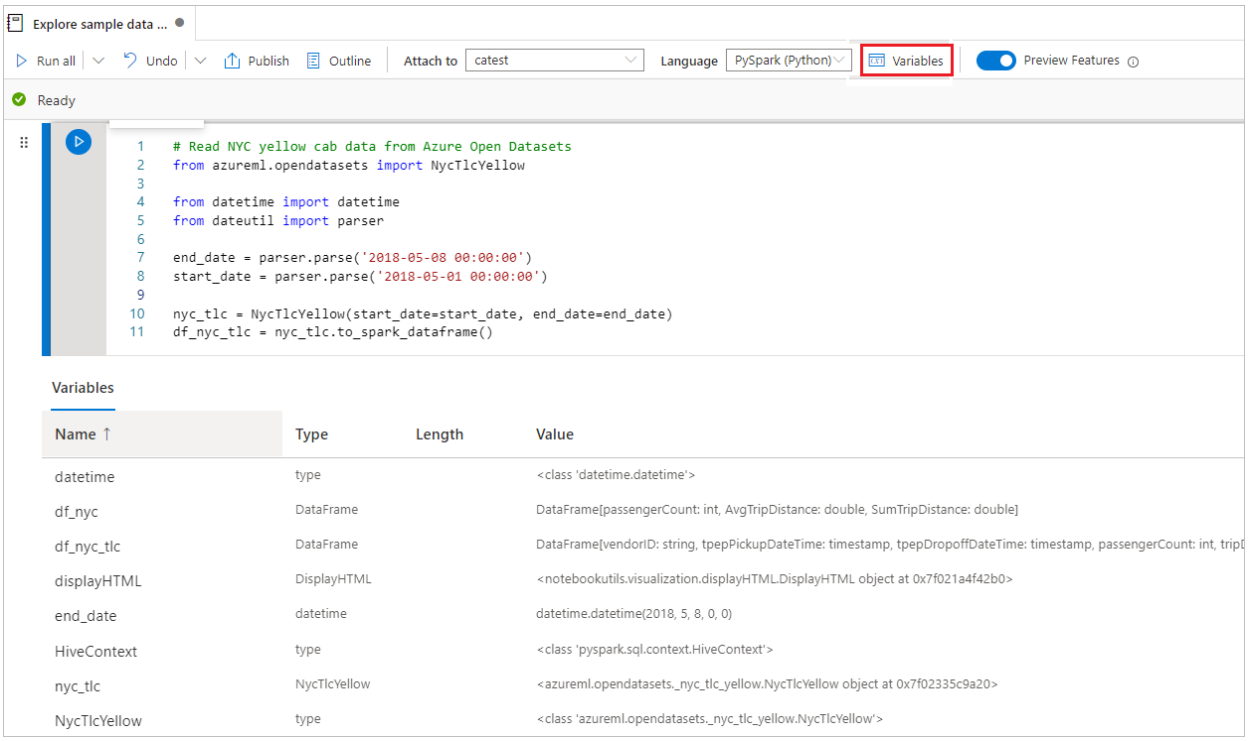
Notebook reference works in both interactive mode and Synapse pipeline.

> **NOTE**
>
> - `%run` command currently only supports to pass a absolute path or notebook name only as parameter, relative path is not supported.
> - `%run` command currently only supports to 4 parameter value types: `int`, `float`, `bool`, `string`, variable replacement operation is not supported.
> - The referenced notebooks are required to be published. You need to publish the notebooks to reference them unless Reference unpublished notebook is enabled. Synapse Studio does not recognize the unpublished notebooks from the Git repo.
> - Referenced notebooks do not support statement that depth is larger than **five**.

Variable explorer

Synapse notebook provides a built-in variables explorer for you to see the list of the variables name, type, length, and value in the current Spark session for PySpark (Python) cells. More variables will show up automatically as they are defined in the code cells. Clicking on each column header will sort the variables in the table.

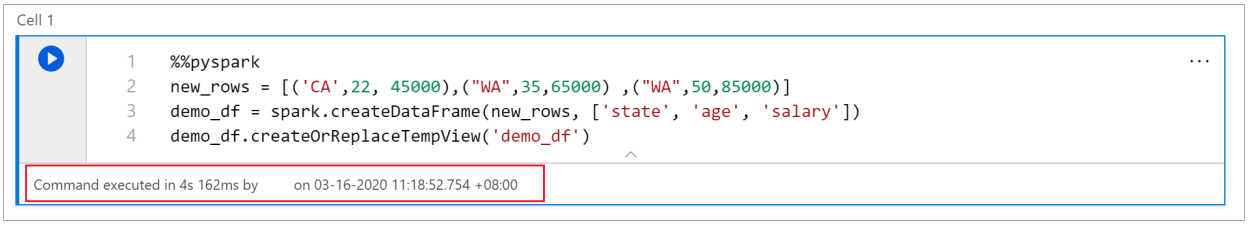
You can select the **Variables** button on the notebook command bar to open or hide the variable explorer.



> **NOTE**
>
> Variable explorer only supports python.

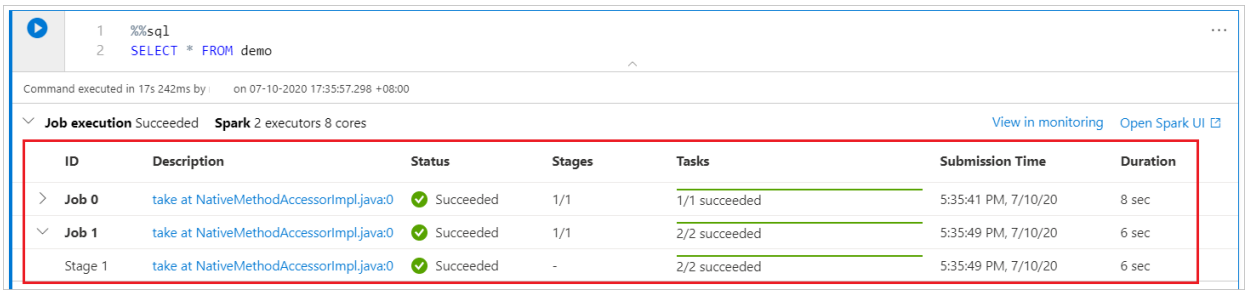
Cell status indicator

A step-by-step cell execution status is displayed beneath the cell to help you see its current progress. Once the cell run is complete, an execution summary with the total duration and end time are shown and kept there for future reference.



Spark progress indicator

Synapse notebook is purely Spark based. Code cells are executed on the serverless Apache Spark pool remotely. A Spark job progress indicator is provided with a real-time progress bar appears to help you understand the job execution status. The number of tasks per each job or stage help you to identify the parallel level of your spark job. You can also drill deeper to the Spark UI of a specific job (or stage) via selecting the link on the job (or stage) name.

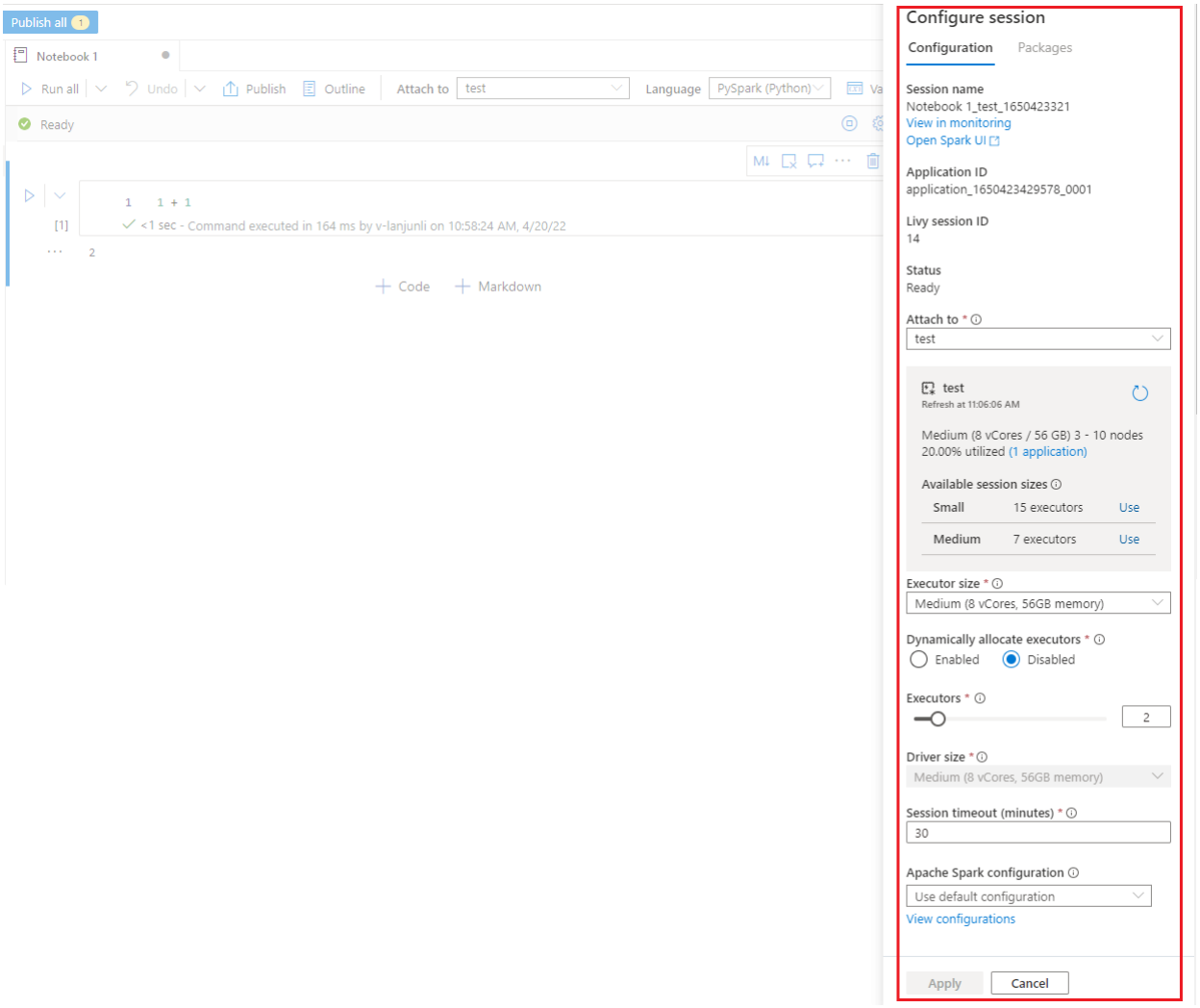


Spark session configuration

You can specify the timeout duration, the number, and the size of executors to give to the current Spark session

in **Configure session**. Restart the Spark session is for configuration changes to take effect. All cached notebook variables are cleared.

You can also create a configuration from the Apache Spark configuration or select an existing configuration. For details, please refer to Apache Spark Configuration Management.



Spark session configuration magic command

You can also specify spark session settings via a magic command **%%configure**. The spark session needs to restart to make the settings effect. We recommend you to run the **%%configure** at the beginning of your notebook. Here is a sample, refer to https://github.com/cloudera/livy#request-body for full list of valid parameters.

```
%%configure
{
    //You can get a list of valid parameters to config the session from
https://github.com/cloudera/livy#request-body.
    "driverMemory":"28g", // Recommended values: ["28g", "56g", "112g", "224g", "400g", "472g"]
    "driverCores":4, // Recommended values: [4, 8, 16, 32, 64, 80]
    "executorMemory":"28g",
    "executorCores":4,
    "jars":
["abfs[s]://<file_system>@<account_name>.dfs.core.windows.net/<path>/myjar.jar","wasb[s]://<containername>@<
accountname>.blob.core.windows.net/<path>/myjar1.jar"],
    "conf":{
    //Example of standard spark property, to find more available properties please
visit:https://spark.apache.org/docs/latest/configuration.html#application-properties.
        "spark.driver.maxResultSize":"10g",
    //Example of customized property, you can specify count of lines that Spark SQL returns by configuring
"livy.rsc.sql.num-rows".
        "livy.rsc.sql.num-rows":"3000"
    }
}
```

NOTE

- "DriverMemory" and "ExecutorMemory" are recommended to set as same value in %%configure, so do "driverCores" and "executorCores".
- You can use %%configure in Synapse pipelines, but if it's not set in the first code cell, the pipeline run will fail due to cannot restart session.
- The %%configure used in mssparkutils.notebook.run is going to be ignored but used in %run notebook will continue executing.
- The standard Spark configuration properties must be used in the "conf" body. We do not support first level reference for the Spark configuration properties.
- Some special spark properties including "spark.driver.cores", "spark.executor.cores", "spark.driver.memory", "spark.executor.memory", "spark.executor.instances" won't take effect in "conf" body.

Parameterized session configuration from pipeline

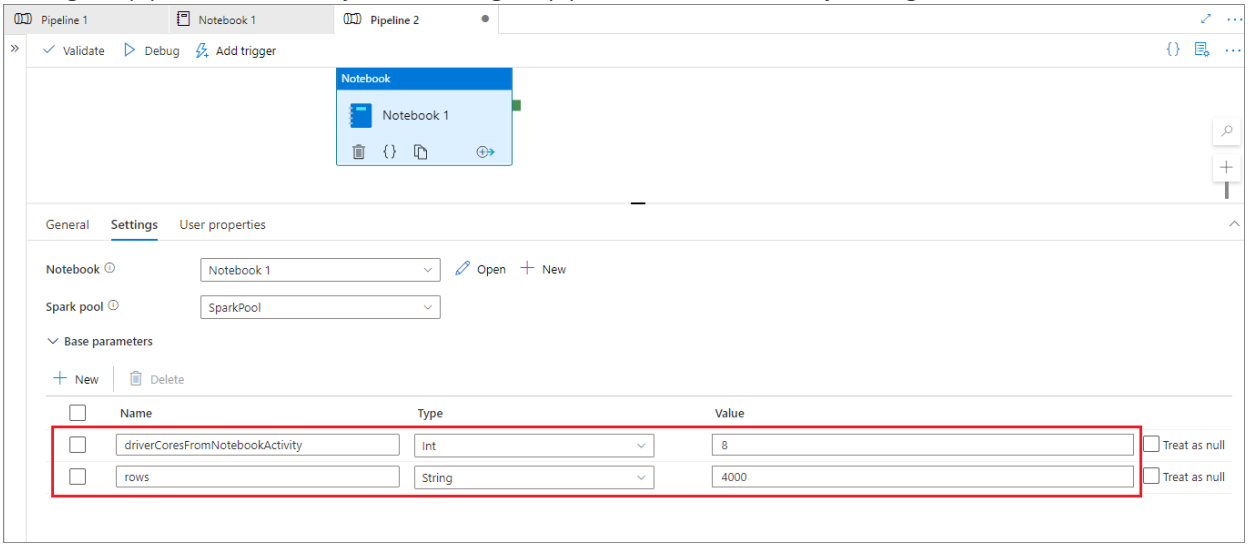
Parameterized session configuration allows you to replace the value in %%configure magic with Pipeline run (Notebook activity) parameters. When preparing %%configure code cell, you can override default values (also configurable, 4 and "2000" in the below example) with an object like this:

```
{
      "activityParameterName": "paramterNameInPipelineNotebookActivity",
      "defaultValue": "defaultValueIfNoParamterFromPipelineNotebookActivity"
}
```

```
%%configure

{
    "driverCores":
    {
        "activityParameterName": "driverCoresFromNotebookActivity",
        "defaultValue": 4
    },
    "conf":
    {
        "livy.rsc.sql.num-rows":
        {
            "activityParameterName": "rows",
            "defaultValue": "2000"
        }
    }
}
```

Notebook will use default value if run a notebook in interactive mode directly or no parameter that match "activityParameterName" is given from Pipeline Notebook activity.

During the pipeline run mode, you can configure pipeline Notebook activity settings as below:



If you want to change the session configuration, pipeline Notebook activity parameters name should be same as activityParameterName in the notebook. When running this pipeline, in this example driverCores in %%configure will be replaced by 8 and livy.rsc.sql.num-rows will be replaced by 4000.

> **NOTE**
> If run pipeline failed because of using this new %%configure magic, you can check more error information by running %%configure magic cell in the interactive mode of the notebook.

Bring data to a notebook

You can load data from Azure Blob Storage, Azure Data Lake Store Gen 2, and SQL pool as shown in the code samples below.

Read a CSV from Azure Data Lake Store Gen2 as a Spark DataFrame

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
account_name = "Your account name"
container_name = "Your container name"
relative_path = "Your path"
adls_path = 'abfss://%s@%s.dfs.core.windows.net/%s' % (container_name, account_name, relative_path)

df1 = spark.read.option('header', 'true') \
                .option('delimiter', ',') \
                .csv(adls_path + '/Testfile.csv')
```

Read a CSV from Azure Blob Storage as a Spark DataFrame

```
from pyspark.sql import SparkSession

# Azure storage access info
blob_account_name = 'Your account name' # replace with your blob name
blob_container_name = 'Your container name' # replace with your container name
blob_relative_path = 'Your path' # replace with your relative folder path
linked_service_name = 'Your linked service name' # replace with your linked service name

blob_sas_token = mssparkutils.credentials.getConnectionStringOrCreds(linked_service_name)

# Allow SPARK to access from Blob remotely

wasb_path = 'wasbs://%s@%s.blob.core.windows.net/%s' % (blob_container_name, blob_account_name,
blob_relative_path)

spark.conf.set('fs.azure.sas.%s.%s.blob.core.windows.net' % (blob_container_name, blob_account_name),
blob_sas_token)
print('Remote blob path: ' + wasb_path)

df = spark.read.option("header", "true") \
            .option("delimiter","|") \
            .schema(schema) \
            .csv(wasbs_path)
```

Read data from the primary storage account

You can access data in the primary storage account directly. There's no need to provide the secret keys. In Data Explorer, right-click on a file and select **New notebook** to see a new notebook with data extractor autogenerated.

IPython Widgets

Widgets are eventful python objects that have a representation in the browser, often as a control like a slider, textbox etc. IPython Widgets only works in Python environment, it's not supported in other languages (e.g. Scala, SQL, C#) yet.

To use IPython Widget

1. You need to import `ipywidgets` module first to use the Jupyter Widget framework.

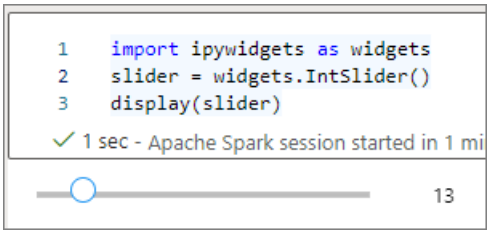
```
import ipywidgets as widgets
```

2. You can use top-level `display` function to render a widget, or leave an expression of **widget** type at the last line of code cell.

```
slider = widgets.IntSlider()
display(slider)
```

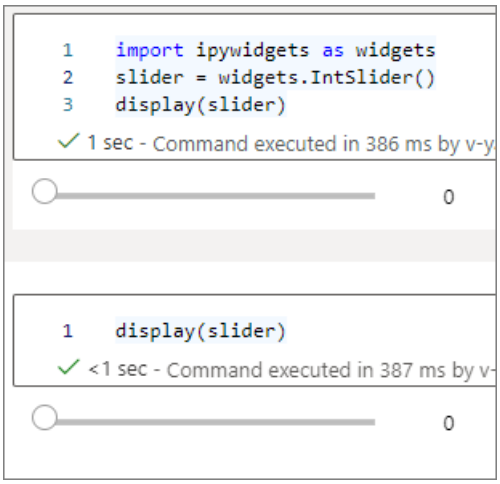
```
slider = widgets.IntSlider()
slider
```

3. Run the cell, the widget will display at the output area.



4. You can use multiple `display()` calls to render the same widget instance multiple times, but they will remain in sync with each other.

```
slider = widgets.IntSlider()
display(slider)
display(slider)
```



5. To render two widgets independent of each other, create two widget instances:

```
slider1 = widgets.IntSlider()
slider2 = widgets.IntSlider()
display(slider1)
display(slider2)
```

Supported widgets

WIDGETS TYPE	WIDGETS
Numeric widgets	IntSlider, FloatSlider, FloatLogSlider, IntRangeSlider, FloatRangeSlider, IntProgress, FloatProgress, BoundedIntText, BoundedFloatText, IntText, FloatText
Boolean widgets	ToggleButton, Checkbox, Valid
Selection widgets	Dropdown, RadioButtons, Select, SelectionSlider, SelectionRangeSlider, ToggleButtons, SelectMultiple
String Widgets	Text, Text area, Combobox, Password, Label, HTML, HTML Math, Image, Button
Play (Animation) widgets	Date picker, Color picker, Controller
Container/Layout widgets	Box, HBox, VBox, GridBox, Accordion, Tabs, Stacked

Known limitations

1. The following widgets are not supported yet, you could follow the corresponding workaround as below:

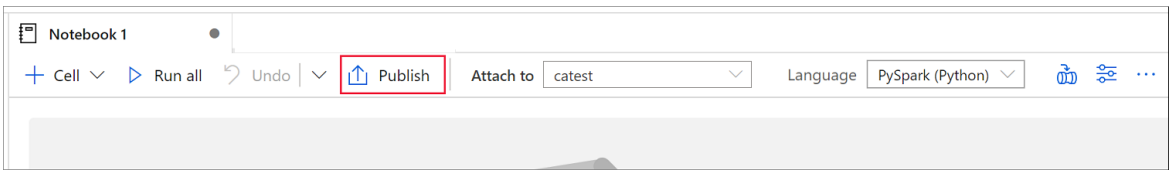
FUNCTIONALITY	WORKAROUND
`Output` widget	You can use `print()` function instead to write text into stdout.
`widgets.jslink()`	You can use `widgets.link()` function to link two similar widgets.
`FileUpload` widget	Not support yet.

2. Global `display` function provided by Synapse does not support displaying multiple widgets in 1 call (i.e. `display(a, b)`), which is different from IPython `display` function.
3. If you close a notebook that contains IPython Widget, you will not be able to see or interact with it until you execute the corresponding cell again.

Save notebooks

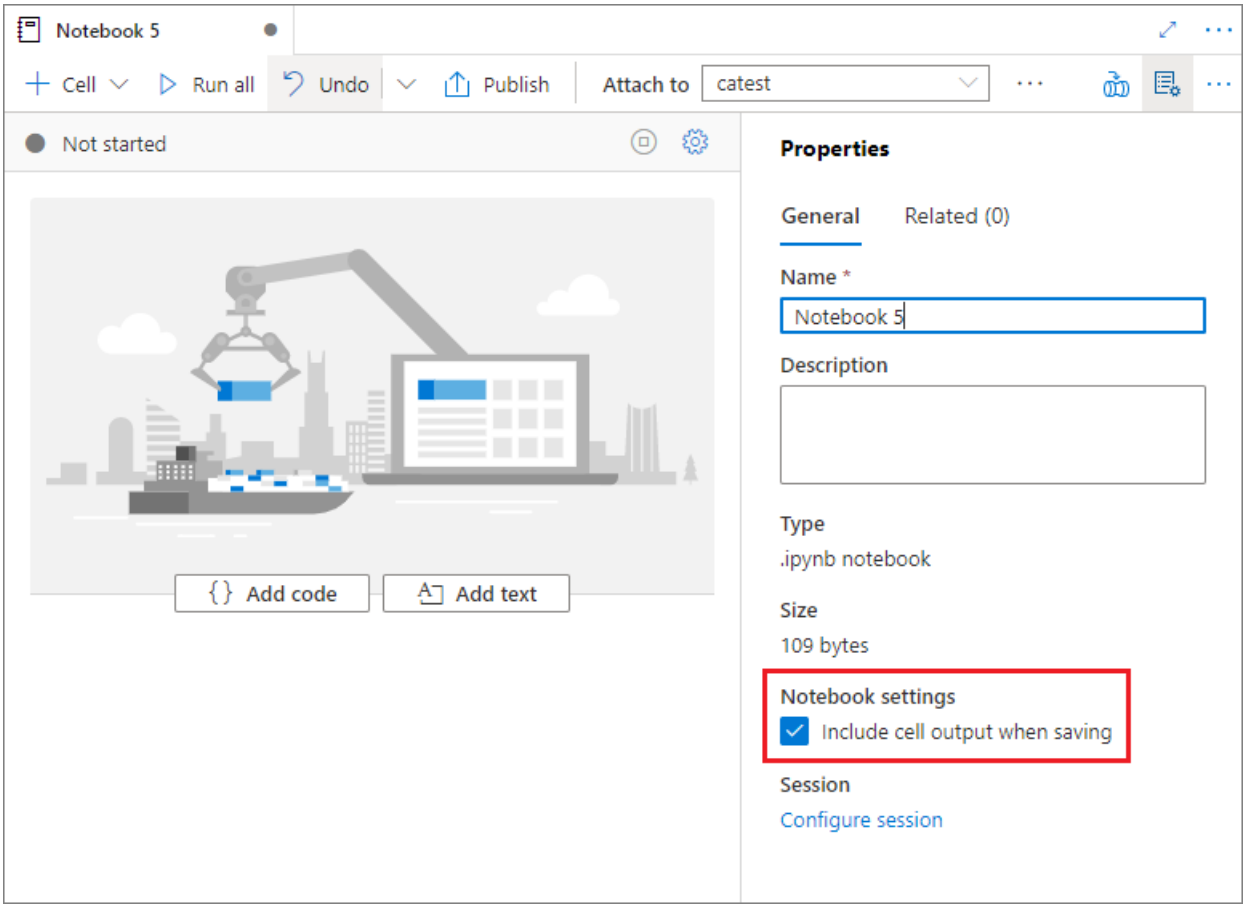
You can save a single notebook or all notebooks in your workspace.

1. To save changes you made to a single notebook, select the **Publish** button on the notebook command bar.



2. To save all notebooks in your workspace, select the **Publish all** button on the workspace command bar.

In the notebook properties, you can configure whether to include the cell output when saving.



Magic commands

You can use familiar Jupyter magic commands in Synapse notebooks. Review the following list as the current available magic commands. Tell us your use cases on GitHub so that we can continue to build out more magic commands to meet your needs.

> **NOTE**
> Only following magic commands are supported in Synapse pipeline : %%pyspark, %%spark, %%csharp, %%sql.

Available line magics: %lsmagic, %time, %timeit, %history, %run, %load

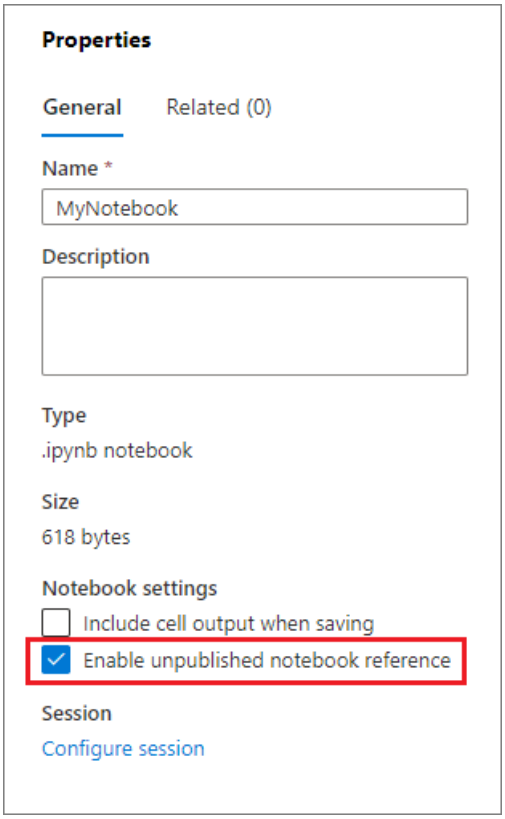
Available cell magics: %%time, %%timeit, %%capture, %%writefile, %%sql, %%pyspark, %%spark, %%csharp, %%html, %%configure

Reference unpublished notebook

Reference unpublished notebook is helpful when you want to debug "locally", when enabling this feature, notebook run will fetch the current content in web cache, if you run a cell including a reference notebooks statement, you will reference the presenting notebooks in the current notebook browser instead of a saved versions in cluster, that means the changes in your notebook editor can be referenced immediately by other

notebooks without having to be published(Live mode) or committed(Git mode), by leveraging this approach you can easily avoid common libraries getting polluted during developing or debugging process.

You can enable Reference unpublished notebook from Properties panel:



For different cases comparison please check the table below:

Notice that %run and mssparkutils.notebook.run has same behavior here. We use `%run` here as an example.

CASE	DISABLE	ENABLE
Live Mode		
- Nb1 (Published) `%run Nb1`	Run published version of Nb1	Run published version of Nb1
- Nb1 (New) `%run Nb1`	Error	Run new Nb1
- Nb1 (Previously published, edited) `%run Nb1`	Run **published** version of Nb1	Run **edited** version of Nb1
Git Mode		
- Nb1 (Published) `%run Nb1`	Run published version of Nb1	Run published version of Nb1
- Nb1 (New) `%run Nb1`	Error	Run new Nb1
- Nb1 (Not published, committed) `%run Nb1`	Error	Run committed Nb1

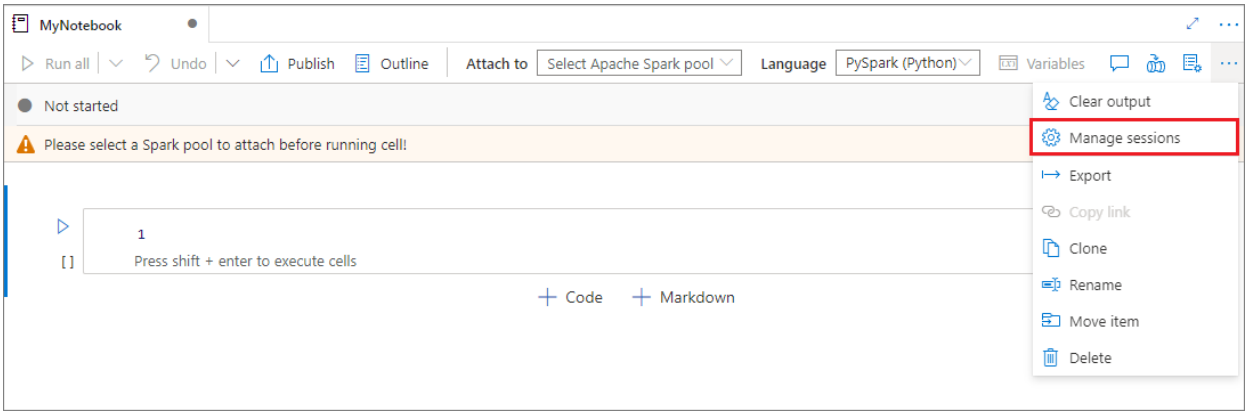
CASE	DISABLE	ENABLE
- Nb1 (Previously published, committed) `%run Nb1`	Run **published** version of Nb1	Run **committed** version of Nb1
- Nb1 (Previously published, new in current branch) `%run Nb1`	Run **published** version of Nb1	Run **new** Nb1
- Nb1 (Not published, previously committed, edited) `%run Nb1`	Error	Run **edited** version of Nb1
- Nb1 (Previously published and committed, edited) `%run Nb1`	Run **published** version of Nb1	Run **edited** version of Nb1

Conclusion

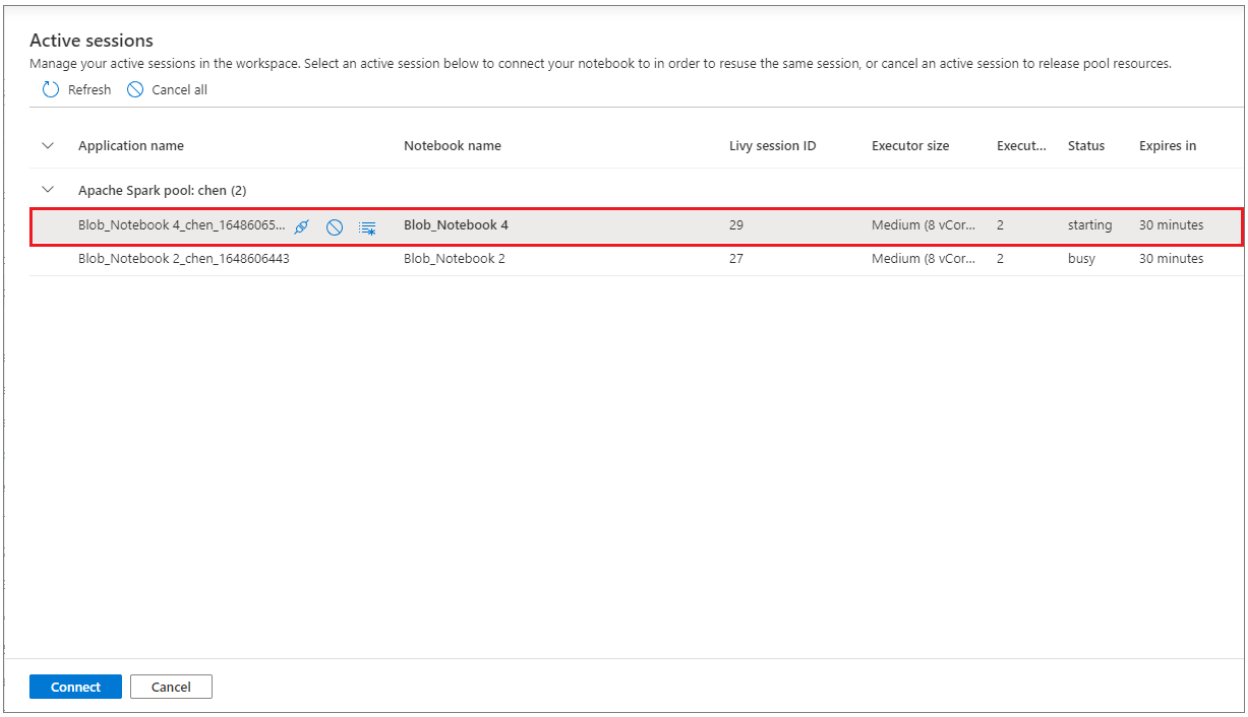
- If disabled, always run **published** version.
- If enabled, priority is: edited / new > committed > published.

Active session management

You can reuse your notebook sessions conveniently now without having to start new ones. Synapse notebook now supports managing your active sessions in the **Manage sessions** list, you can see all the sessions in the current workspace started by you from notebook.



In the **Active sessions** list you can see the session information and the corresponding notebook that is currently attached to the session. You can operate Detach with notebook, Stop the session, and View in monitoring from here. Moreover, you can easily connect your selected notebook to an active session in the list started from another notebook, the session will be detached from the previous notebook (if it's not idle) then attach to the current one.



Python logging in Notebook

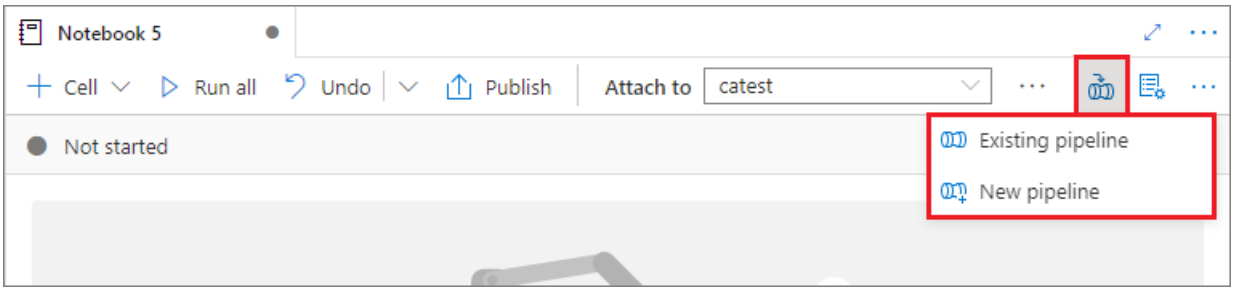
You can find Python logs and set different log levels and format following the sample code below:

```
import logging

# Customize the logging format for all loggers
FORMAT = "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
formatter = logging.Formatter(fmt=FORMAT)
for handler in logging.getLogger().handlers:
    handler.setFormatter(formatter)

# Customize log level for all loggers
logging.getLogger().setLevel(logging.INFO)

# Customize the log level for a specific logger
customizedLogger = logging.getLogger('customized')
customizedLogger.setLevel(logging.WARNING)

# logger that use the default global log level
defaultLogger = logging.getLogger('default')
defaultLogger.debug("default debug message")
defaultLogger.info("default info message")
defaultLogger.warning("default warning message")
defaultLogger.error("default error message")
defaultLogger.critical("default critical message")

# logger that use the customized log level
customizedLogger.debug("customized debug message")
customizedLogger.info("customized info message")
customizedLogger.warning("customized warning message")
customizedLogger.error("customized error message")
customizedLogger.critical("customized critical message")
```

Integrate a notebook

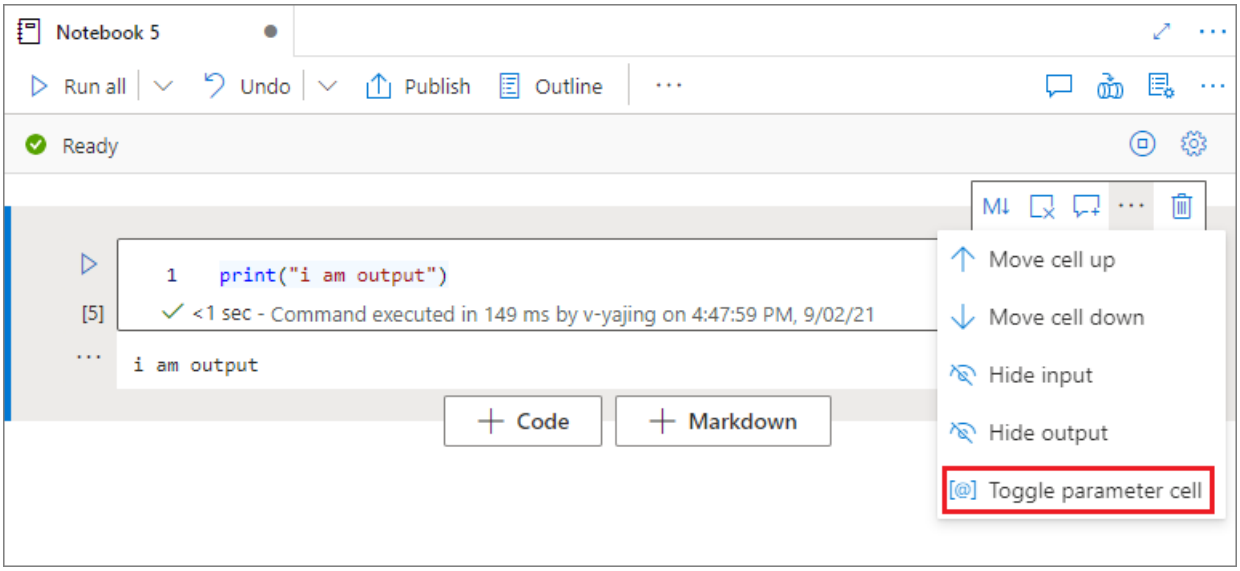
Add a notebook to a pipeline

Select the **Add to pipeline** button on the upper right corner to add a notebook to an existing pipeline or create a new pipeline.



Designate a parameters cell

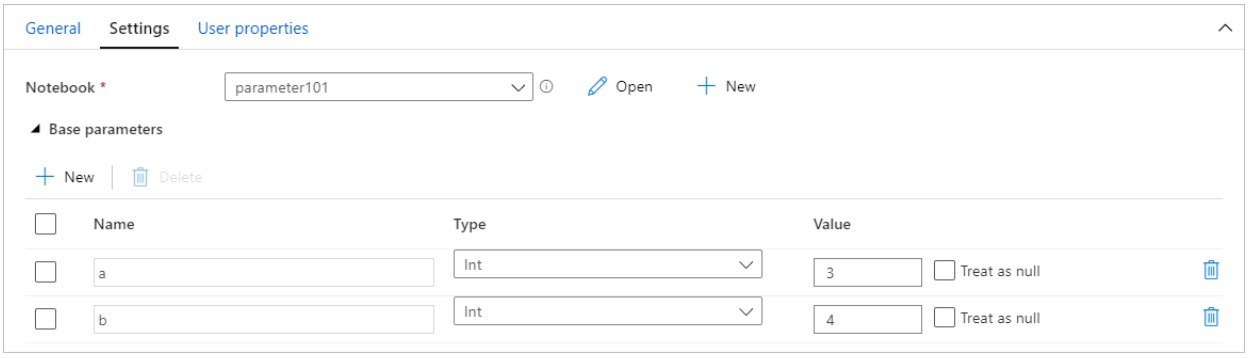
To parameterize your notebook, select the ellipses (...) to access the **more commands** at the cell toolbar. Then select **Toggle parameter cell** to designate the cell as the parameters cell.



Azure Data Factory looks for the parameters cell and treats this cell as defaults for the parameters passed in at execution time. The execution engine will add a new cell beneath the parameters cell with input parameters in order to overwrite the default values.

Assign parameters values from a pipeline

Once you've created a notebook with parameters, you can execute it from a pipeline with the Synapse Notebook activity. After you add the activity to your pipeline canvas, you will be able to set the parameters values under **Base parameters** section on the **Settings** tab.

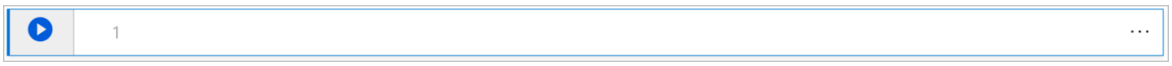


When assigning parameter values, you can use the pipeline expression language or system variables.

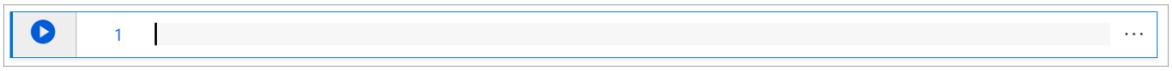
Shortcut keys

Similar to Jupyter Notebooks, Synapse notebooks have a modal user interface. The keyboard does different things depending on which mode the notebook cell is in. Synapse notebooks support the following two modes for a given code cell: command mode and edit mode.

1. A cell is in command mode when there is no text cursor prompting you to type. When a cell is in Command mode, you can edit the notebook as a whole but not type into individual cells. Enter command mode by pressing `ESC` or using the mouse to select outside of a cell's editor area.



2. Edit mode is indicated by a text cursor prompting you to type in the editor area. When a cell is in edit mode, you can type into the cell. Enter edit mode by pressing `Enter` or using the mouse to select on a cell's editor area.



Shortcut keys under command mode

ACTION	SYNAPSE NOTEBOOK SHORTCUTS
Run the current cell and select below	Shift+Enter
Run the current cell and insert below	Alt+Enter
Run current cell	Ctrl+Enter
Select cell above	Up
Select cell below	Down
Select previous cell	K
Select next cell	J
Insert cell above	A
Insert cell below	B
Delete selected cells	Shift+D
Switch to edit mode	Enter

Shortcut keys under edit mode

Using the following keystroke shortcuts, you can more easily navigate and run code in Synapse notebooks when in Edit mode.

ACTION	SYNAPSE NOTEBOOK SHORTCUTS
Move cursor up	Up
Move cursor down	Down
Undo	Ctrl + Z
Redo	Ctrl + Y

ACTION	SYNAPSE NOTEBOOK SHORTCUTS
Comment/Uncomment	Ctrl + /
Delete word before	Ctrl + Backspace
Delete word after	Ctrl + Delete
Go to cell start	Ctrl + Home
Go to cell end	Ctrl + End
Go one word left	Ctrl + Left
Go one word right	Ctrl + Right
Select all	Ctrl + A
Indent	Ctrl +]
Dedent	Ctrl + [
Switch to command mode	Esc

Next steps

- Check out Synapse sample notebooks
- Quickstart: Create an Apache Spark pool in Azure Synapse Analytics using web tools
- What is Apache Spark in Azure Synapse Analytics
- Use .NET for Apache Spark with Azure Synapse Analytics
- .NET for Apache Spark documentation
- Azure Synapse Analytics

Synapse Studio SQL scripts in Azure Synapse Analytics

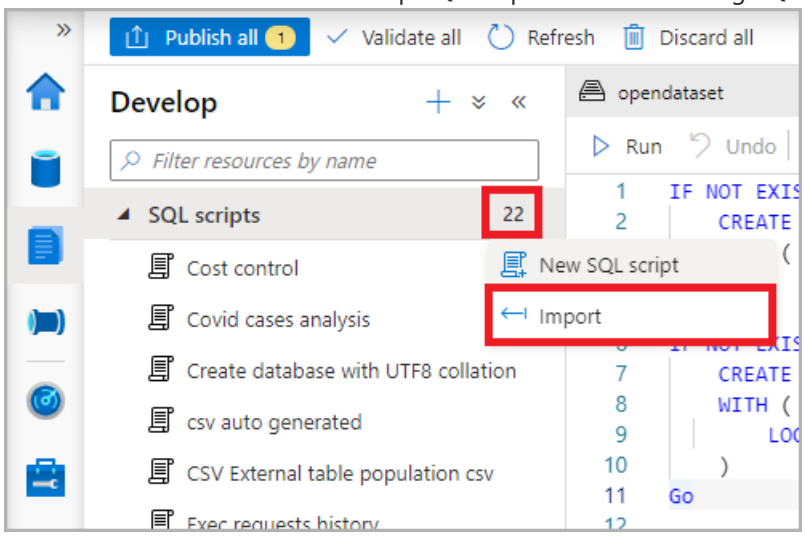
8/19/2022 • 2 minutes to read • Edit Online

Synapse Studio provides a SQL script web interface for you to author SQL queries.

Begin authoring in SQL script

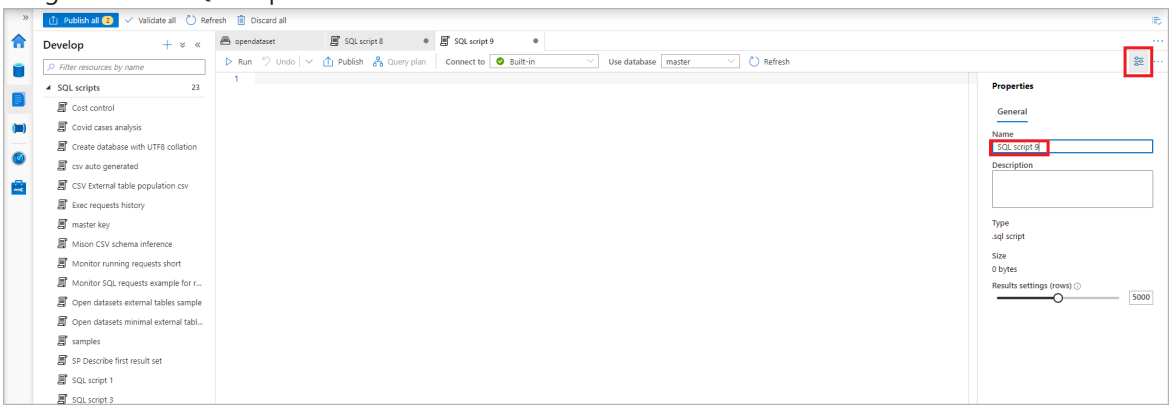
There are several ways to start the authoring experience in SQL script. You can create a new SQL script through one of the following methods.

1. From the Develop menu, select the **"+"** icon and choose **SQL script**.
2. From the **Actions** menu, choose **New SQL script**.
3. Choose **Import** from the **Actions** menu under Develop SQL scripts. Select an existing SQL script from your local storage.

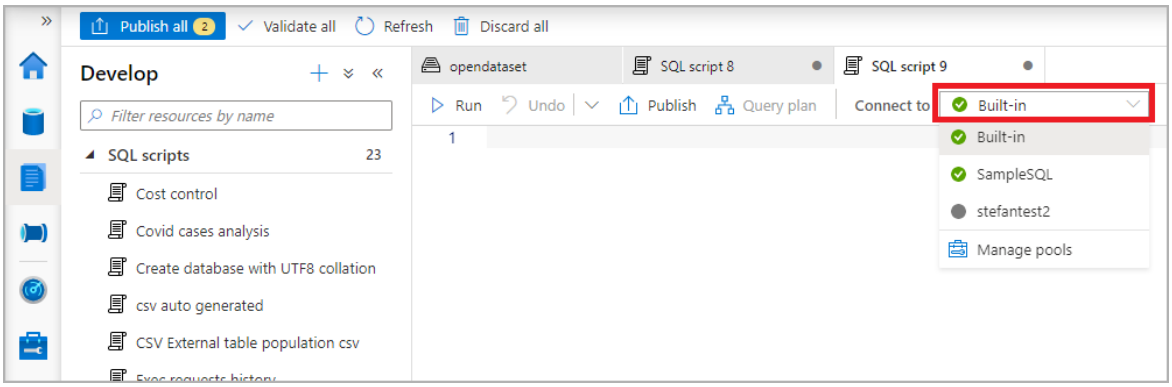


Create your SQL script

1. Choose a name for your SQL script by selecting the **Property** button and replacing the default name assigned to the SQL script.



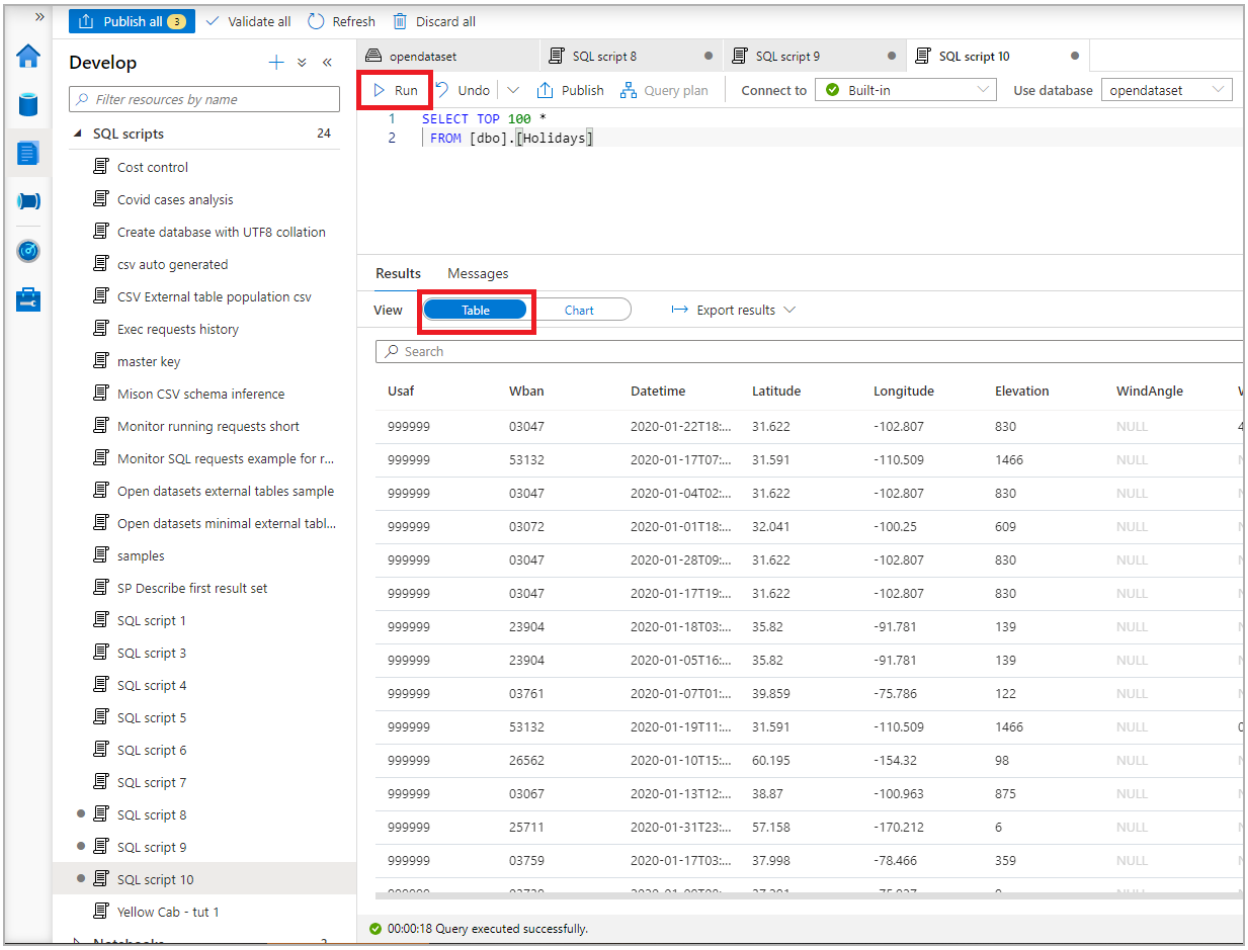
2. Choose the specific dedicated SQL pool or serverless SQL pool from the **Connect to** drop-down menu. Or if necessary, choose the database from **Use database**.



3. Start authoring your SQL script using the intellisense feature.

Run your SQL script

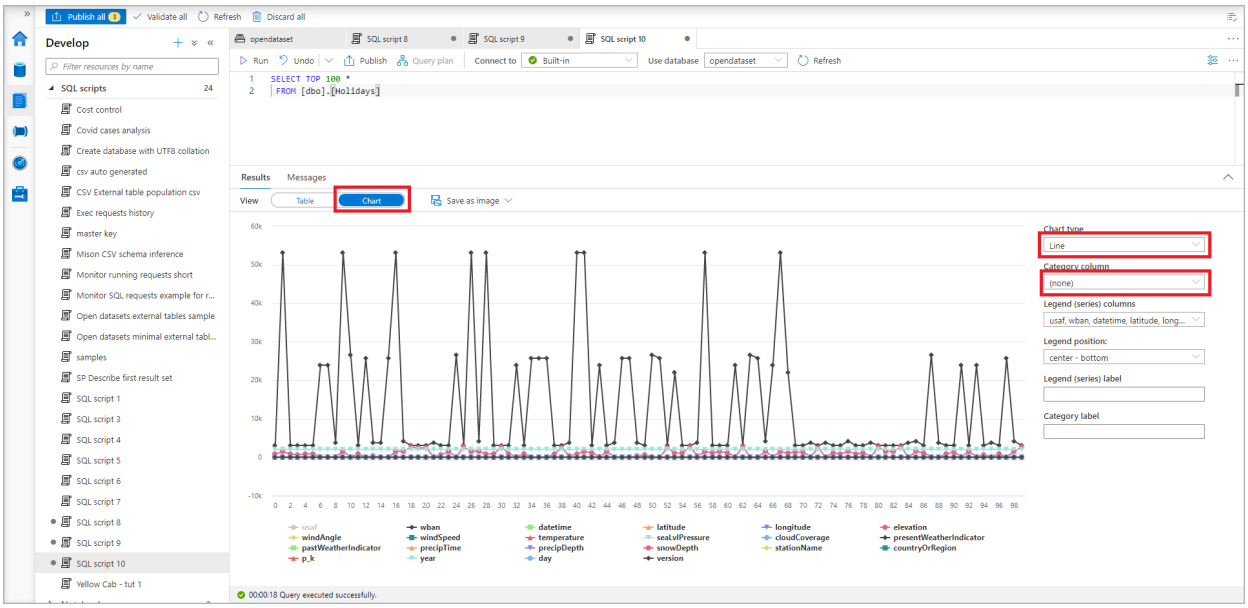
Select the **Run** button to execute your SQL script. The results are displayed by default in a table.



Export your results

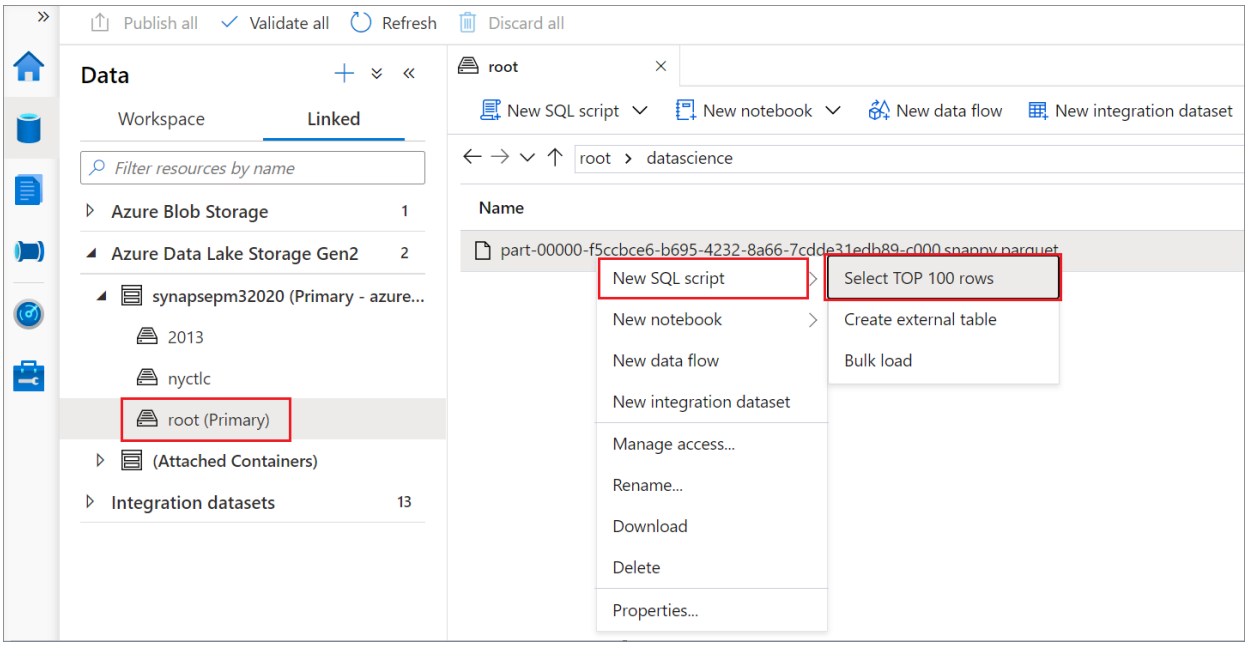
You can export the results to your local storage in different formats (including CSV, Excel, JSON, XML) by selecting "Export results" and choosing the extension.

You can also visualize the SQL script results in a chart by selecting the **Chart** button. Select the "Chart type" and **Category column**. You can export the chart into a picture by selecting **Save as image**.



Explore data from a Parquet file

You can explore Parquet files in a storage account using SQL script to preview the file contents.

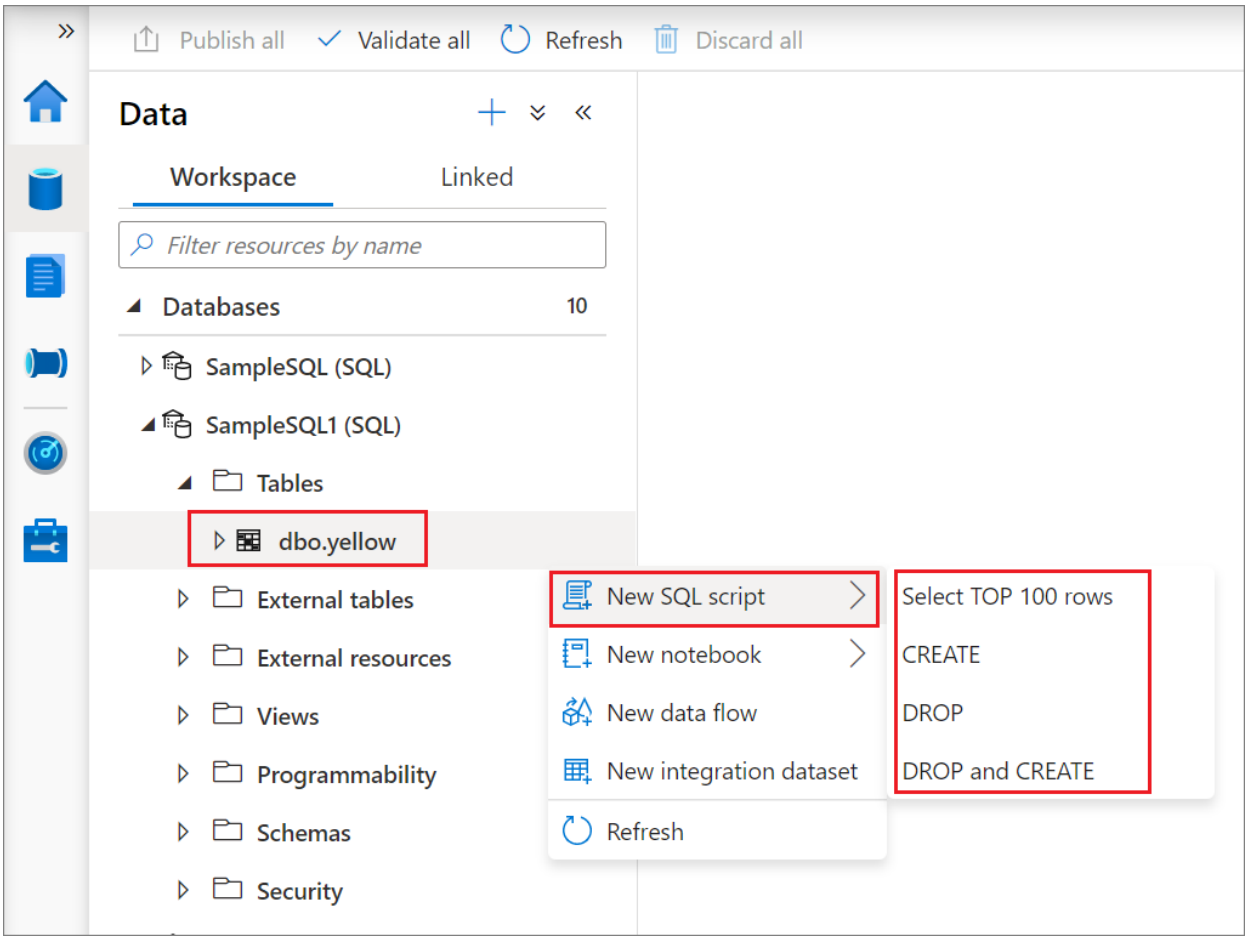


SQL Tables, external tables, views

By selecting the **Actions** menu under data, you can select several actions such as:

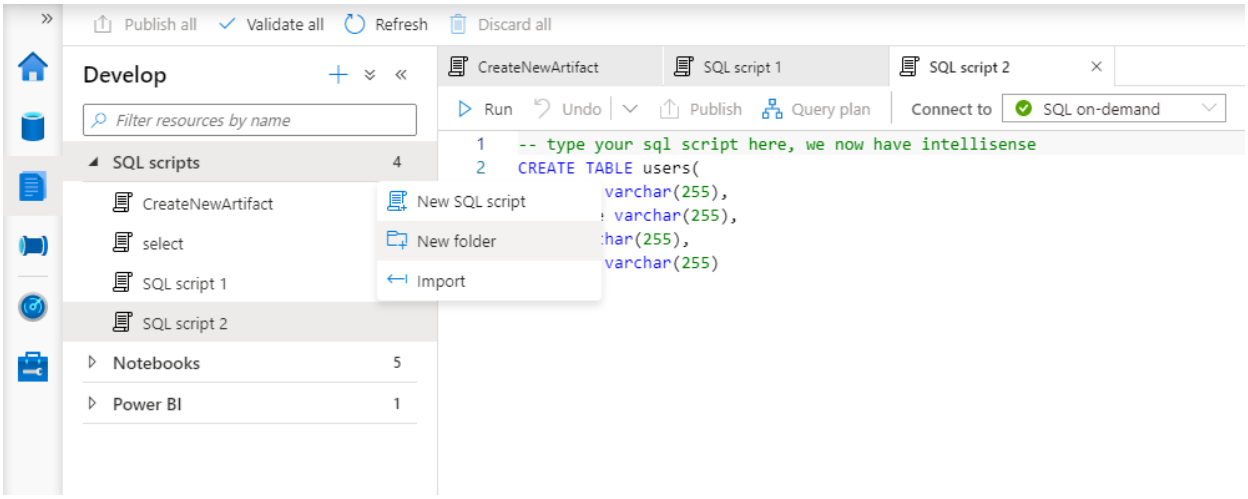
- New SQL script
- Select TOP 1000 rows
- CREATE
- DROP and CREATE

Explore the available gesture by right-clicking the nodes of SQL databases.

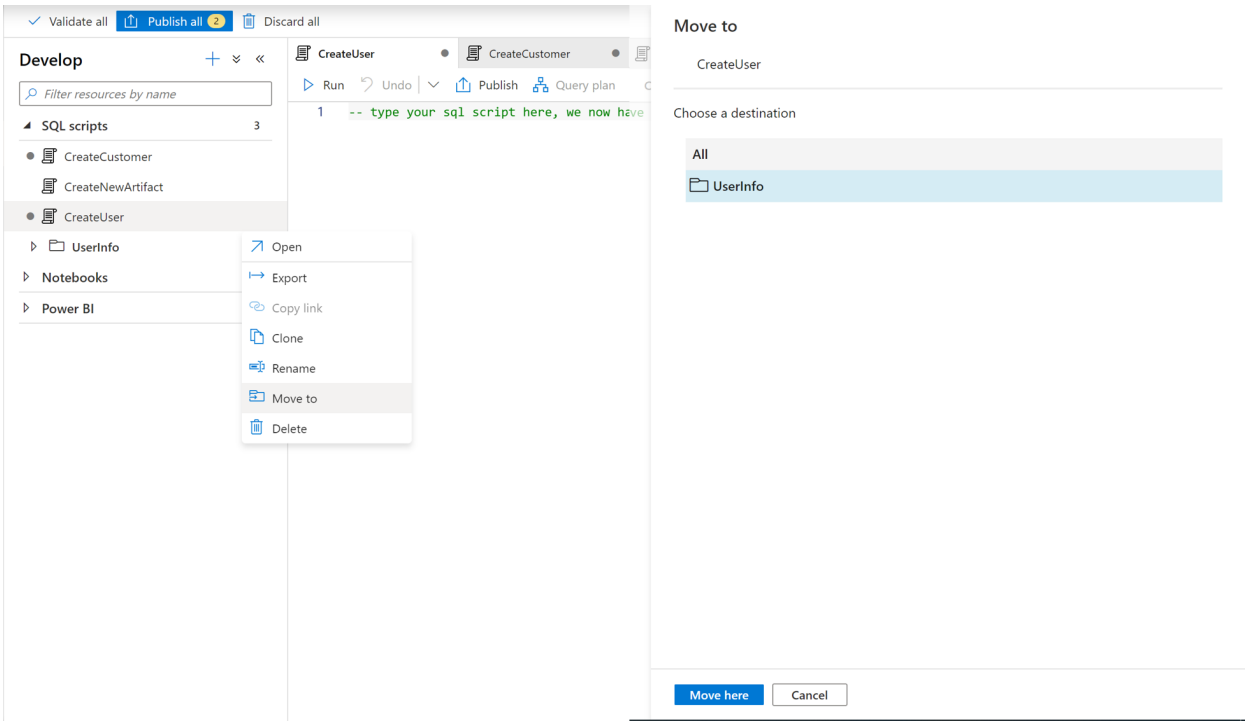


Create folders and move SQL scripts into a folder

From the Actions menu under Develop SQL scripts Choose "New folder" from the "Actions" menu under Develop SQL scripts. And type in the name of the new folder in the pop-up window.



To move a SQL script into a folder, you can select the sql script and choose "Move To" from the Actions menu. Then find the destination folder in the new window and move the sql script into selected folder.You can also quickly drag the sql script and drop it into a folder.



Next steps

For more information about authoring a SQL script, see Azure Synapse Analytics.

Data flows in Azure Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

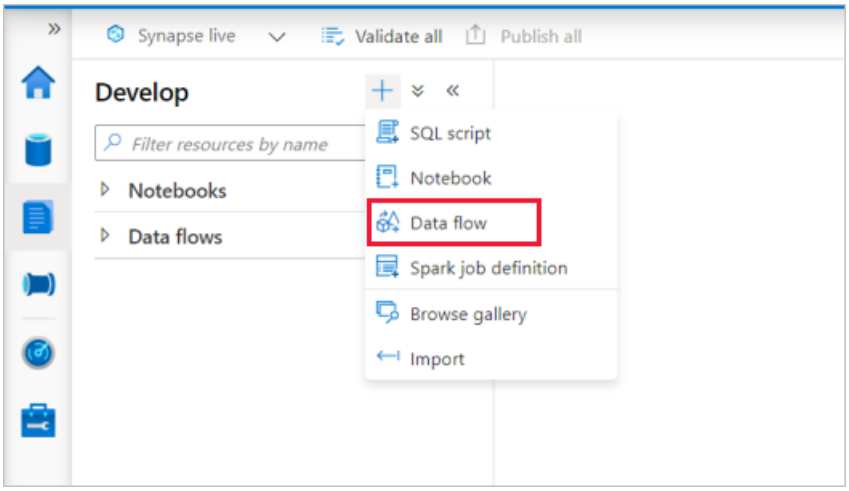
What are data flows?

Data flows are visually designed data transformations in Azure Synapse Analytics. Data flows allow data engineers to develop data transformation logic without writing code. The resulting data flows are executed as activities within Azure Synapse Analytics pipelines that use scaled-out Apache Spark clusters. Data flow activities can be operationalized using existing Azure Synapse Analytics scheduling, control, flow, and monitoring capabilities.

Data flows provide an entirely visual experience with no coding required. Your data flows run on Synapse-managed execution clusters for scaled-out data processing. Azure Synapse Analytics handles all the code translation, path optimization, and execution of your data flow jobs.

Getting started

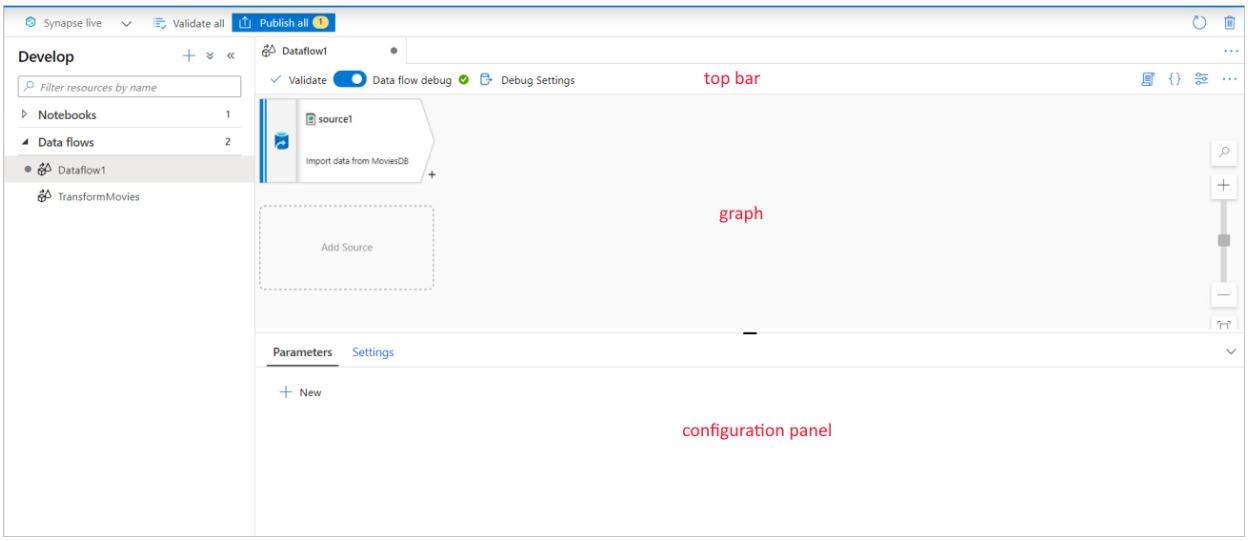
Data flows are created from the Develop pane in Synapse studio. To create a data flow, select the plus sign next to **Develop**, and then select **Data Flow**.



This action takes you to the data flow canvas, where you can create your transformation logic. Select **Add source** to start configuring your source transformation. For more information, see Source transformation.

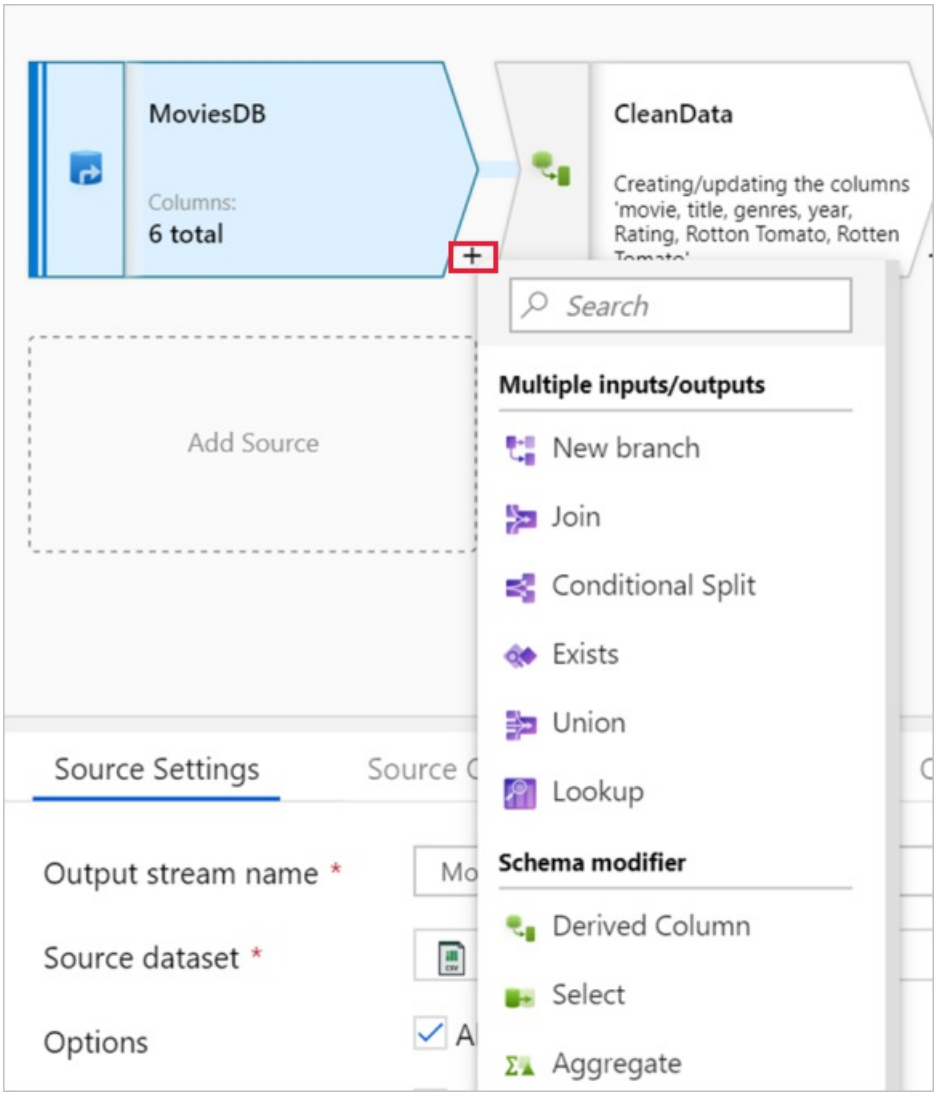
Authoring data flows

Data flow has a unique authoring canvas designed to make building transformation logic easy. The data flow canvas is separated into three parts: the top bar, the graph, and the configuration panel.



Graph

The graph displays the transformation stream. It shows the lineage of source data as it flows into one or more sinks. To add a new source, select **Add source**. To add a new transformation, select the plus sign on the lower right of an existing transformation. Learn more on how to manage the data flow graph.



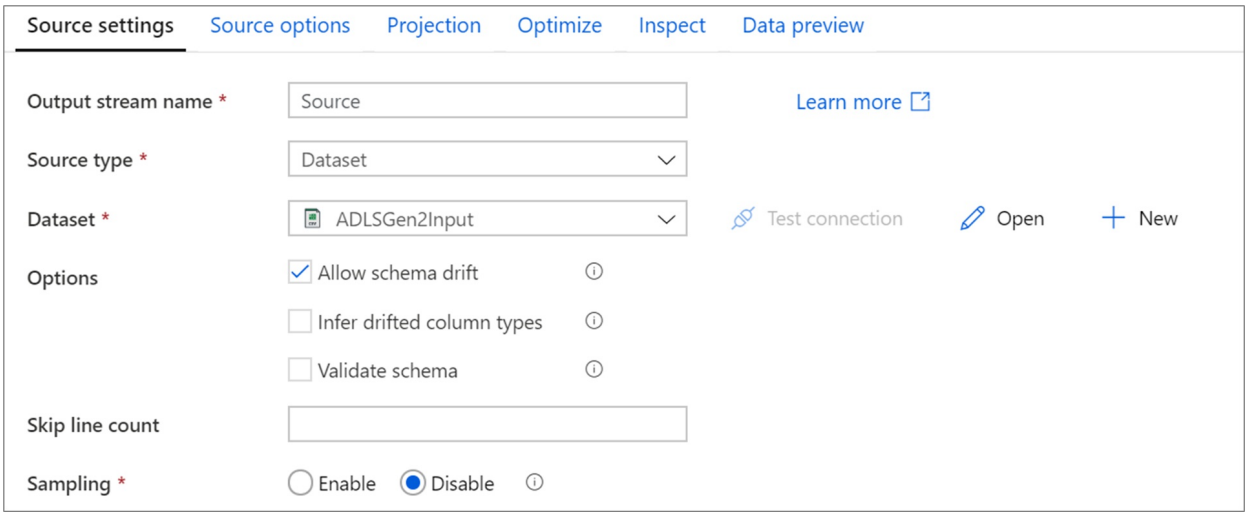
Configuration panel

The configuration panel shows the settings specific to the currently selected transformation. If no transformation is selected, it shows the data flow. In the overall data flow configuration, you can add parameters via the **Parameters** tab. For more information, see Data flow parameters.

Each transformation contains at least four configuration tabs.

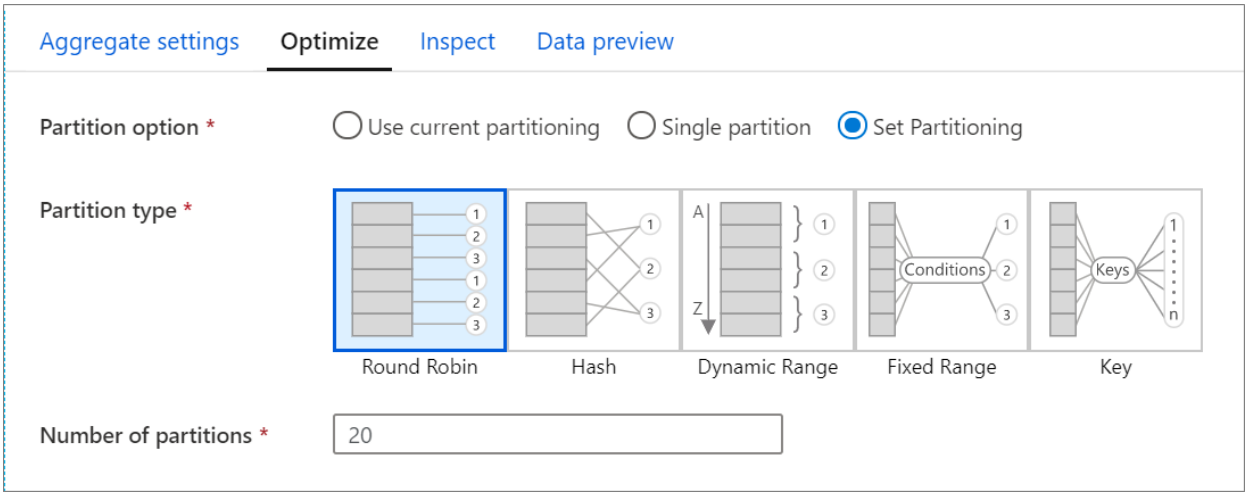
Transformation settings

The first tab in each transformation's configuration pane contains the settings specific to that transformation. For more information, see that transformation's documentation page.



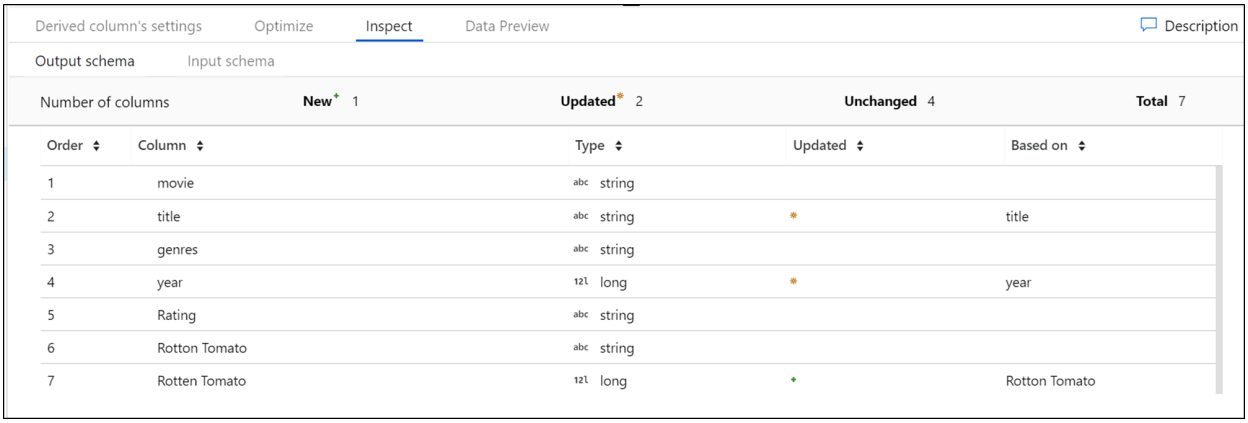
Optimize

The **Optimize** tab contains settings to configure partitioning schemes. To learn more about how to optimize your data flows, see the mapping data flow performance guide.



Inspect

The **Inspect** tab provides a view into the metadata of the data stream that you're transforming. You can see column counts, the columns changed, the columns added, data types, the column order, and column references. **Inspect** is a read-only view of your metadata. You don't need to have debug mode enabled to see metadata in the **Inspect** pane.



As you change the shape of your data through transformations, you'll see the metadata changes flow in the **Inspect** pane. If there isn't a defined schema in your source transformation, then metadata won't be visible in the **Inspect** pane. Lack of metadata is common in schema drift scenarios.

Data preview

If debug mode is on, the **Data Preview** tab gives you an interactive snapshot of the data at each transform. For more information, see Data preview in debug mode.

Top bar

The top bar contains actions that affect the whole data flow, like validation and debug settings. You can view the underlying JSON code and data flow script of your transformation logic as well.

Available transformations

View the mapping data flow transformation overview to get a list of available transformations.

Data flow activity

Data flows are operationalized within Azure Synapse Analytics pipelines using the data flow activity. All a user has to do is specify which integration runtime to use and pass in parameter values. For more information, learn about the Azure integration runtime.

Debug mode

Debug mode allows you to interactively see the results of each transformation step while you build and debug your data flows. The debug session can be used both in when building your data flow logic and running pipeline debug runs with data flow activities. To learn more, see the debug mode documentation.

Monitoring data flows

Data flow integrates with existing Azure Synapse Analytics monitoring capabilities. To learn how to understand data flow monitoring output, see monitoring mapping data flows.

The Azure Synapse Analytics team has created a performance tuning guide to help you optimize the execution time of your data flows after building your business logic.

Next steps

- Learn how to create a source transformation.
- Learn how to build your data flows in debug mode.

Use maintenance schedules to manage service updates and maintenance

8/19/2022 • 5 minutes to read • Edit Online

The maintenance schedule feature integrates the Service Health Planned Maintenance Notifications, Resource Health Check Monitor, and maintenance scheduling service for Synapse SQL pool (data warehouse) within Azure Synapse Analytics.

You should use maintenance scheduling to choose a time window when it's convenient to receive new features, upgrades, and patches. You will need to choose a primary and a secondary maintenance window within a seven-day period, each window must be within separate day ranges.

For example, you can schedule a primary window of Saturday 22:00 to Sunday 01:00, and then schedule a secondary window of Wednesday 19:00 to 22:00. If maintenance can't be performed during your primary maintenance window, it will try the maintenance again during your secondary maintenance window. Service maintenance could on occasion occur during both the primary and secondary windows. To ensure rapid completion of all maintenance operations, DW400c and lower data warehouse tiers could complete maintenance outside of a designated maintenance window.

All newly created data warehouse instances will have a system-defined maintenance schedule applied during deployment. The schedule can be edited as soon as deployment is complete.

Although a maintenance window can be between three and eight hours this does not mean the data warehouse will be offline for the duration. Maintenance can occur at any time within that window and you should expect a single disconnect during that period lasting ~5 -6 mins as the service deploys new code to your data warehouse. DW400c and lower may experience multiple brief losses in connectivity at various times during the maintenance window. When maintenance starts, all active sessions will be canceled, and non-committed transactions will be rolled back. To minimize instance downtime, make sure that your data warehouse has no long-running transactions before your chosen maintenance period.

All maintenance operations should finish within the specified maintenance windows unless we are required to deploy a time sensitive update. If your data warehouse is paused during a scheduled maintenance, it will be updated during the resume operation. You'll be notified immediately after your data warehouse maintenance is completed.

Alerts and monitoring

Integration with Service Health notifications and the Resource Health Check Monitor allows customers to stay informed of impending maintenance activity. This automation takes advantage of Azure Monitor. You can decide how you want to be notified of impending maintenance events. Also, you can choose which automated flows will help you manage downtime and minimize operational impact.

A 24-hour advance notification precedes all maintenance events that aren't for the DWC400c and lower tiers.

> **NOTE**
> In the event we are required to deploy a time critical update, advanced notification times may be significantly reduced. This could occur outside an identified maintenance window due to the critical nature of the update.

If you received advance notification that maintenance will take place, but maintenance can't be performed

during the time period in the notification, you'll receive a cancellation notification. Maintenance will then resume during the next scheduled maintenance period.

All active maintenance events appear in the **Service Health - Planned Maintenance** section. The Service Health history includes a full record of past events. You can monitor maintenance via the Azure Service Health check portal dashboard during an active event.

Maintenance schedule availability

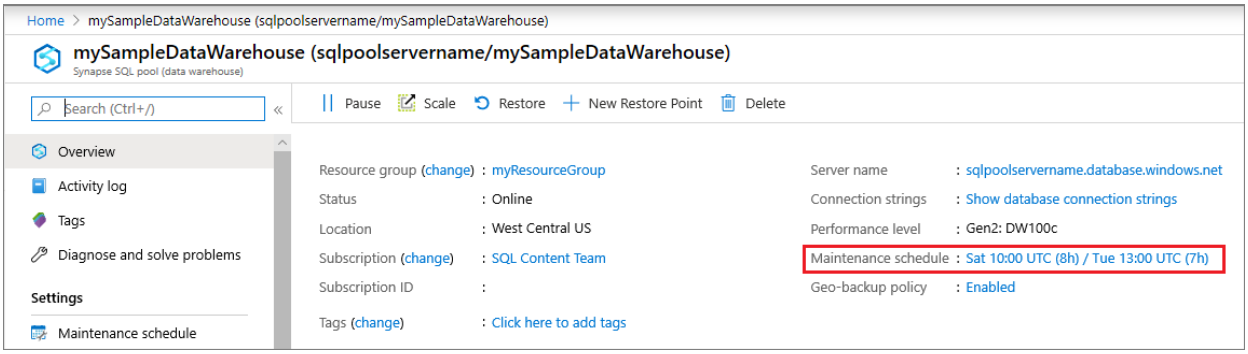
Even if maintenance scheduling isn't available in your selected region, you can view and edit your maintenance schedule at any time. When maintenance scheduling becomes available in your region, the identified schedule will immediately become active on your Synapse SQL pool.

View a maintenance schedule

By default, all newly created data warehouse instances have an eight-hour primary and secondary maintenance window applied during deployment. As indicated above, you can change the windows as soon deployment is complete. No maintenance will take place outside the specified maintenance windows without prior notification.

To view the maintenance schedule that has been applied to your Synapse SQL pool, complete the following steps:

1. Sign in to the Azure portal.
2. Select the Synapse SQL pool that you want to view.
3. The selected Synapse SQL pool opens on the overview blade. The maintenance schedule that's applied to the data warehouse appears below **Maintenance schedule**.



Change a maintenance schedule

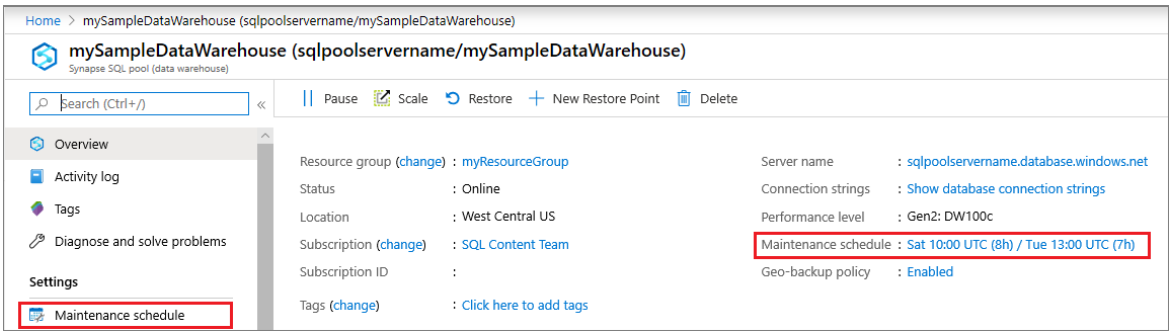
A maintenance schedule can be updated or changed at any time. If the selected instance is going through an active maintenance cycle, the settings will be saved. They'll become active during the next identified maintenance period. Learn more about monitoring your data warehouse during an active maintenance event.

Identifying the primary and secondary windows

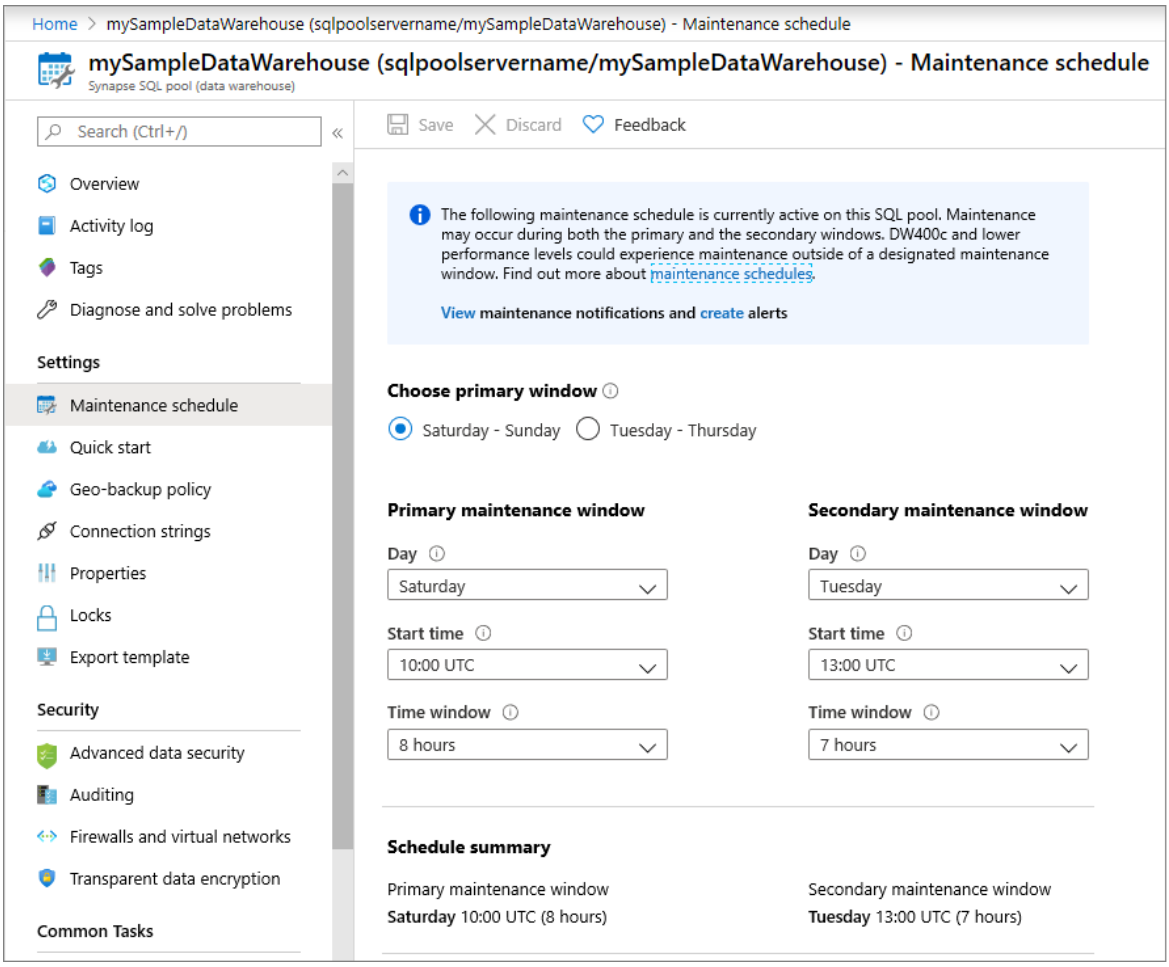
The primary and secondary windows must have separate day ranges. An example is a primary window of Tuesday–Thursday and a secondary of window of Saturday–Sunday.

To change the maintenance schedule for your Synapse SQL pool, complete the following steps:

1. Sign in to the Azure portal.
2. Select the Synapse SQL pool that you want to update. The page opens on the overview blade. Open the page for maintenance schedule settings by selecting the **Maintenance Schedule summary** link on the overview blade. Or, select the **Maintenance Schedule** option on the left-side resource menu.



3. Identify the preferred day range for your primary maintenance window by using the options at the top of the page. This selection determines if your primary window will occur on a weekday or over the weekend. Your selection will update the drop-down values. During preview, some regions might not yet support the full set of available **Day** options.



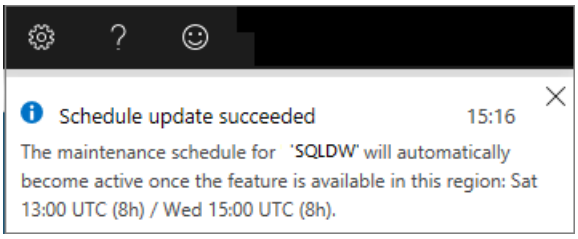
4. Choose your preferred primary and secondary maintenance windows by using the drop-down list boxes:

   - **Day**: Preferred day to perform maintenance during the selected window.
   - **Start time**: Preferred start time for the maintenance window.
   - **Time window**: Preferred duration of your time window.

   The **Schedule summary** area at the bottom of the blade is updated based on the values that you selected.

5. Select **Save**. A message appears, confirming that your new schedule is now active.

   If you're saving a schedule in a region that doesn't support maintenance scheduling, the following message appears. Your settings are saved and become active when the feature becomes available in your selected region.



Next steps

- Learn more about creating, viewing, and managing alerts by using Azure Monitor.
- Learn more about webhook actions for log alert rules.
- Learn more Creating and managing Action Groups.
- Learn more about Azure Service Health.

Source control in Synapse Studio

8/19/2022 • 11 minutes to read • Edit Online

By default, Synapse Studio authors directly against the Synapse service. If you have a need for collaboration using Git for source control, Synapse Studio allows you to associate your workspace with a Git repository, Azure DevOps, or GitHub.

This article will outline how to configure and work in a Synapse workspace with git repository enabled. And we also highlight some best practices and a troubleshooting guide.

> **NOTE**
> To use GitHub in Azure Gov and Azure China, you can bring your own GitHub OAuth application in Synapse Studio for git integration. The configure experience is same with ADF. You can refer to the announcement blog.

Prerequisites

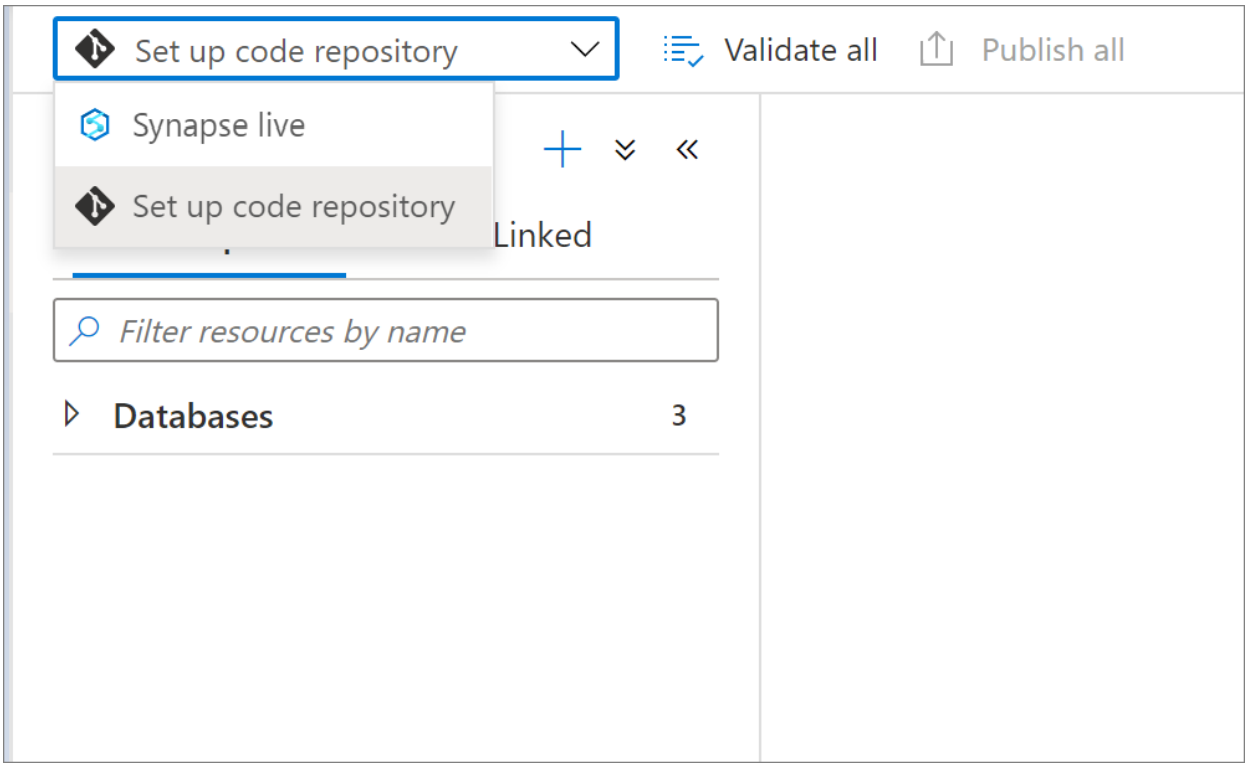
Users must have the Azure Contributor (Azure RBAC) or higher role on the Synapse workspace to configure, edit settings and disconnect a Git repository with Synapse.

Configure Git repository in Synapse Studio

After launching your Synapse Studio, you can configure a git repository in your workspace. A Synapse Studio workspace can be associated with only one git repository at a time.

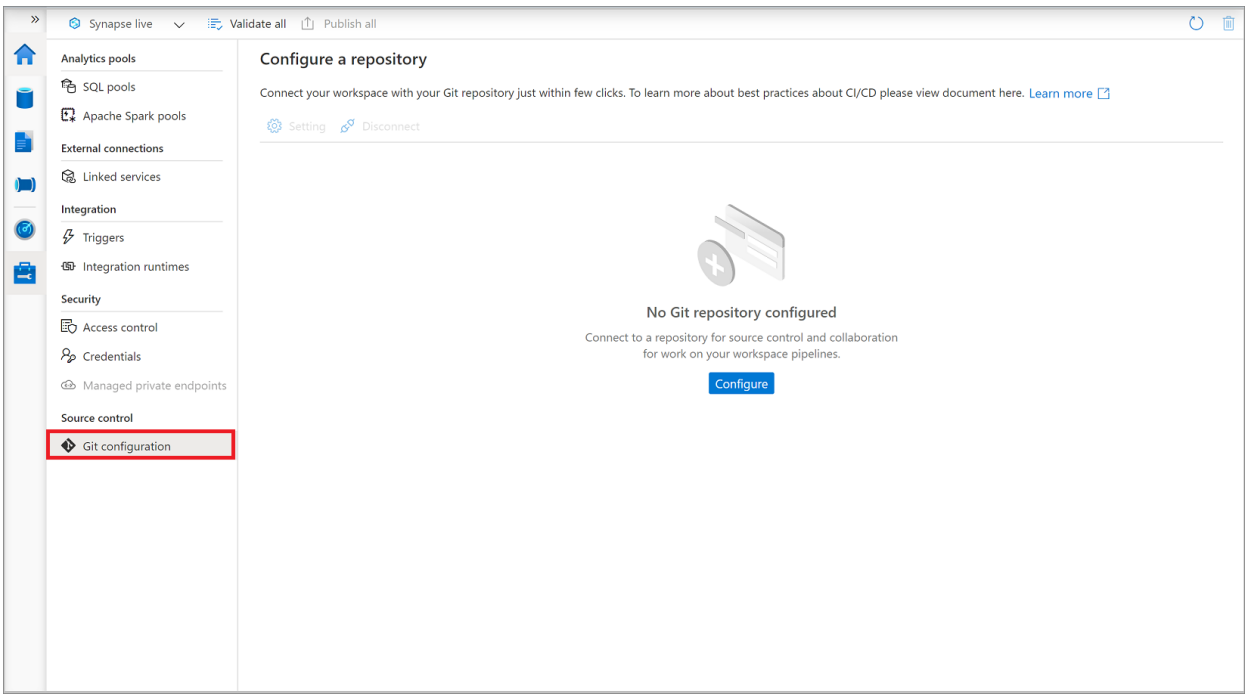
Configuration method 1: global bar

In the Synapse Studio global bar, select the **Synapse Live** drop-down menu, and then select **Set up code repository**.



Configuration method 2: Manage hub

Go to the Manage hub of Synapse Studio. Select **Git configuration** in the **Source control** section. If you have no repository connected, click **Configure**.



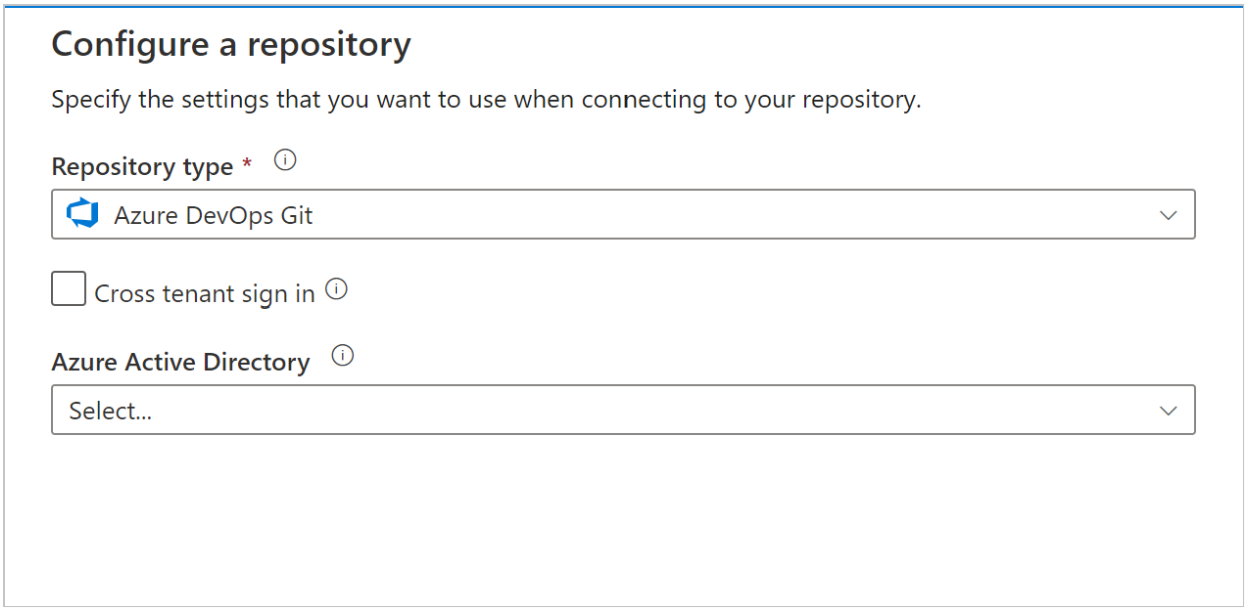
You can connect either Azure DevOps or GitHub git repository in your workspace.

Connect with Azure DevOps Git

You can associate a Synapse workspace with an Azure DevOps Repository for source control, collaboration, versioning, and so on. If you don't have an Azure DevOps repository, follow these instructions to create your repository resources.

Azure DevOps Git repository settings

When connecting to your git repository, first select your repository type as Azure DevOps git, and then select one Azure AD tenant from the dropdown list, and click **Continue**.



The configuration pane shows the following Azure DevOps git settings:

SETTING	DESCRIPTION	VALUE
Repository Type	The type of the Azure Repos code repository.	Azure DevOps Git or GitHub
Cross tenant sign in	Checkbox to sign in with cross tenant account.	unselected (default)
Azure Active Directory	Your Azure AD tenant name.	`<your tenant name>`
Azure DevOps account	Your Azure Repos organization name. You can locate your Azure Repos organization name at `https://{organization name}.visualstudio.com`. You can sign in to your Azure Repos organization to access your Visual Studio profile and see your repositories and projects.	`<your organization name>`
ProjectName	Your Azure Repos project name. You can locate your Azure Repos project name at `https://{organization name}.visualstudio.com/{project name}`.	`<your Azure Repos project name>`
RepositoryName	Your Azure Repos code repository name. Azure Repos projects contain Git repositories to manage your source code as your project grows. You can create a new repository or use an existing repository that's already in your project.	`<your Azure Repos code repository name>`
Collaboration branch	Your Azure Repos collaboration branch that is used for publishing. By default, its `master`. Change this setting in case you want to publish resources from another branch. You can select existing branches or create new	`<your collaboration branch name>`
Root folder	Your root folder in your Azure Repos collaboration branch.	`<your root folder name>`
Import existing resources to repository	Specifies whether to import existing resources from the Synapse Studio into an Azure Repos Git repository. Check the box to import your workspace resources (except pools) into the associated Git repository in JSON format. This action exports each resource individually. When this box isn't checked, the existing resources aren't imported.	Checked (default)
Import resource into this branch	Select which branch the resources (sql script, notebook, spark job definition, dataset, dataflow etc.) are imported to.	

Your can also use repository link to quickly point to the git repository you want to connect with.

Use a different Azure Active Directory tenant

The Azure Repos Git repo can be in a different Azure Active Directory tenant. To specify a different Azure AD tenant, you have to have administrator permissions for the Azure subscription that you're using. For more info, see change subscription administrator

> **IMPORTANT**
> To connect to another Azure Active Directory, the user logged in must be a part of that active directory.

Use your personal Microsoft account

To use a personal Microsoft account for Git integration, you can link your personal Azure Repo to your organization's Active Directory.

1. Add your personal Microsoft account to your organization's Active Directory as a guest. For more info, see Add Azure Active Directory B2B collaboration users in the Azure portal.
2. Log in to the Azure portal with your personal Microsoft account. Then switch to your organization's Active Directory.
3. Go to the Azure DevOps section, where you now see your personal repo. Select the repo and connect with Active Directory.

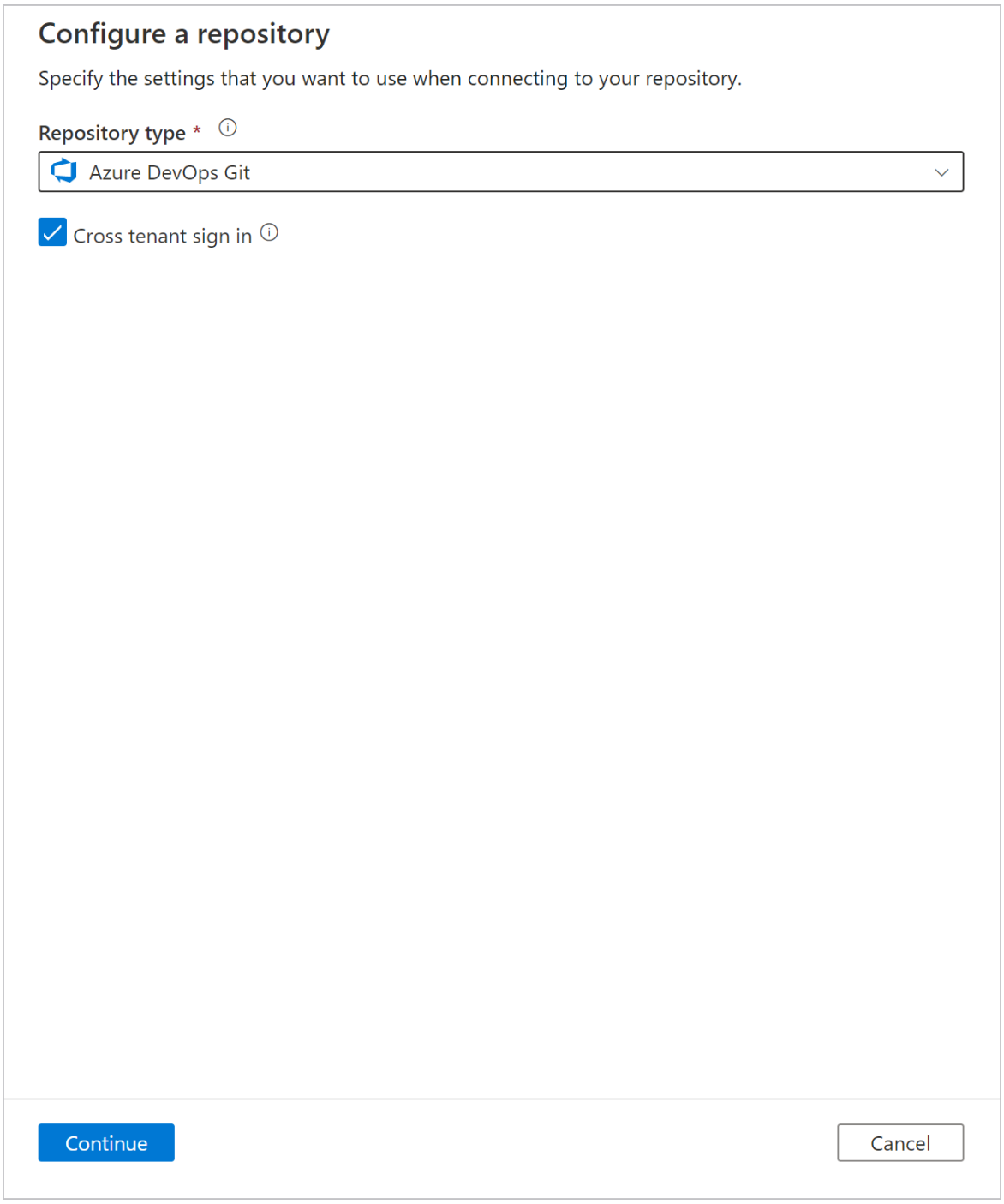
After these configuration steps, your personal repo is available when you set up Git integration in the Synapse Studio.

For more info about connecting Azure Repos to your organization's Active Directory, see Connect your organization to Azure Active Directory.

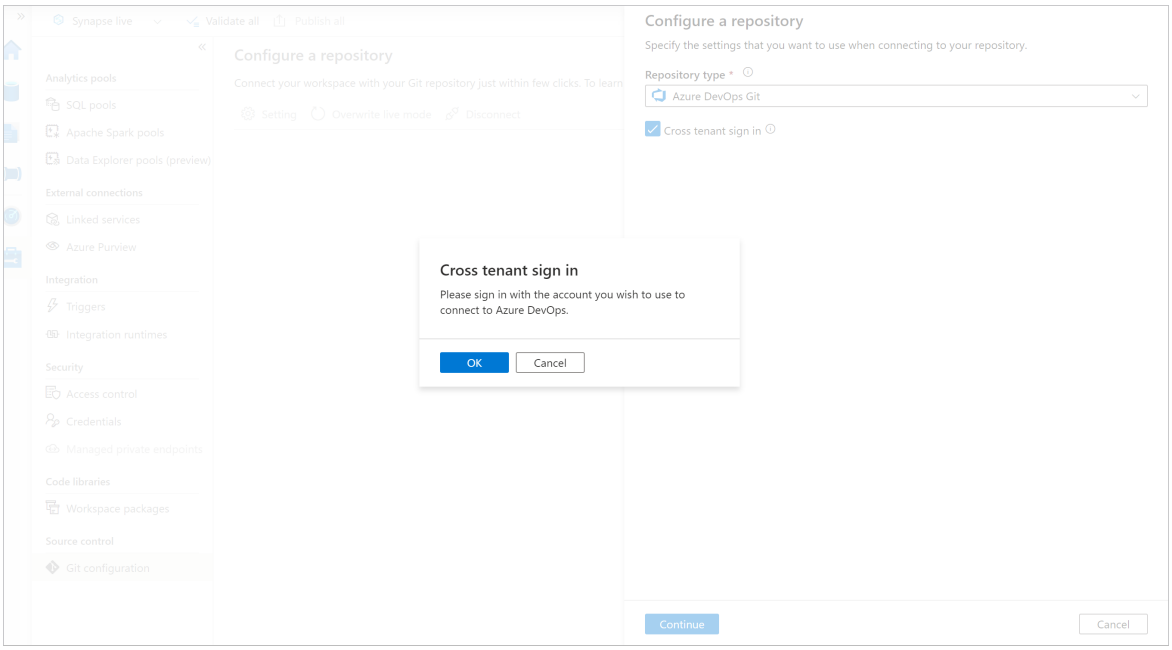
Use a cross tenant Azure DevOps account

When your Azure DevOps is not in the same tenant as the Synapse workspace, you can configure the workspace with your cross tenant Azure DevOps account with the guide below.

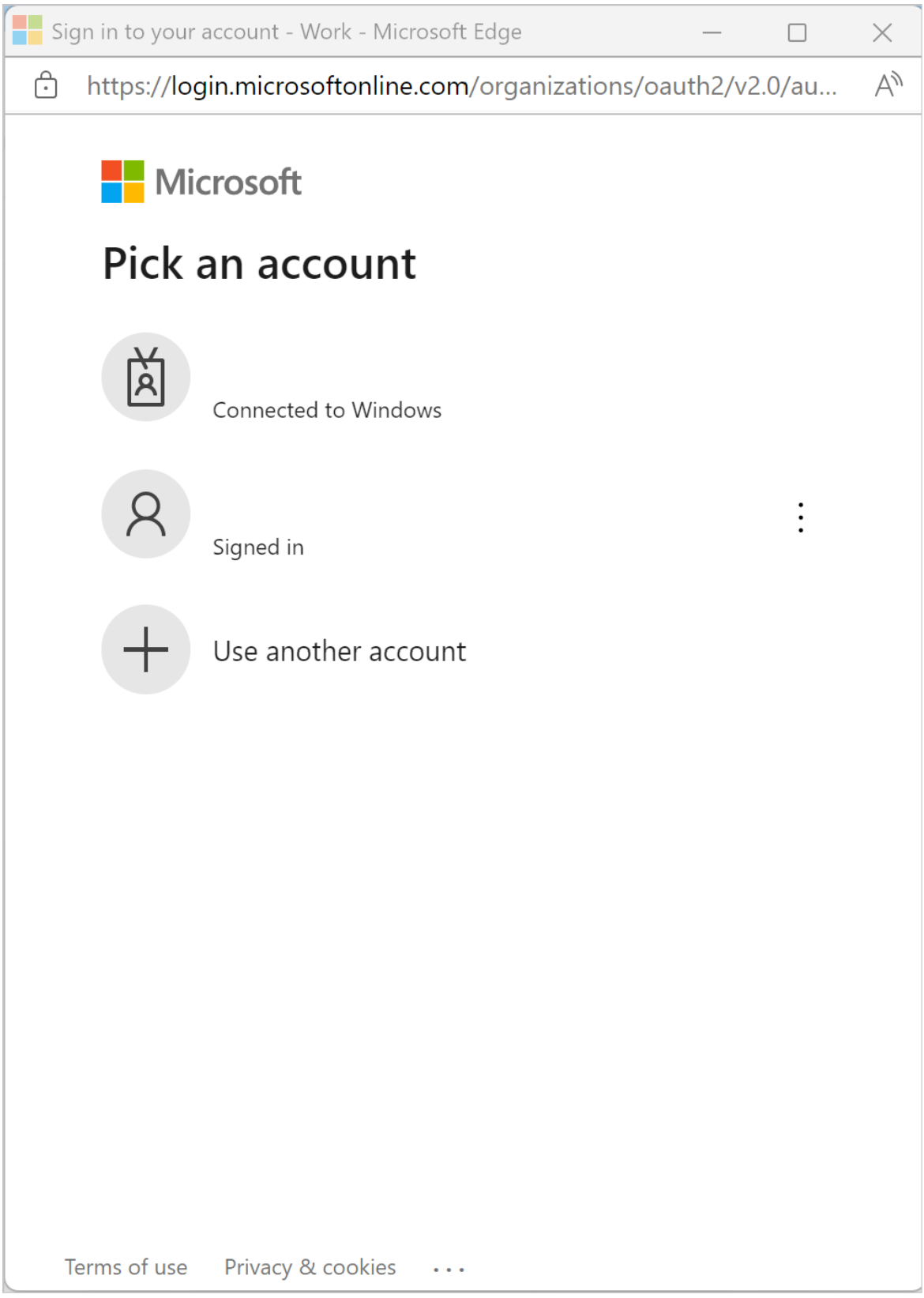
1. Select the **Cross tenant sign in** option and click **Continue**



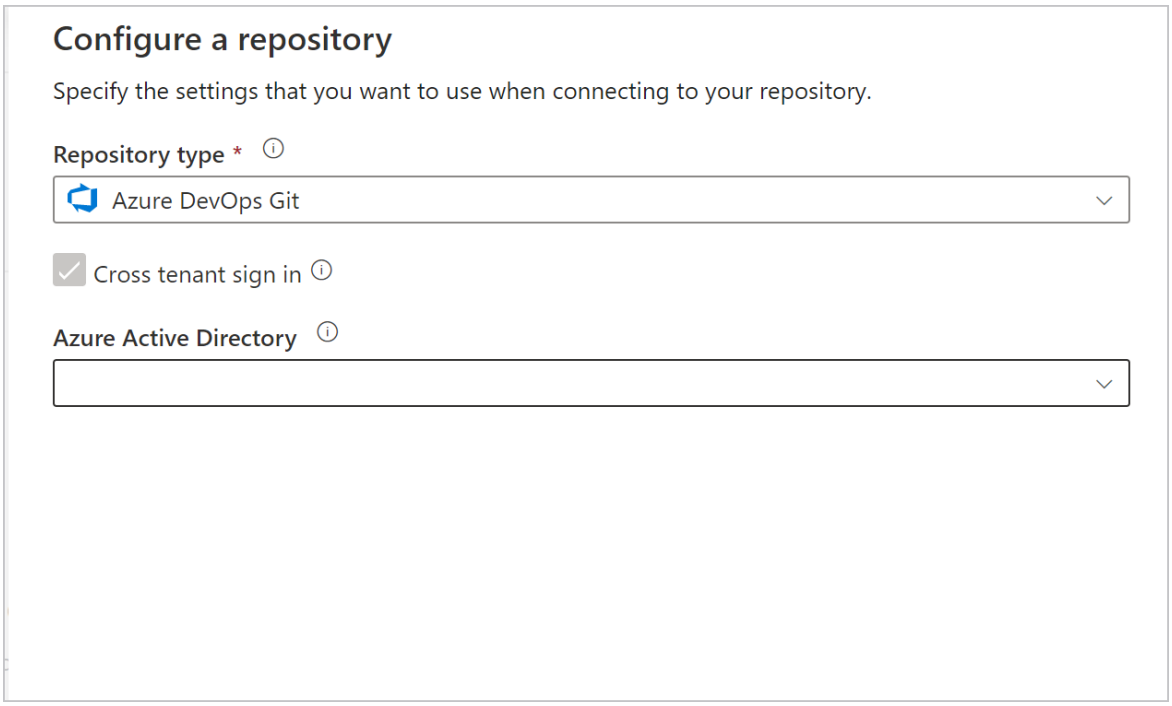
2. Select **OK** in the dialog box.



3. click **Use another account** and login with your Azure DevOps account.



4. After signing in, choose the directory and repository and configure it accordingly.



NOTE

To login the workspace, you need to use the first sign-in to log into to your Synapse workspace user account. Your cross tenant Azure DevOps account is only used for signing into and getting access to the Azure DevOps repo associated with this Synapse workspace.

Connect with GitHub

You can associate a workspace with a GitHub repository for source control, collaboration, versioning. If you don't have a GitHub account or repository, follow these instructions to create your resources.

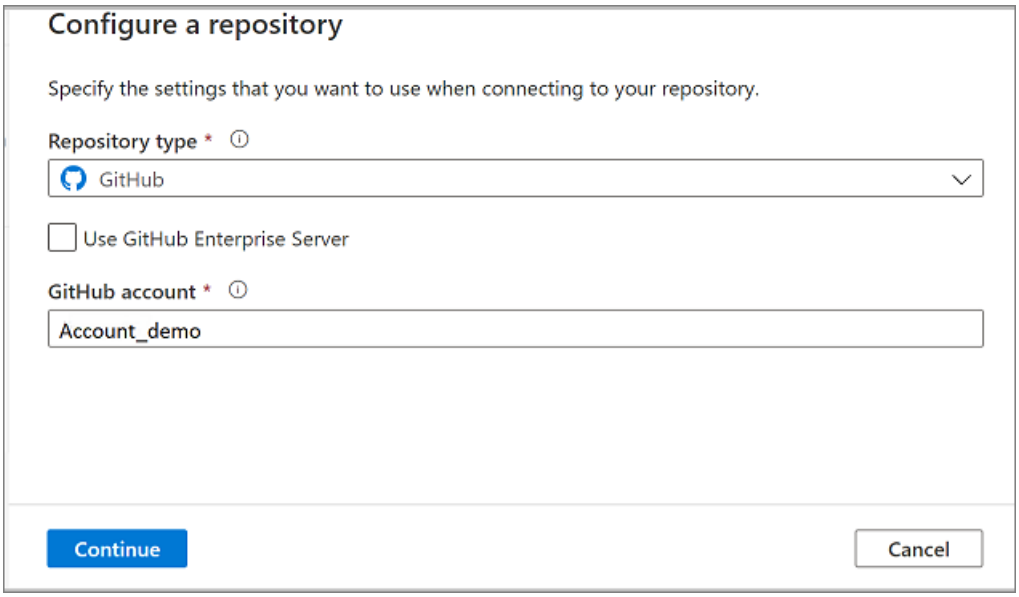
The GitHub integration with Synapse Studio supports both public GitHub (that is, https://github.com) and GitHub Enterprise. You can use both public and private GitHub repositories as long you have read and write permission to the repository in GitHub.

GitHub settings

When connecting to your git repository, first select your repository type as GitHub, and then provide your GitHub account, your GitHub Enterprise Server URL if you use GitHub Enterprise Server, or your GitHub Enterprise organization name if you use GitHub Enterprise Cloud. Select **Continue**.

NOTE

If you're using GitHub Enterprise Cloud, leave the **Use GitHub Enterprise Server** checkbox cleared.



The configuration pane shows the following GitHub repository settings:

SETTING	DESCRIPTION	VALUE
Repository Type	The type of the Azure Repos code repository.	GitHub
Use GitHub Enterprise	Checkbox to select GitHub Enterprise	unselected (default)
GitHub Enterprise URL	The GitHub Enterprise root URL (must be HTTPS for local GitHub Enterprise server). For example: `https://github.mydomain.com`. Required only if **Use GitHub Enterprise** is selected	`<your GitHub enterprise url>`
GitHub account	Your GitHub account name. This name can be found from https://github.com/{account name}/{repository name}. Navigating to this page prompts you to enter GitHub OAuth credentials to your GitHub account.	`<your GitHub account name>`
Repository Name	Your GitHub code repository name. GitHub accounts contain Git repositories to manage your source code. You can create a new repository or use an existing repository that's already in your account.	`<your repository name>`
Collaboration branch	Your GitHub collaboration branch that is used for publishing. By default, its master. Change this setting in case you want to publish resources from another branch.	`<your collaboration branch>`
Root folder	Your root folder in your GitHub collaboration branch.	`<your root folder name>`

SETTING	DESCRIPTION	VALUE
Import existing resources to repository	Specifies whether to import existing resources from the Synapse Studio into a Git repository. Check the box to import your workspace resources (except pools) into the associated Git repository in JSON format. This action exports each resource individually. When this box isn't checked, the existing resources aren't imported.	Selected (default)
Import resource into this branch	Select which branch the resources (sql script, notebook, spark job definition, dataset, dataflow etc.) is imported.	

GitHub organizations

Connecting to a GitHub organization requires the organization to grant permission to Synapse Studio. A user with ADMIN permissions on the organization must perform the below steps.

Connecting to GitHub for the first time

If you're connecting to GitHub from Synapse Studio for the first time, follow these steps to connect to a GitHub organization.

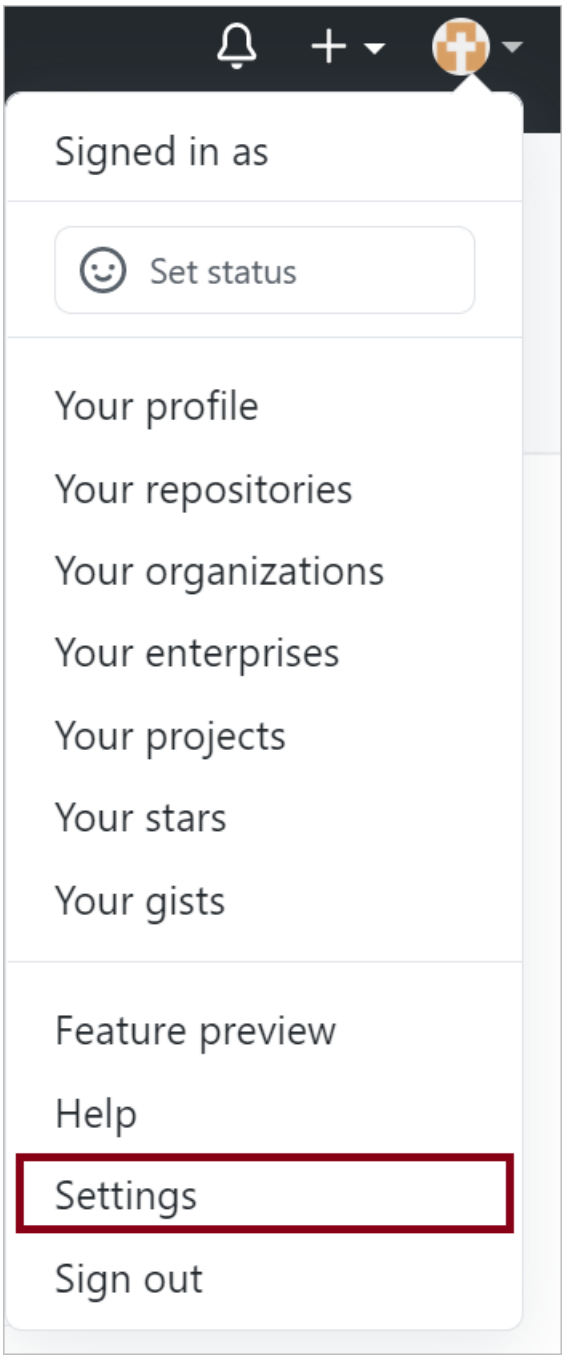
1. In the Git configuration pane, enter the organization name in the *GitHub Account* field. A prompt to login into GitHub will appear.
2. Login using your user credentials.
3. You'll be asked to authorize Synapse as an application called *Azure Synapse*. On this screen, you will see an option to grant permission for Synapse to access the organization. If you don't see the option to grant permission, ask an admin to manually grant the permission through GitHub.

Once you follow these steps, your workspace will be able to connect to both public and private repositories within your organization. If you are unable to connect, try clearing the browser cache and retrying.

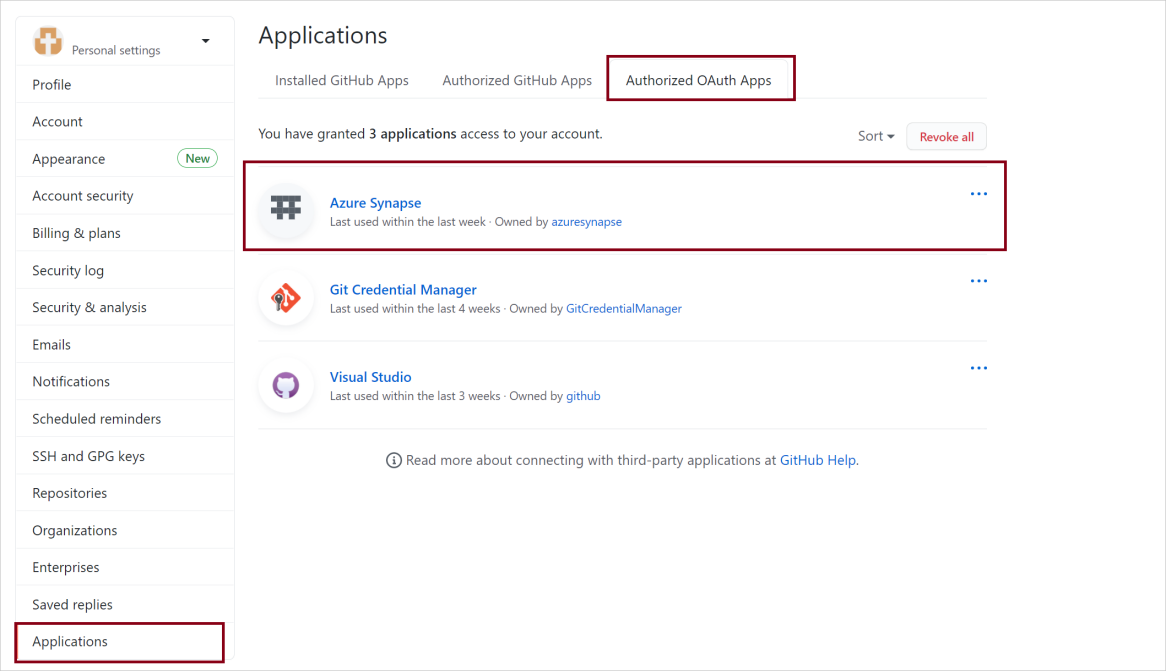
Already connected to GitHub using a personal account

If you have already connected to GitHub and only granted permission to access a personal account, follow the below steps to grant permissions to an organization.

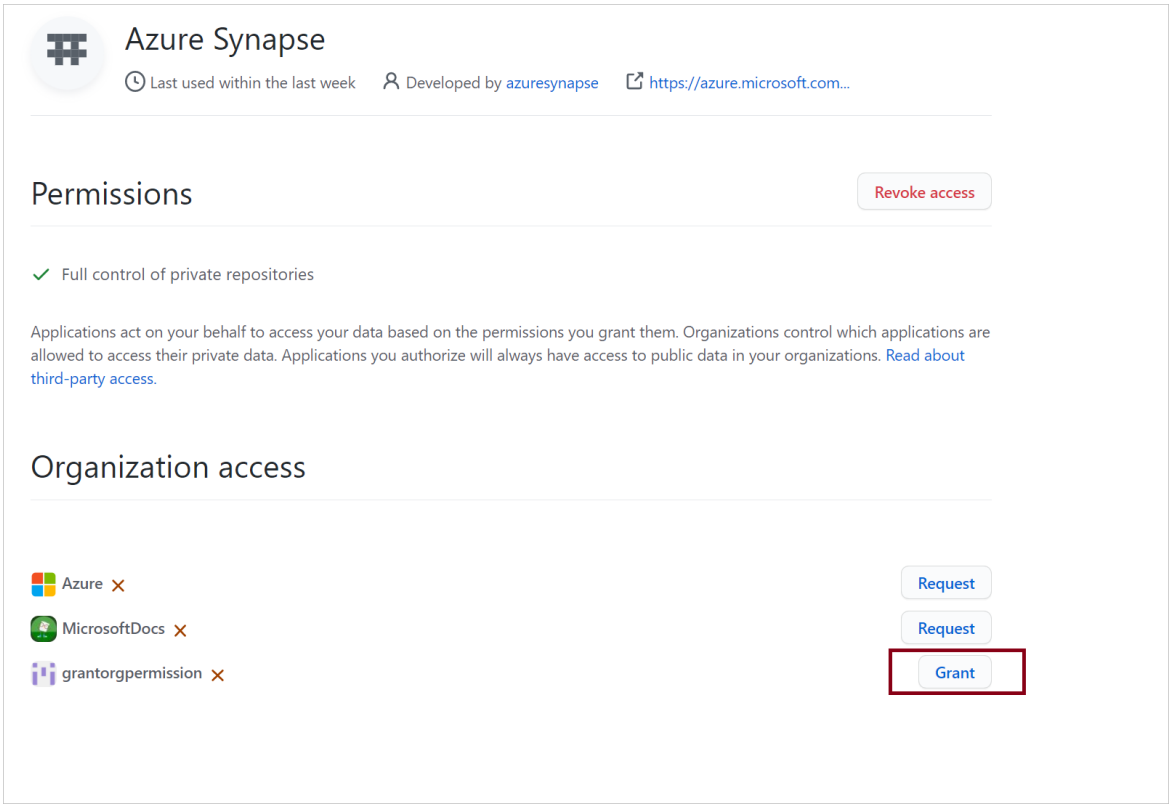
1. Go to GitHub and open **Settings**.



2. Select **Applications**. In the **Authorized OAuth apps** tab, you should see *Azure Synapse*.



3. Select the *Azure Synapse* and grant the access to your organization.



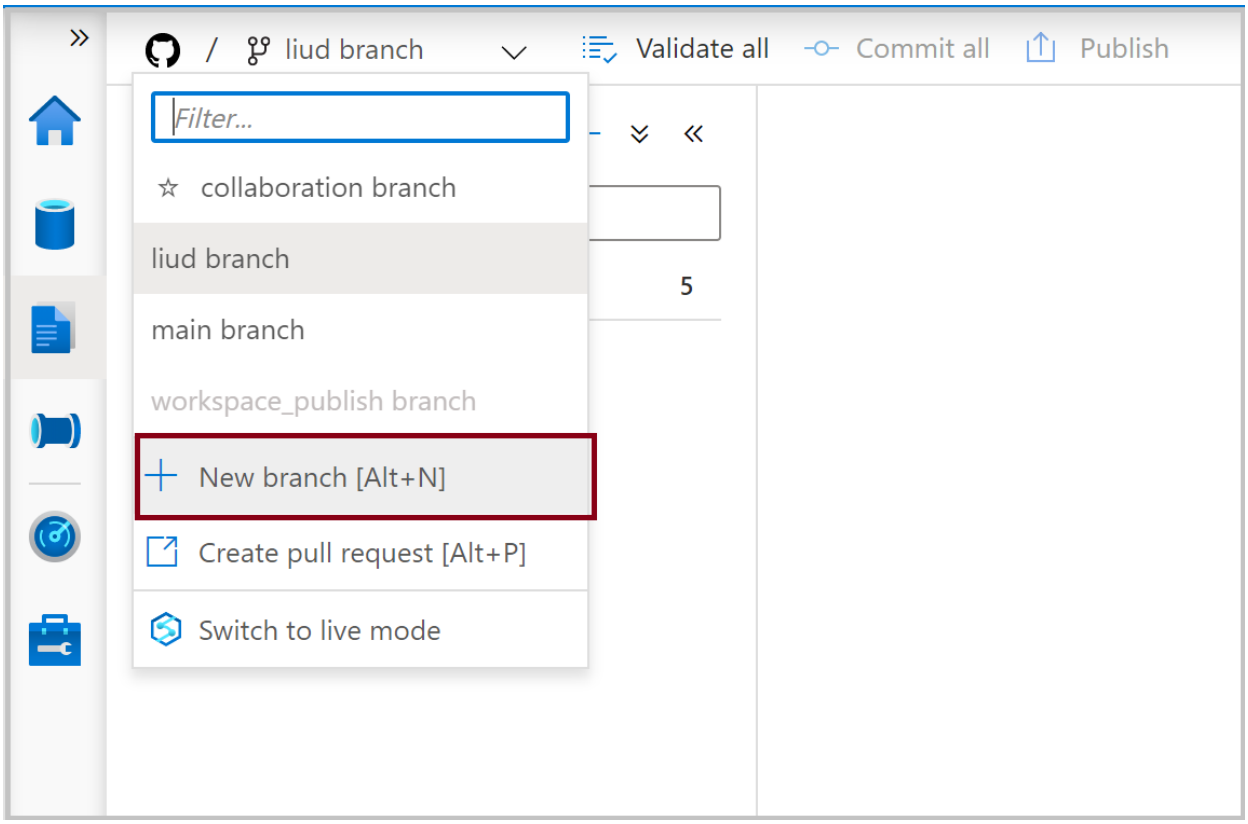
Once you complete these steps, your workspace will be able to connect to both public and private repositories within your organization.

Version control

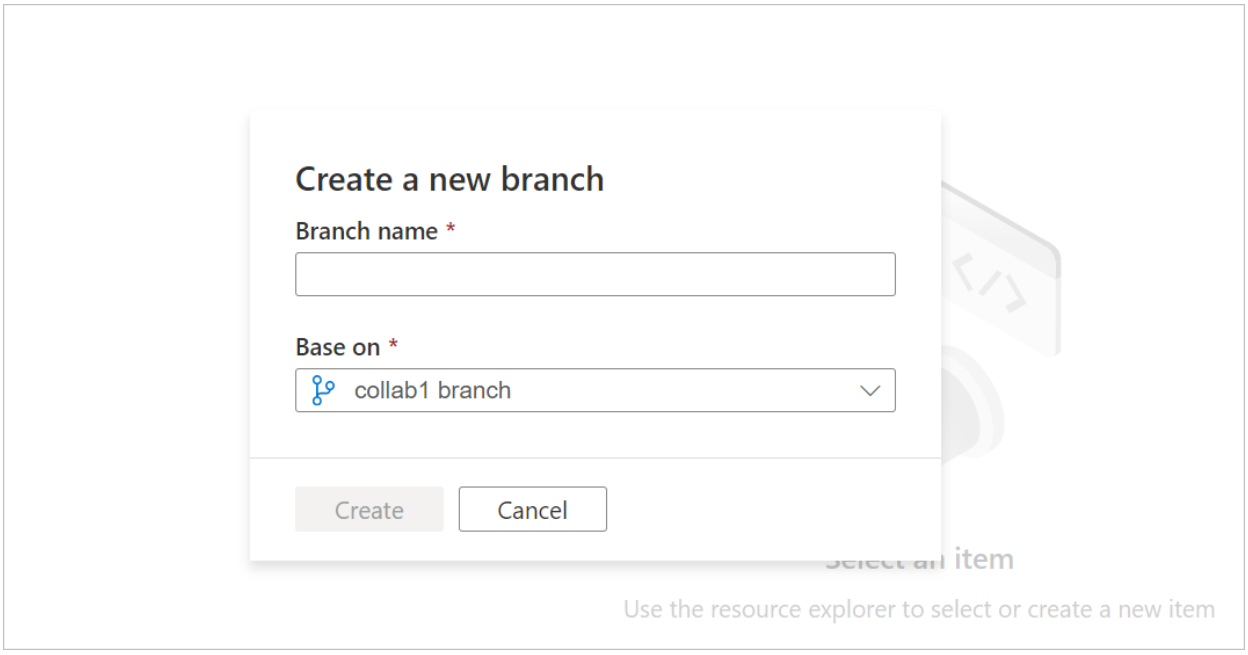
Version control systems (also known as *source control*) allows developers to collaborate on code and track changes.Source control is an essential tool for multi-developer projects.

Creating feature branches

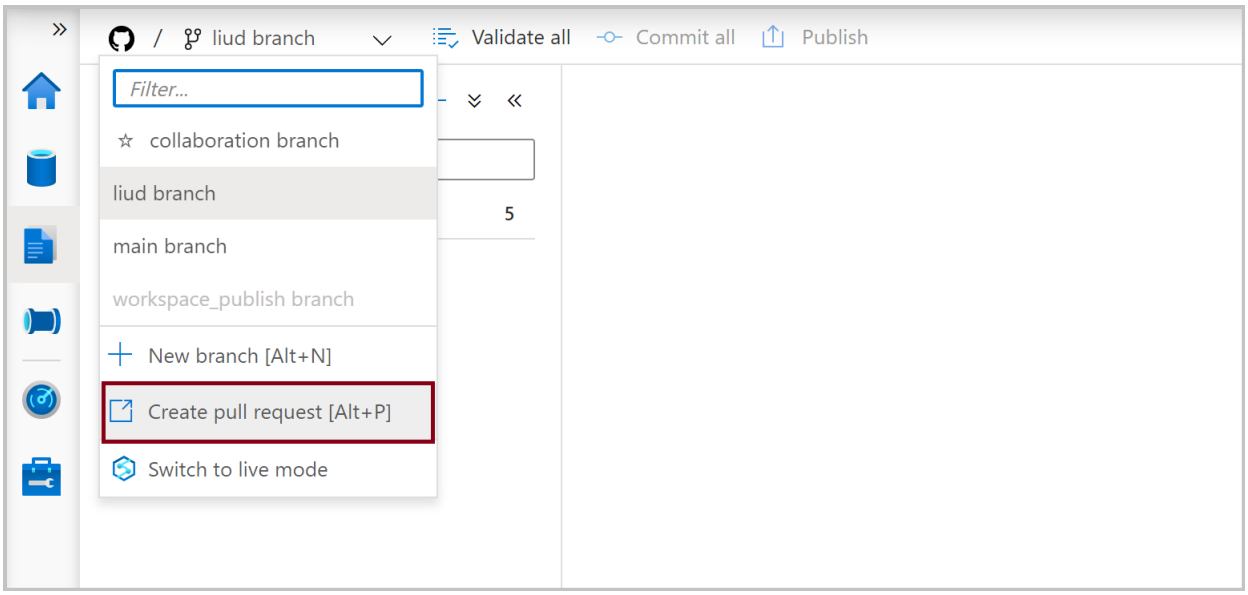
Each Git repository that's associated with a Synapse Studio has a collaboration branch. (`main` or `master` is the default collaboration branch). Users can also create feature branches by clicking **+ New Branch** in the branch dropdown.



Once the new branch pane appears, enter the name of your feature branch and select a branch to base the work off of.



When you are ready to merge the changes from your feature branch to your collaboration branch, click on the branch dropdown and select **Create pull request**. This action takes you to Git provider where you can raise pull requests, do code reviews, and merge changes to your collaboration branch. You are only allowed to publish to the Synapse service from your collaboration branch.



Configure publishing settings

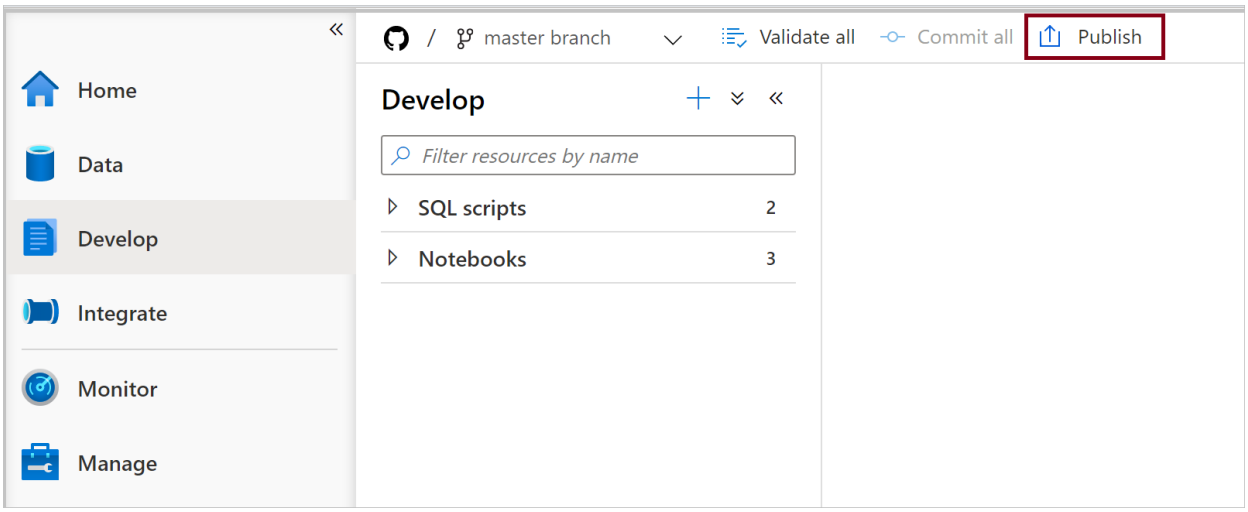
By default, Synapse Studio generates the workspace templates and saves them into a branch called `workspace_publish`. To configure a custom publish branch, add a `publish_config.json` file to the root folder in the collaboration branch. When publishing, Synapse Studio reads this file, looks for the field `publishBranch`, and saves workspace template files to the specified location. If the branch doesn't exist, Synapse Studio will automatically create it. And example of what this file looks like is below:

```
{
    "publishBranch": "workspace_publish"
}
```

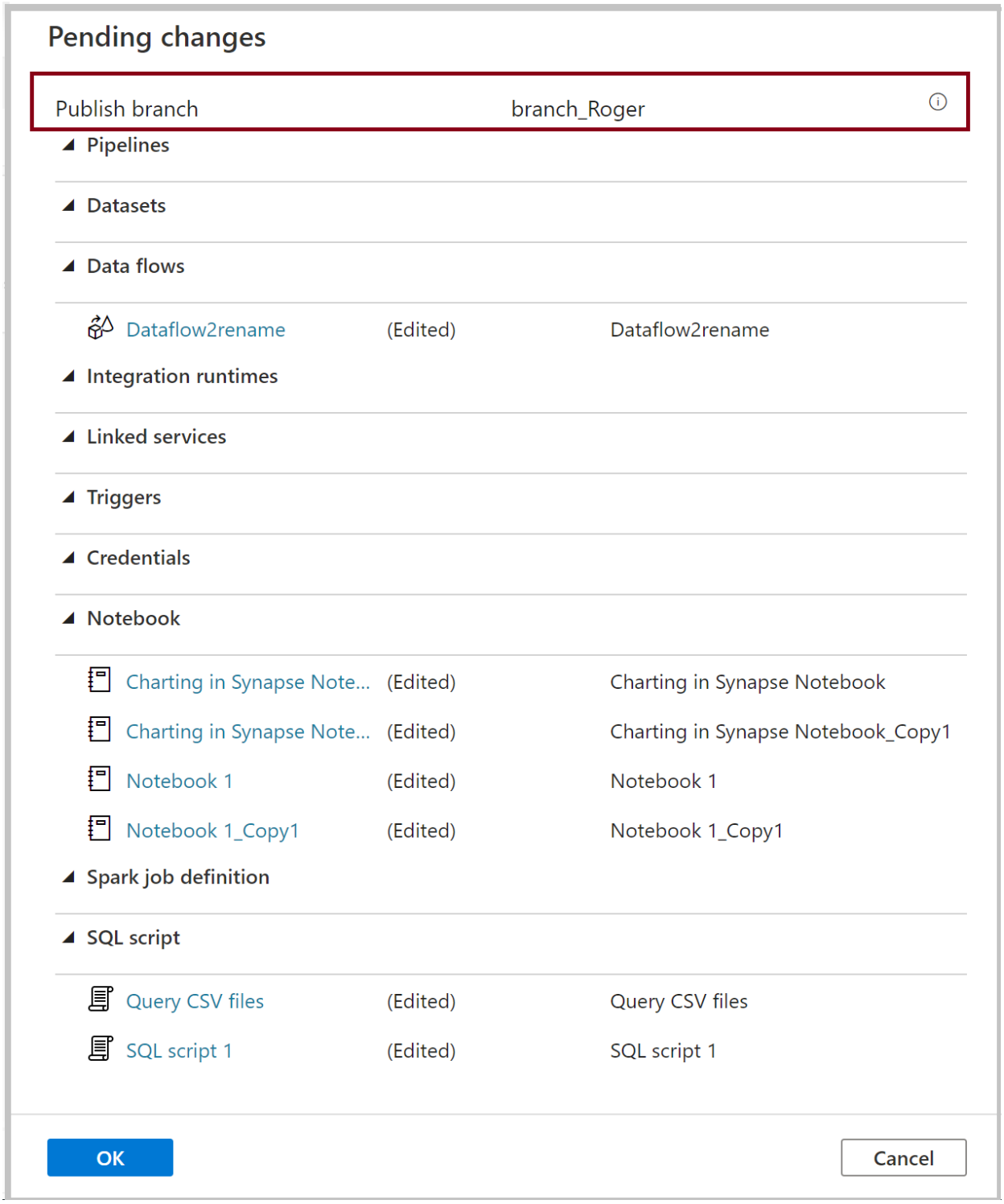
Synapse Studio can only have one publish branch at a time. When you specify a new publish branch, the previous publish branch would not been deleted. If you want to remove the previous publish branch, delete it manually.

Publish code changes

After merging changes to the collaboration branch , click **Publish** to manually publish your code changes in the collaboration branch to the Synapse service.



A side pane will open where you confirm that the publish branch and pending changes are correct. Once you verify your changes, click **OK** to confirm the publish.

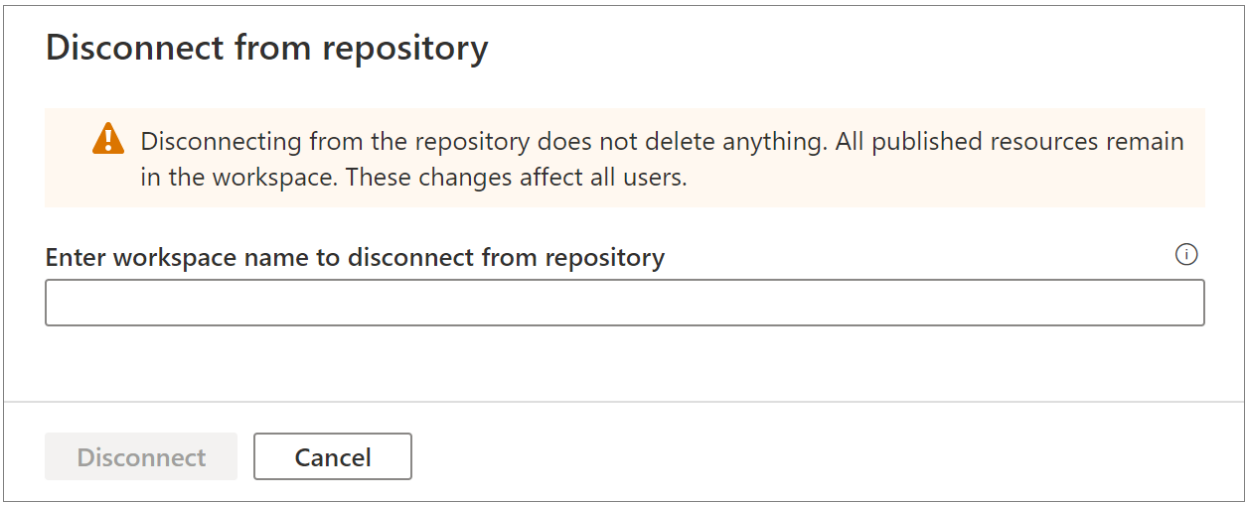


IMPORTANT

The collaboration branch is not representative of what's deployed in the service. The changes in collaboration branch *must* be published manually.

Switch to a different Git repository

To switch to a different Git repository, go to Git configuration page in the management hub under **Source control**. Select **Disconnect**.



Enter your workspace name and click **Disconnect** to remove the Git repository associated with your workspace.

After you remove the association with the current repo, you can configure your Git settings to use a different repo and then import existing resources to the new repo.

> **IMPORTANT**
> Removing Git configuration from a workspace doesn't delete anything from the repository. Synapse workspace will contain all published resources. You can continue to edit the workspace directly against the service.

Best practices for Git integration

- **Permissions**. After you have a git repository connected to your workspace, anyone who can access to your git repo with any role in your workspace will be able to update artifacts, like sql script, notebook,spark job definition, dataset, dataflow and pipeline in git mode. Typically you don't want every team member to have permissions to update workspace. Only grant git repository permission to Synapse workspace artifact authors.
- **Collaboration**. It's recommended to not allow direct check-ins to the collaboration branch. This restriction can help prevent bugs as every check-in will go through a pull request review process described in Creating feature branches.
- **Synapse live mode**. After publishing in git mode, all changes will be reflected in Synapse live mode. In Synapse live mode, publishing is disabled. And you can view, run artifacts in live mode if you have been granted the right permission.
- **Edit artifacts in Studio**. Synapse studio is the only place you can enable workspace source control and sync changes to git automatically. Any change via SDK, PowerShell, will not be synced to git. We recommend you always edit artifact in Studio when git is enabled.

Troubleshooting git integration

Access to git mode

If you have been granted the permission to the GitHub git repository linked with your workspace, but you can not access to Git mode:

1. Clear your cache and refresh the page.
2. Login your GitHub account.

Stale publish branch

If the publish branch is out of sync with the collaboration branch and contains out-of-date resources despite a recent publish, try following these steps:

1. Remove your current Git repository
2. Reconfigure Git with the same settings, but make sure **Import existing resources to repository** is checked and choose the same branch.
3. Create a pull request to merge the changes to the collaboration branch

Unsupported features

- Synapse Studio doesn't allow cherry-picking of commits or selective publishing of resources.
- Synapse Studio doesn't support customize commit message.
- By design, delete action in Studio will be committed to git directly

Next steps

- To implement continuous integration and deployment, see Continuous integration and delivery (CI/CD).

Continuous integration and delivery for an Azure Synapse Analytics workspace

8/19/2022 • 16 minutes to read • Edit Online

Continuous integration (CI) is the process of automating the build and testing of code every time a team member commits a change to version control. Continuous delivery (CD) is the process of building, testing, configuring, and deploying from multiple testing or staging environments to a production environment.

In an Azure Synapse Analytics workspace, CI/CD moves all entities from one environment (development, test, production) to another environment. Promoting your workspace to another workspace is a two-part process. First, use an Azure Resource Manager template (ARM template) to create or update workspace resources (pools and workspace). Then, migrate artifacts like SQL scripts and notebooks, Spark job definitions, pipelines, datasets, and other artifacts by using **Synapse Workspace Deployment** tools in Azure DevOps or on GitHub.

This article outlines how to use an Azure DevOps release pipeline and GitHub Actions to automate the deployment of an Azure Synapse workspace to multiple environments.

Prerequisites

To automate the deployment of an Azure Synapse workspace to multiple environments, the following prerequisites and configurations must be in place.

Azure DevOps

- Prepare an Azure DevOps project for running the release pipeline.
- Grant any users who will check in code Basic access at the organization level, so they can see the repository.
- Grant Owner permission to the Azure Synapse repository.
- Make sure that you've created a self-hosted Azure DevOps VM agent or use an Azure DevOps hosted agent.
- Grant permissions to create an Azure Resource Manager service connection for the resource group.
- An Azure Active Directory (Azure AD) administrator must install the Azure DevOps Synapse Workspace Deployment Agent extension in the Azure DevOps organization.
- Create or nominate an existing service account for the pipeline to run as. You can use a personal access token instead of a service account, but your pipelines won't work after the user account is deleted.

GitHub

- Create a GitHub repository that contains the Azure Synapse workspace artifacts and the workspace template.
- Make sure that you've created a self-hosted runner or use a GitHub-hosted runner.

Azure Active Directory

- If you're using a service principal, in Azure AD, create a service principal to use for deployment.
- If you're using a managed identity, enable the system-assigned managed identity on your VM in Azure as the agent or runner, and then add it to Azure Synapse Studio as Synapse admin.
- Use the Azure AD admin role to complete these actions.

Azure Synapse Analytics

> **NOTE**
>
> You can automate and deploy these prerequisites by using the same pipeline, an ARM template, or the Azure CLI, but these processes aren't described in this article.

- The "source" workspace that's used for development must be configured with a Git repository in Azure Synapse Studio. For more information, see Source control in Azure Synapse Studio.
- Set up a blank workspace to deploy to:
    1. Create a new Azure Synapse workspace.
    2. Grant the VM agent and the service principal Contributor permission to the resource group in which the new workspace is hosted.
    3. In the workspace, don't configure the Git repository connection.
    4. In the Azure portal, find the new Azure Synapse workspace, and then grant Owner permission to yourself and to the user that will run the Azure DevOps pipeline Azure Synapse workspace.
    5. Add the Azure DevOps VM agent and the service principal to the Contributor role for the workspace. (The role should have been inherited, but verify that it is.)
    6. In the Azure Synapse workspace, go to **Studio** > **Manage** > **Access Control**. Add the Azure DevOps VM agent and the service principal to the workspace admin group.
    7. Open the storage account that's used for the workspace. On the **Identity and access management** pane, add the VM agent and the service principal to the Storage Blob Data Contributor role.
    8. Create a key vault in the support subscription, and ensure that both the existing workspace and the new workspace have at least GET and LIST permissions to the vault.
    9. For the automated deployment to work, ensure that any connection strings that are specified in your linked services are in the key vault.

Other prerequisites

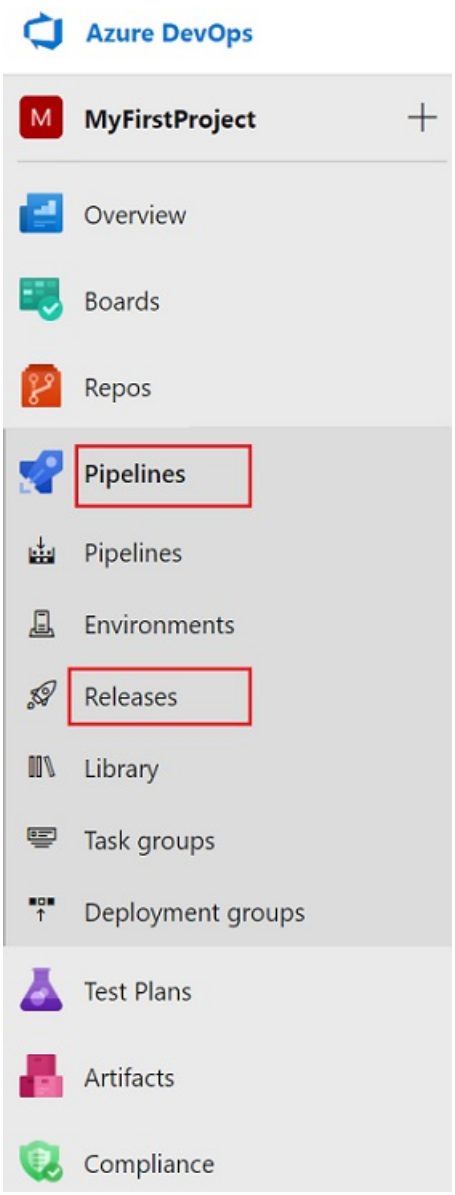
- Spark pools and self-hosted integration runtimes aren't created in a workspace deployment task. If you have a linked service that uses a self-hosted integration runtime, manually create the runtime in the new workspace.
- If the items in the development workspace are attached with the specific pools, make sure that you create or parameterize the same names for the pools in the target workspace in the parameter file.
- If your provisioned SQL pools are paused when you attempt to deploy, the deployment might fail.

For more information, see CI/CD in Azure Synapse Analytics Part 4 - The release pipeline.

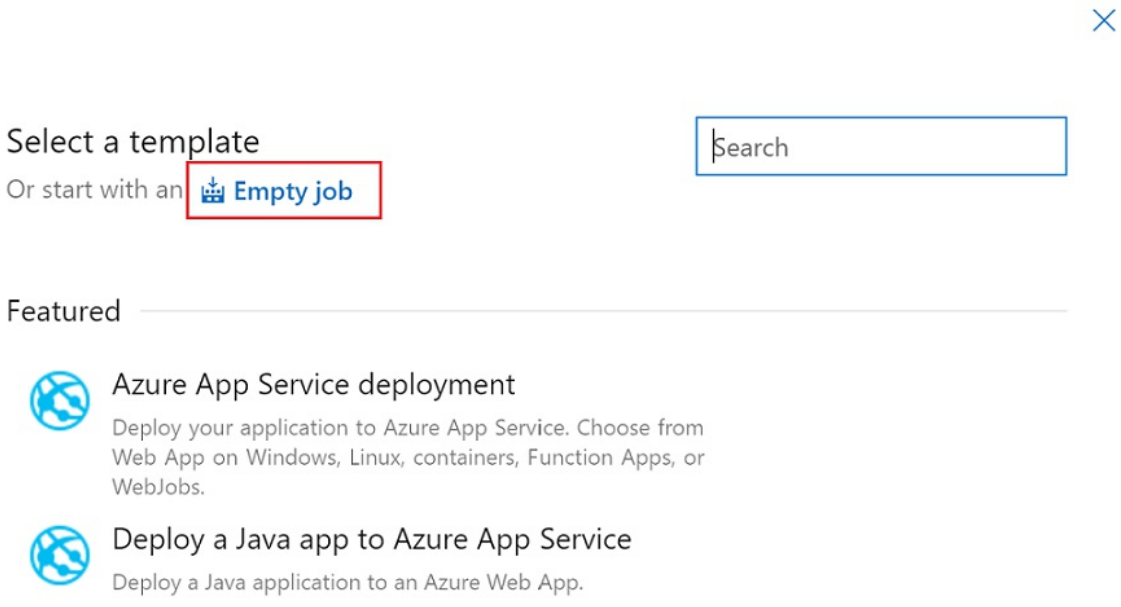
Set up a release pipeline in Azure DevOps

In this section, you'll learn how to deploy an Azure Synapse workspace in Azure DevOps.

1. In Azure DevOps, open the project you created for the release.
2. On the left menu, select **Pipelines** > **Releases**.

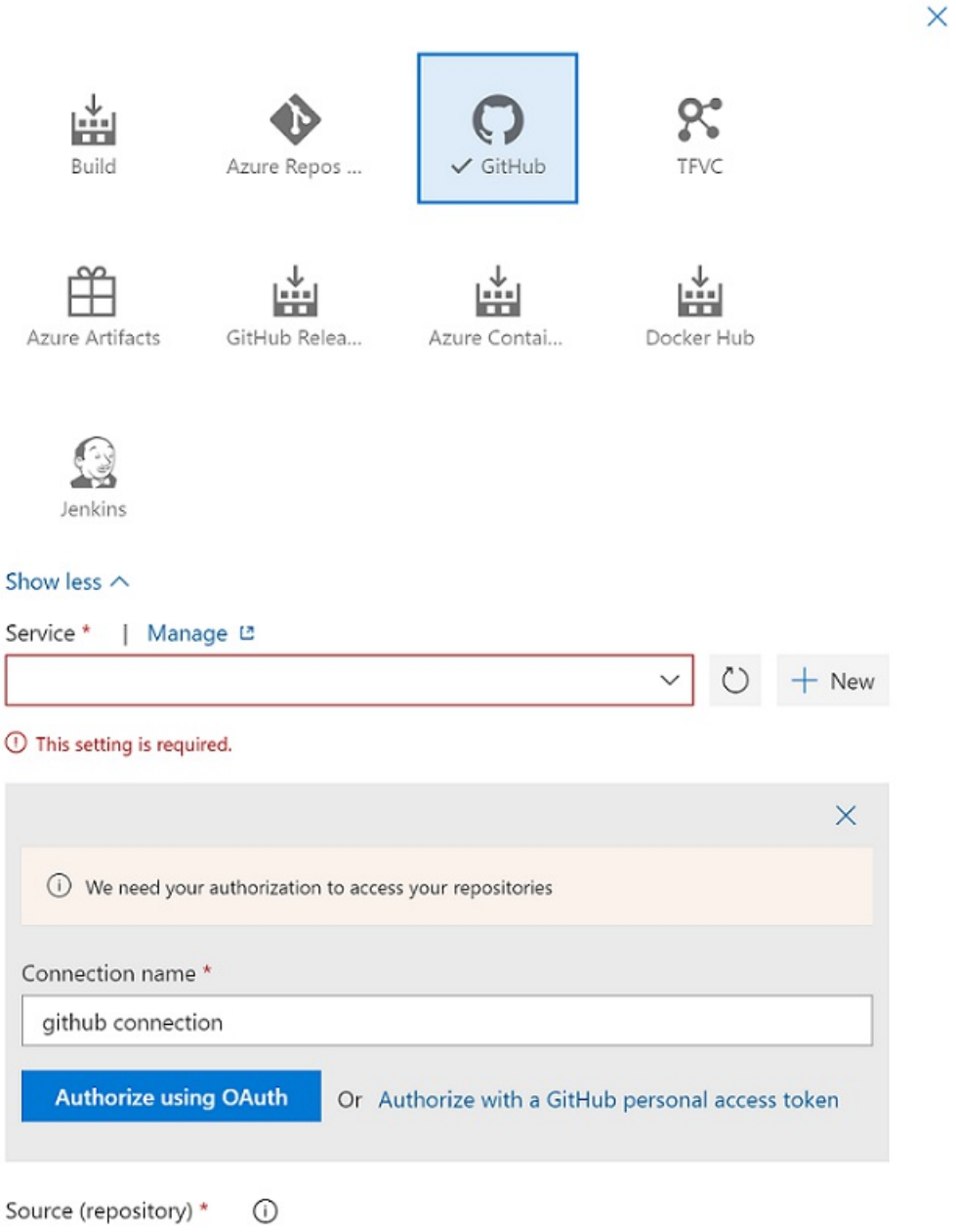


3. Select **New pipeline**. If you have existing pipelines, select **New** > **New release pipeline**.
4. Select the **Empty job** template.

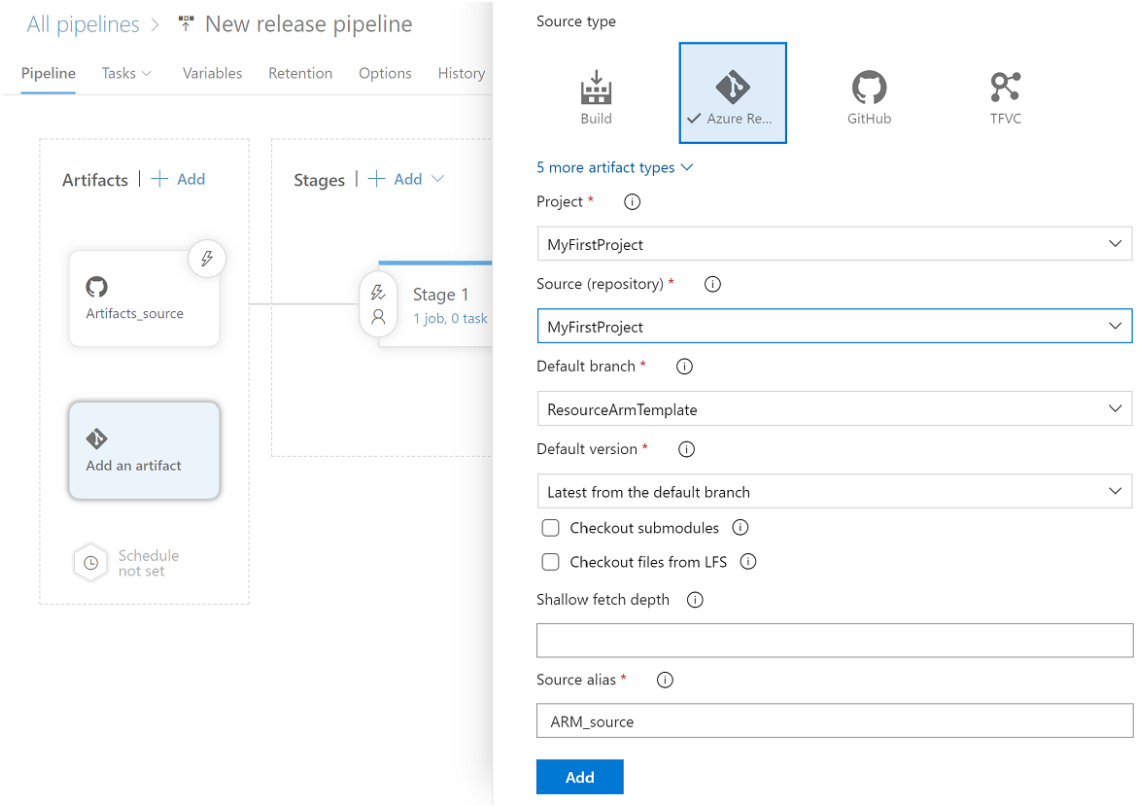


5. In **Stage name**, enter the name of your environment.

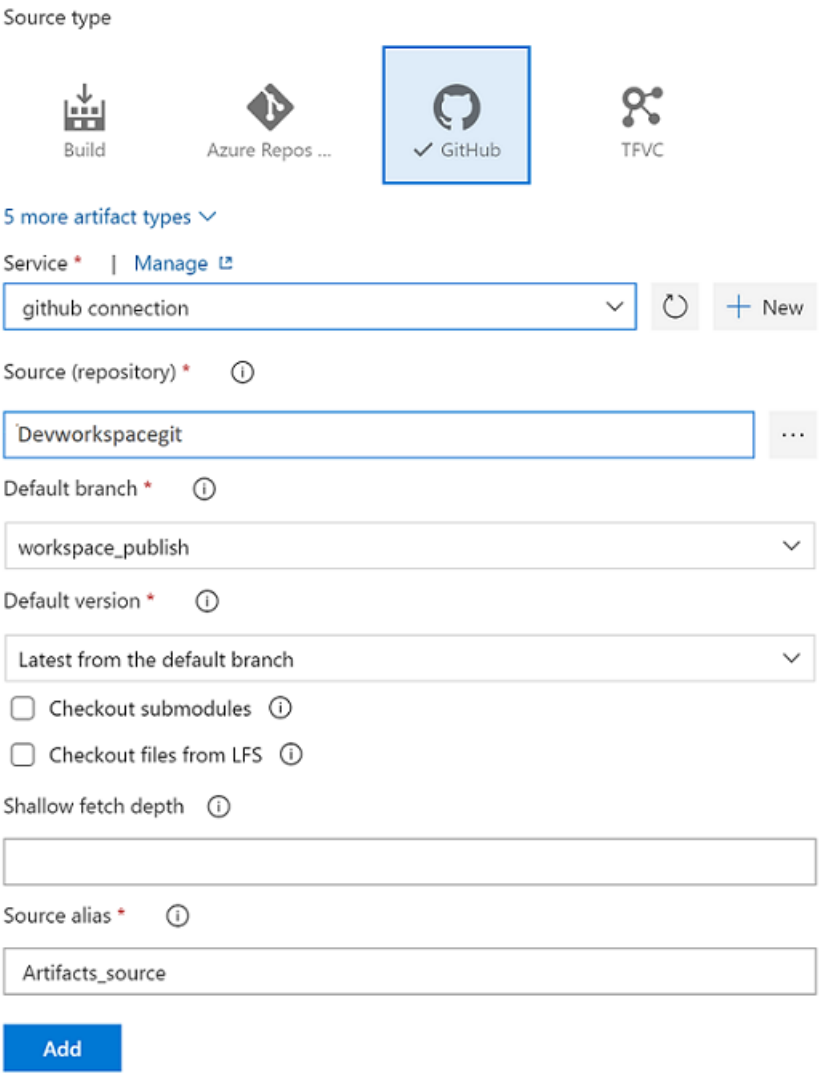
6. Select **Add artifact**, and then select the Git repository that's configured with Azure Synapse Studio in your development environment. Select the Git repository in which you manage your pools and workspace ARM template. If you use GitHub as the source, create a service connection for your GitHub account and pull repositories. For more information, see service connections.



7. Select the resource ARM template branch. For the **Default version**, select **Latest from default branch**.



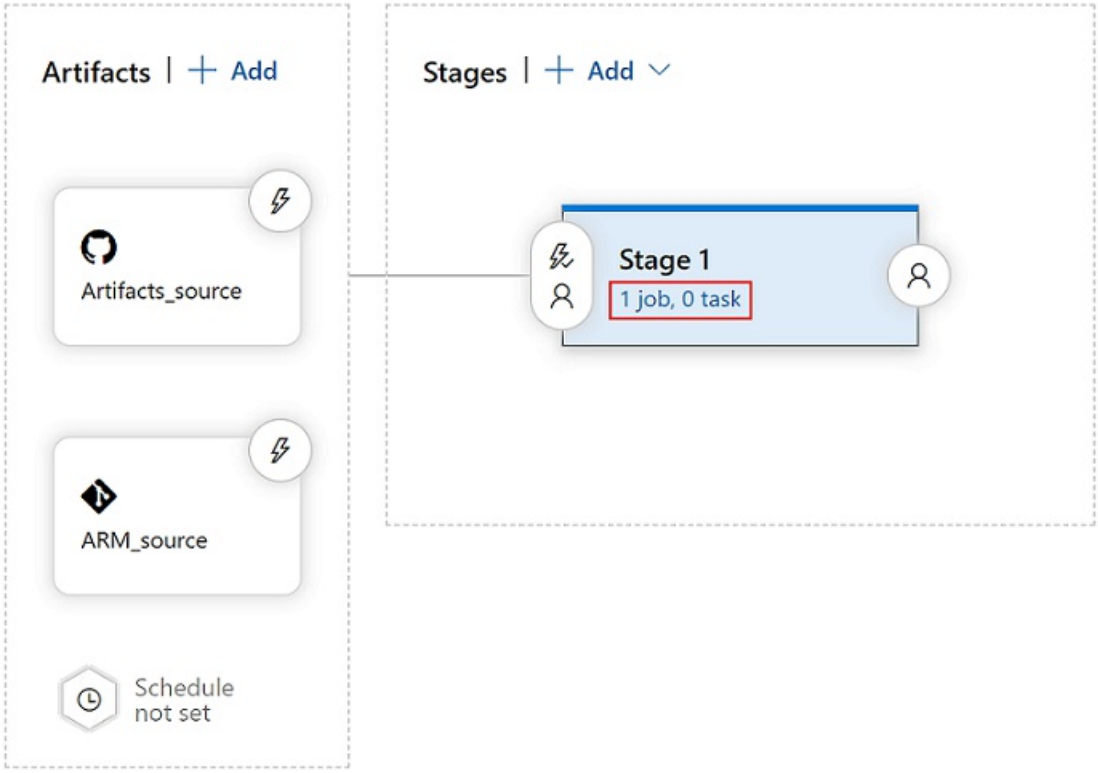
8. For the artifacts **Default branch**, select the repository publish branch or other non-publish branches which include Synapse artifacts. By default, the publish branch is `workspace_publish`. For the **Default version**, select **Latest from default branch**.



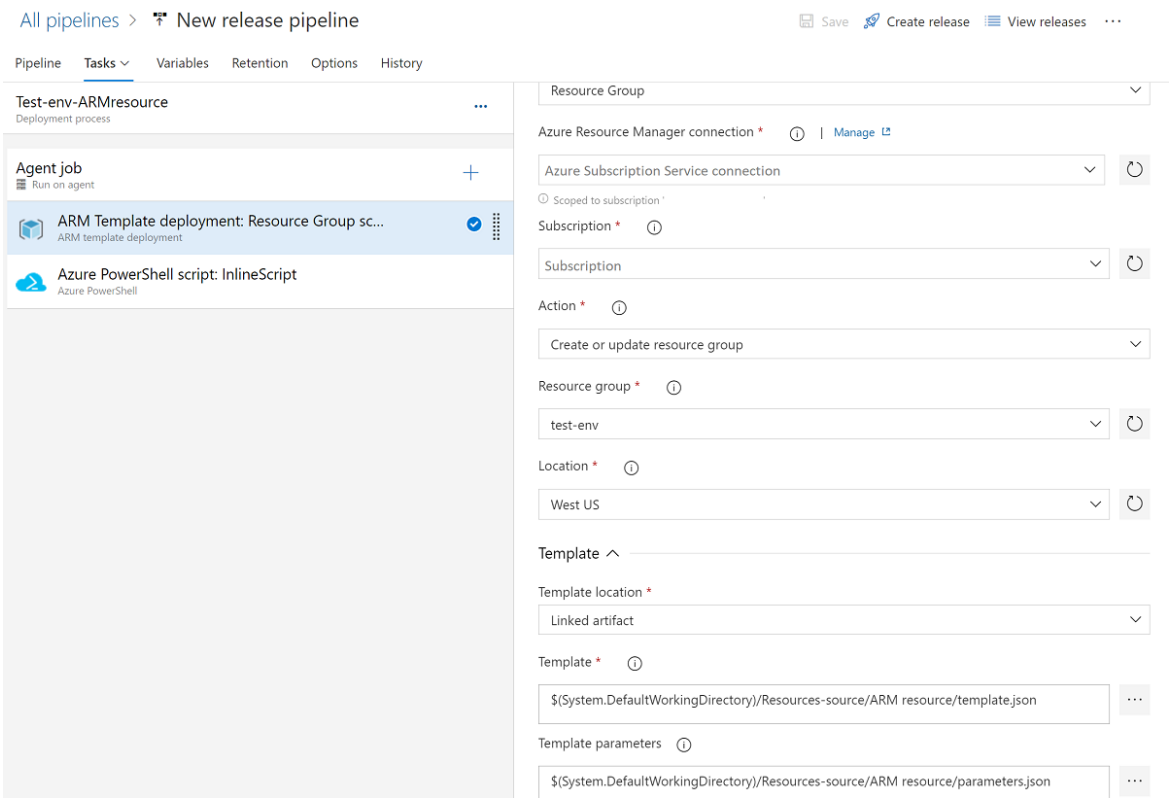
Set up a stage task for an ARM template to create and update a resource

If you have an ARM template that deploys a resource, such as an Azure Synapse workspace, a Spark and SQL pool, or a key vault, add an Azure Resource Manager deployment task to create or update those resources:

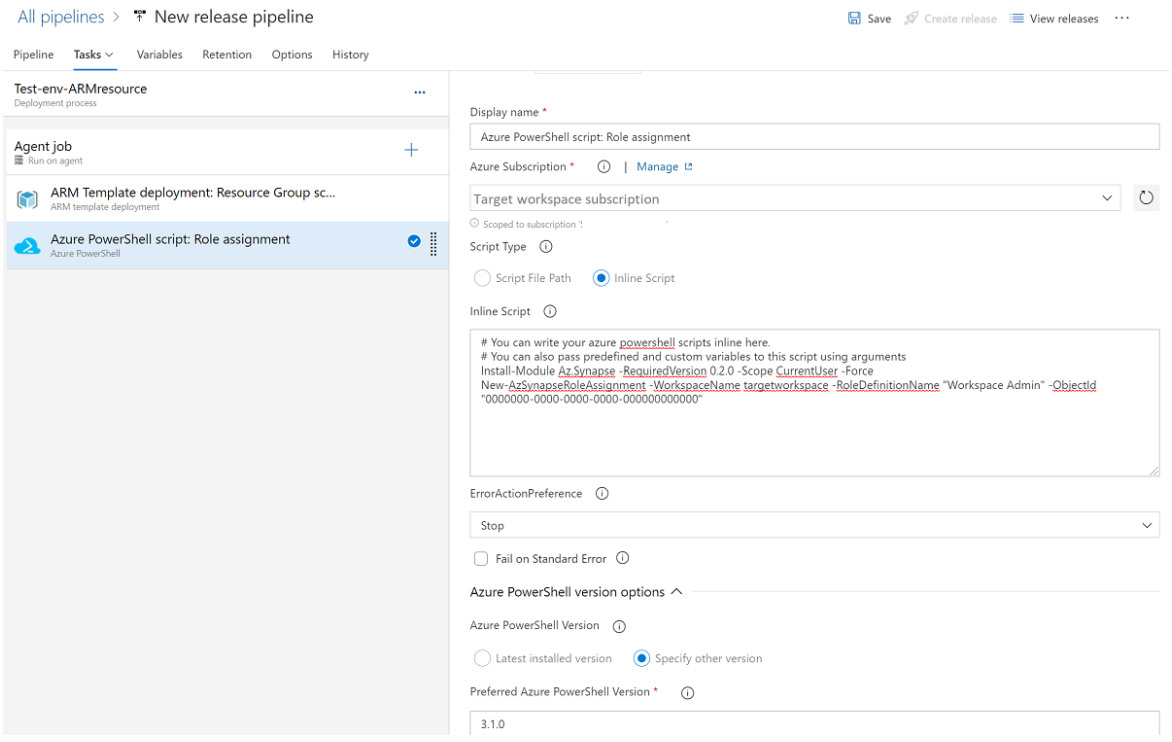
1. In the stage view, select **View stage tasks**.



2. Create a new task. Search for **ARM Template Deployment**, and then select **Add**.
3. On the deployment **Tasks** tab, select the subscription, resource group, and location for the workspace. Provide credentials if necessary.
4. For **Action**, select **Create or update resource group**.
5. For **Template**, select the ellipsis button (...). Go to the ARM template of the workspace.
6. For **Template parameters**, select ... to choose the parameters file.



7. For **Override template parameters**, select ..., and then enter the parameter values you want to use for the workspace.
8. For **Deployment mode**, select **Incremental**.
9. (Optional) Add **Azure PowerShell** for the grant and update the workspace role assignment. If you use a release pipeline to create an Azure Synapse workspace, the pipeline's service principal is added as the default workspace admin. You can run PowerShell to grant other accounts access to the workspace.



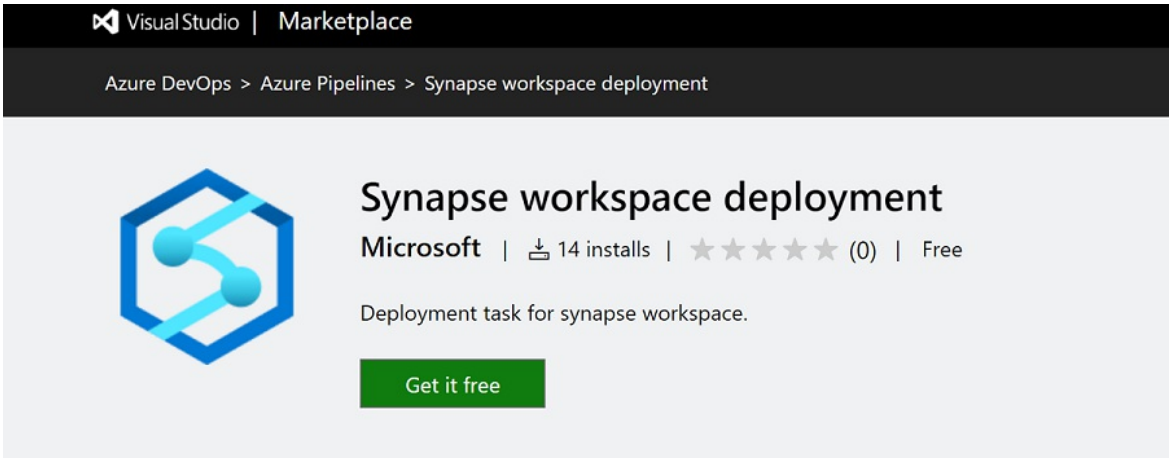
> **WARNING**
> In complete deployment mode, resources in the resource group that aren't specified in the new ARM template are *deleted*. For more information, see Azure Resource Manager deployment modes.

Set up a stage task for Azure Synapse artifacts deployment

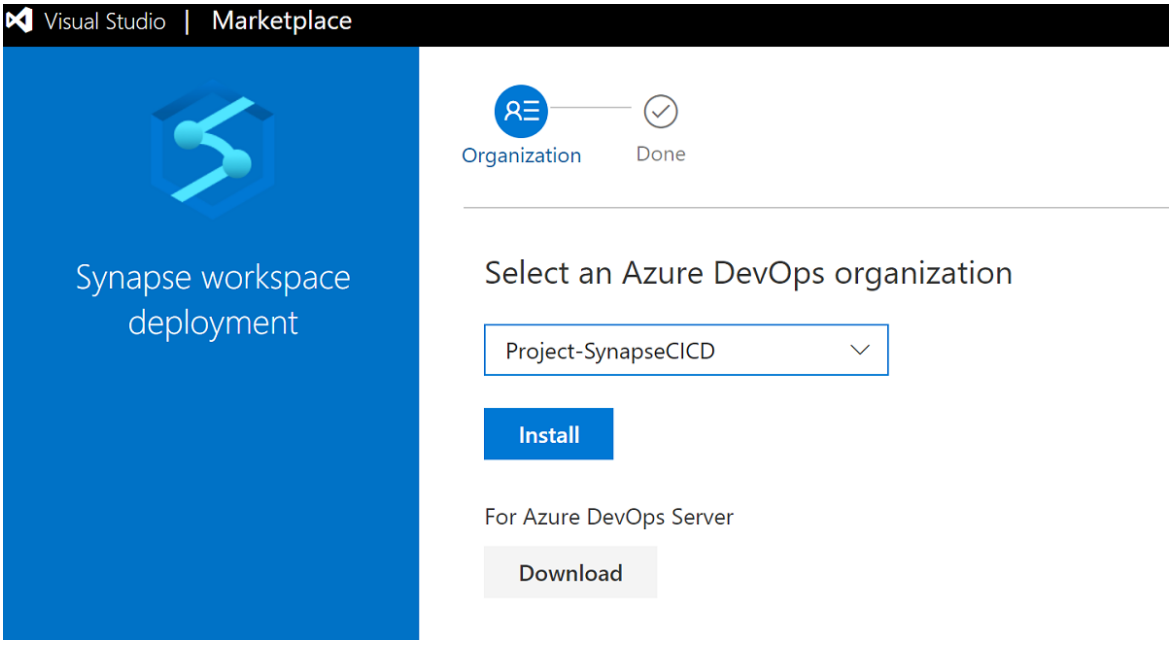
Use the Synapse workspace deployment extension to deploy other items in your Azure Synapse workspace. Items that you can deploy include datasets, SQL scripts and notebooks, spark job definitions, integration runtime, data flow, credentials, and other artifacts in workspace.

Install and add deployment extension

1. Search for and get the extension from Visual Studio Marketplace.

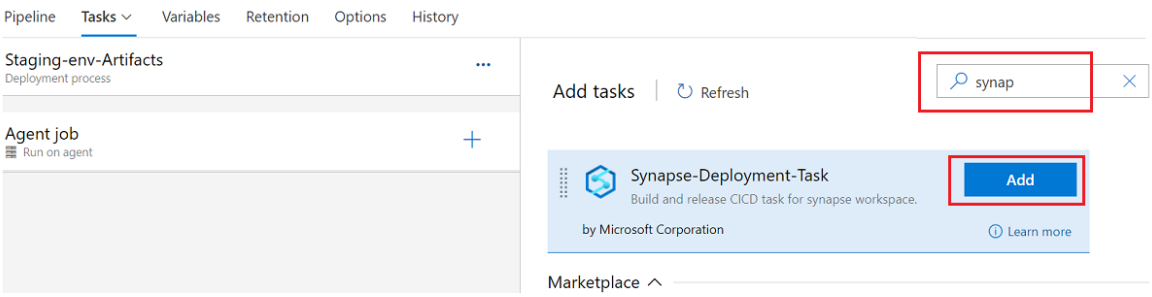


2. Select the Azure DevOps organization in which you want to install the extension.



3. Make sure that the Azure DevOps pipeline's service principal has been granted the Subscription permission and is assigned as the Synapse workspace admin for the workspace.

4. To create a new task, search for **Synapse workspace deployment**, and then select **Add**.



Configure the deployment task

The deployment task supports 3 types of operations, validate only, deploy and validate and deploy.

Validate is to validate the Synapse artifacts in non-publish branch with the task and generate the workspace template and parameter template file. The validation operation only works in the YAML pipeline. The sample YAML file is as below:

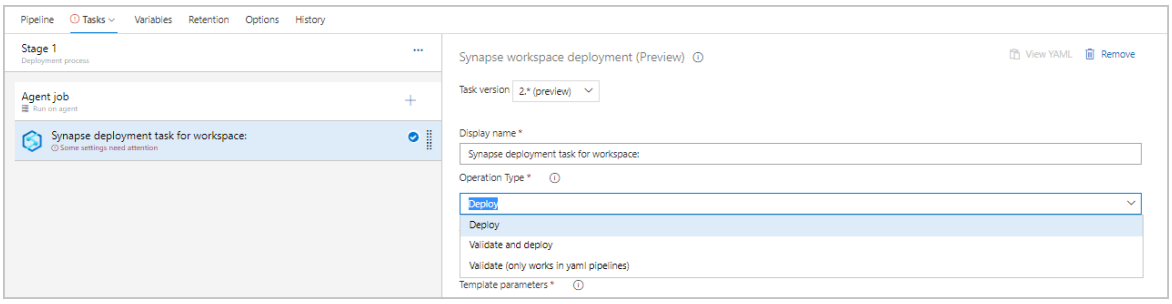
```
pool:
  vmImage: ubuntu-latest

resources:
  repositories:
  - repository: <repository name>
    type: git
    name: <name>
    ref: <user/collaboration branch>

steps:
  - checkout: <name>
  - task: Synapse workspace deployment@2
    continueOnError: true
    inputs:
      operation: 'validate'
      ArtifactsFolder: '$(System.DefaultWorkingDirectory)/ArtifactFolder'
      TargetWorkspaceName: '<target workspace name>'
```

Deploy The inputs of the operation deploy include Synapse workspace template and parameter template, which can be created after publishing in the workspace publish branch or after the validation. It is same as the version 1.x.

Validate and deploy can be used to directly deploy the workspace from non-publish branch with the artifact root folder.

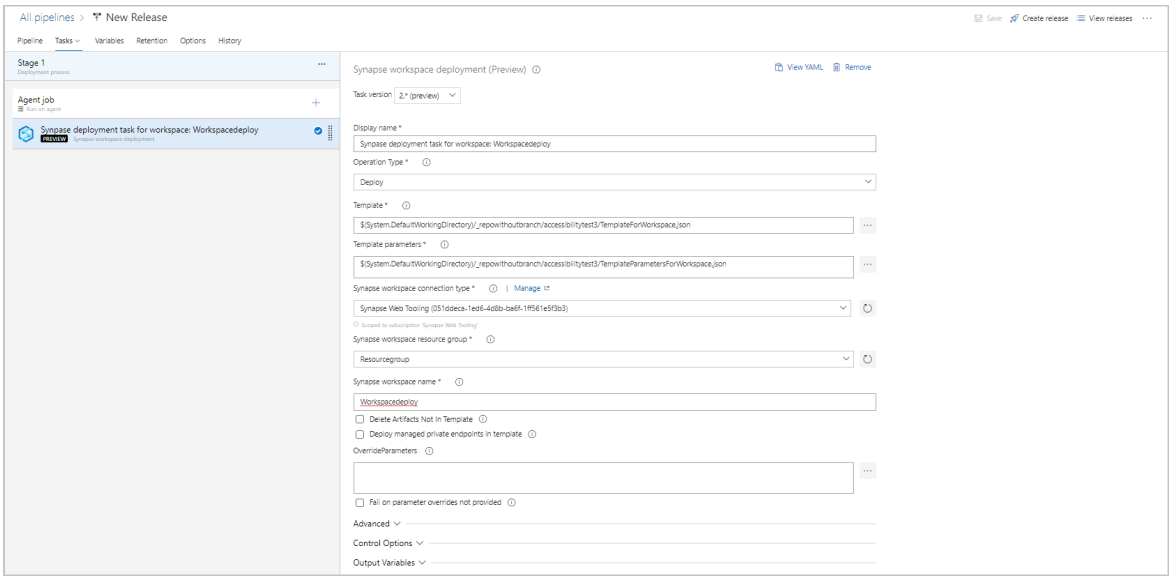
You can choose the operation types based on the use case. Following part is an example of the deploy.

1. In the task, select the operation type as **Deploy**.

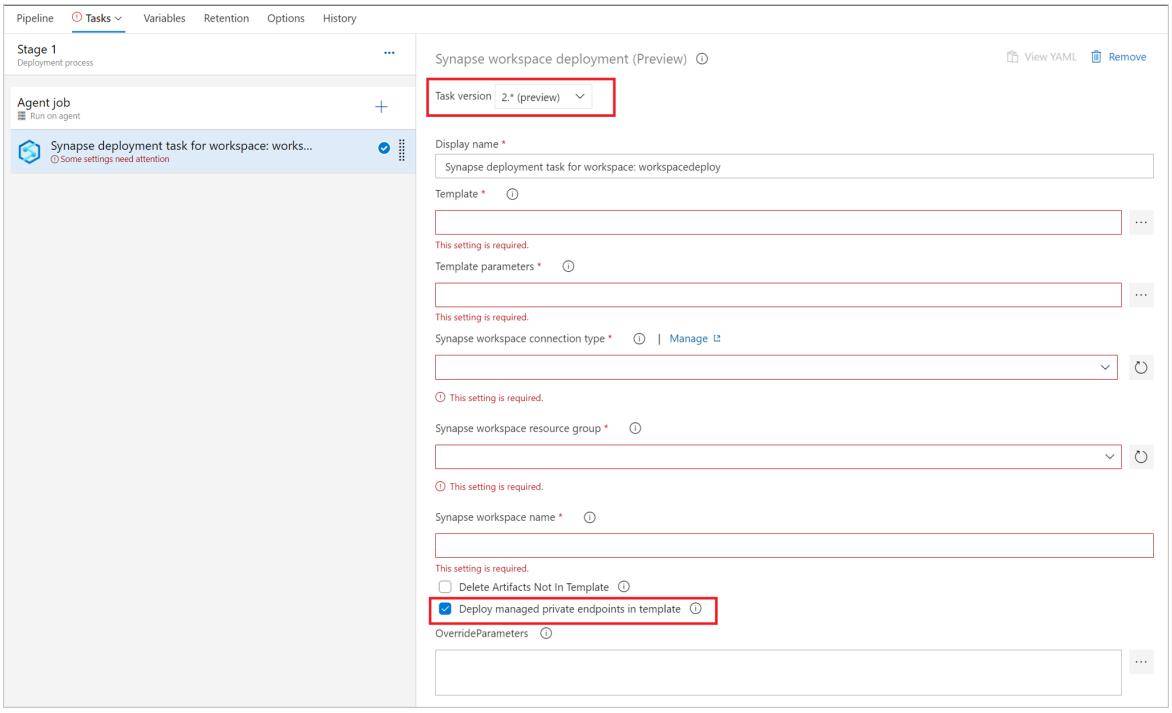


2. In the task, next to **Template**, select ... to choose the template file.
3. Next to **Template parameters**, select ... to choose the parameters file.
4. Select a connection, resource group, and name for the workspace.

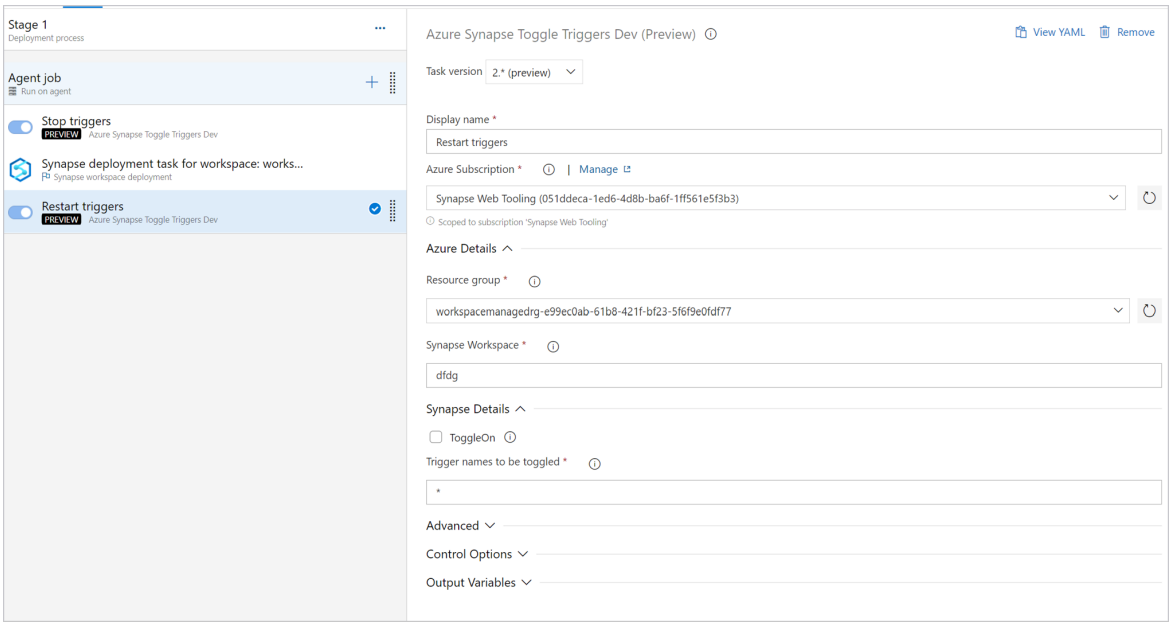
5. Next to **Override template parameters**, select ... . Enter the parameter values you want to use for the workspace, including connection strings and account keys that are used in your linked services. For more information, see CI/CD in Azure Synapse Analytics.



6. The deployment of managed private endpoint is only supported in version 2.x. please make sure you select the right version and check the **Deploy managed private endpoints in template**.



7. To manage triggers, you can use trigger toggle to stop the triggers before deployment. And you can also add a task to restart the triggers after the deployment task.

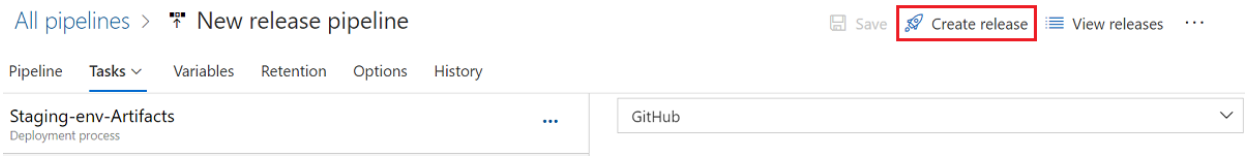


IMPORTANT

In CI/CD scenarios, the integration runtime type in different environments must be the same. For example, if you have a self-hosted integration runtime in the development environment, the same integration runtime must be self-hosted in other environments, such as in test and production. Similarly, if you're sharing integration runtimes across multiple stages, the integration runtimes must be linked and self-hosted in all environments, such as in development, test, and production.

Create a release for deployment

After you save all changes, you can select **Create release** to manually create a release. To learn how to automate release creation, see Azure DevOps release triggers.



Set up a release in GitHub Actions

In this section, you'll learn how to create GitHub workflows by using GitHub Actions for Azure Synapse workspace deployment.

You can use the GitHub Actions for Azure Resource Manager template to automate deploying an ARM template to Azure for the workspace and compute pools.

Workflow file

Define a GitHub Actions workflow in a YAML (.yml) file in the */.github/workflows/* path in your repository. The definition contains the various steps and parameters that make up the workflow.

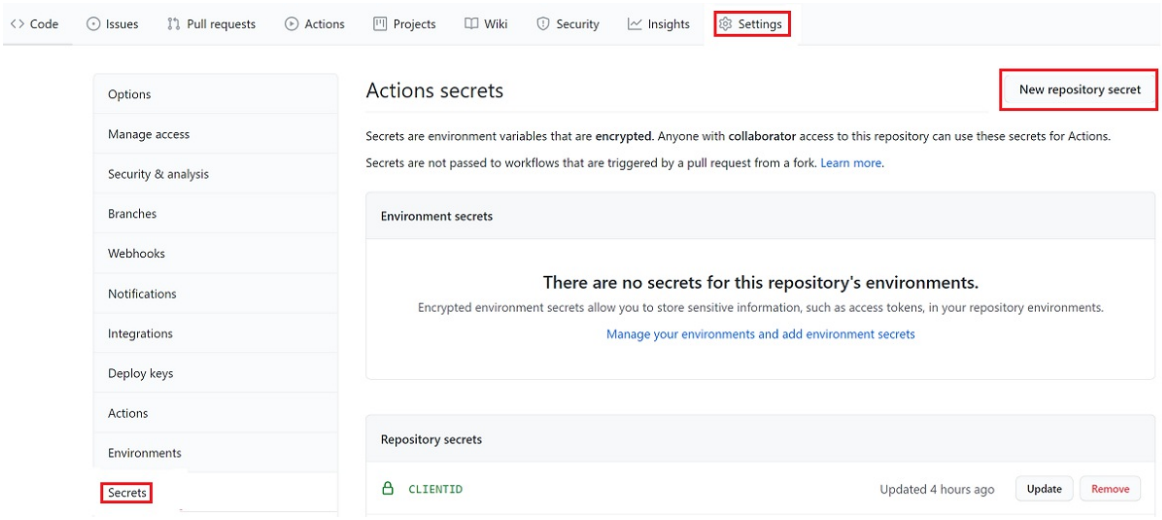
The .yml file has two sections:

SECTION	TASKS
Authentication	1. Define a service principal. 2. Create a GitHub secret.
Deploy	Deploy the workspace artifacts.

Configure GitHub Actions secrets

GitHub Actions secrets are environment variables that are encrypted. Anyone who has Collaborator permission to this repository can use these secrets to interact with Actions in the repository.

1. In the GitHub repository, select the **Settings** tab, and then select **Secrets** > **New repository secret**.



2. Add a new secret for the client ID, and add a new client secret if you use the service principal for deployment. You can also choose to save the subscription ID and tenant ID as secrets.

Add your workflow

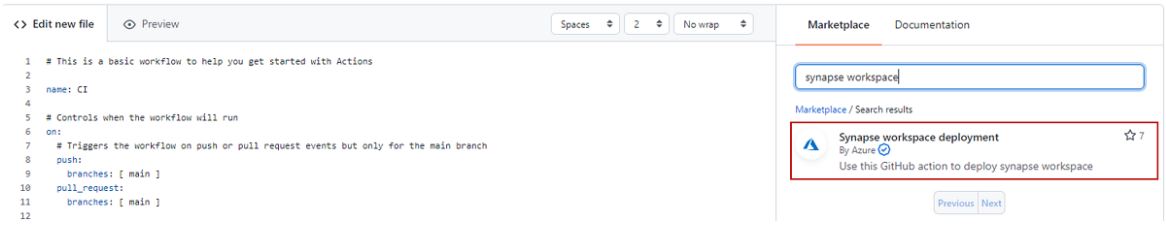
In your GitHub repository, go to **Actions**.

1. Select **Set up your workflow yourself**.

2. In the workflow file, delete everything after the `on:` section. For example, your remaining workflow might look like this example:

```
name: CI

on:
push:
    branches: [ master ]
pull_request:
    branches: [ master ]
```

3. Rename your workflow. On the **Marketplace** tab, search for the Synapse workspace deployment action, and then add the action.



4. Set the required values and the workspace template:

```
name: workspace deployment

on:
    push:
        branches: [ publish_branch ]
jobs:
    release:
        # You also can use the self-hosted runners.
        runs-on: windows-latest
        steps:
        # Checks out your repository under $GITHUB_WORKSPACE, so your job can access it.
        - uses: actions/checkout@v2
        - uses: azure/synapse-workspace-deployment@release-1.0
        with:
          TargetWorkspaceName: 'target workspace name'
          TemplateFile: './path of the TemplateForWorkspace.json'
          ParametersFile: './path of the TemplateParametersForWorkspace.json'
          OverrideArmParameters: './path of the parameters.yaml'
          environment: 'Azure Public'
          resourceGroup: 'target workspace resource group'
          clientId: ${{secrets.CLIENTID}}
          clientSecret:  ${{secrets.CLIENTSECRET}}
          subscriptionId: 'subscriptionId of the target workspace'
          tenantId: 'tenantId'
          DeleteArtifactsNotInTemplate: 'true'
          managedIdentity: 'False'
```

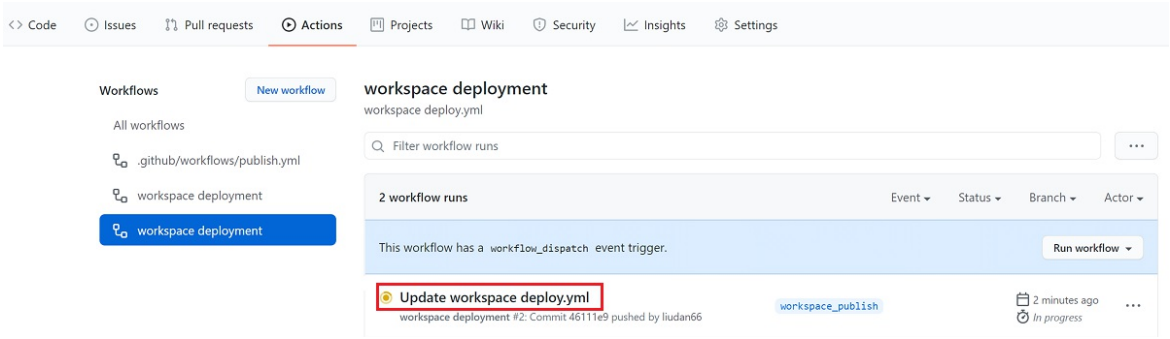
5. You're ready to commit your changes. Select **Start commit**, enter the title, and then add a description (optional). Then, select **Commit new file**.

The file appears in the *.github/workflows* folder in your repository.

> **NOTE**
> Managed identity is supported only with self-hosted VMs in Azure. Be sure to set the runner to self-hosted. Enable the system-assigned managed identity for your VM and add it to Azure Synapse Studio as Synapse admin.

Review your deployment

1. In your GitHub repository, go to **Actions**.
2. To see detailed logs of your workflow's run, open the first result:



Create custom parameters in the workspace template

If you use automated CI/CD and want to change some properties during deployment, but the properties aren't parameterized by default, you can override the default parameter template.

To override the default parameter template, create a custom parameter template named *template-parameters-definition.json* in the root folder of your Git branch. You must use this exact file name. When Azure Synapse workspace publishes from the collaboration branch or the deployment task validates the artifacts in other branches, it reads this file and uses its configuration to generate the parameters. If Azure Synapse workspace doesn't find that file, is uses the default parameter template.

Custom parameter syntax

You can use the following guidelines to create a custom parameters file:

- Enter the property path under the relevant entity type.
- Setting a property name to `*` indicates that you want to parameterize all properties under the property (only down to the first level, not recursively). You can set exceptions to this configuration.
- Setting the value of a property as a string indicates that you want to parameterize the property. Use the format `<action>:<name>:<stype>`.
  - `<action>` can be one of these characters:
    - `=` means keep the current value as the default value for the parameter.
    - `-` means don't keep the default value for the parameter.
    - `|` is a special case for secrets from Azure Key Vault for connection strings or keys.
  - `<name>` is the name of the parameter. If it's blank, it takes the name of the property. If the value starts with a `-` character, the name is shortened. For example, `AzureStorage1_properties_typeProperties_connectionString` would be shortened to `AzureStorage1_connectionString`.
  - `<stype>` is the type of parameter. If `<stype>` is blank, the default type is `string`. Supported values: `string`, `securestring`, `int`, `bool`, `object`, `secureobject` and `array`.
- Specifying an array in the file indicates that the matching property in the template is an array. Azure Synapse iterates through all the objects in the array by using the definition that's specified. The second object, a string, becomes the name of the property, which is used as the name for the parameter for each iteration.
- A definition can't be specific to a resource instance. Any definition applies to all resources of that type.
- By default, all secure strings (such as Key Vault secrets) and secure strings (such as connection strings, keys, and tokens) are parameterized.

Parameter template definition example

Here's an example of what a parameter template definition looks like:

```
{
    "Microsoft.Synapse/workspaces/notebooks": {
        "properties": {
            "bigDataPool": {
```

```
                "referenceName": "="
            }
        }
    },
    "Microsoft.Synapse/workspaces/sqlscripts": {
        "properties": {
            "content": {
                "currentConnection": {
                    "*": "-"
                }
            }
        }
    },
    "Microsoft.Synapse/workspaces/pipelines": {
        "properties": {
            "activities": [{
                "typeProperties": {
                    "waitTimeInSeconds": "-::int",
                    "headers": "=::object",
                    "activities": [
                        {
                            "typeProperties": {
                                "url": "-:-webUrl:string"
                            }
                        }
                    ]
                }
            }]
        }
    },
    "Microsoft.Synapse/workspaces/integrationRuntimes": {
        "properties": {
            "typeProperties": {
                "*": "="
            }
        }
    },
    "Microsoft.Synapse/workspaces/triggers": {
        "properties": {
            "typeProperties": {
                "recurrence": {
                    "*": "=",
                    "interval": "=:triggerSuffix:int",
                    "frequency": "=:-freq"
                },
                "maxConcurrency": "="
            }
        }
    },
    "Microsoft.Synapse/workspaces/linkedServices": {
        "*": {
            "properties": {
                "typeProperties": {
                    "accountName": "=",
                    "username": "=",
                    "connectionString": "|:-connectionString:secureString",
                    "secretAccessKey": "|"
                }
            }
        },
        "AzureDataLakeStore": {
            "properties": {
                "typeProperties": {
                    "dataLakeStoreUri": "="
                }
            }
        },
        "AzureKeyVault": {
            "properties": {
```

```
                "typeProperties": {
                    "baseUrl": "|:baseUrl:secureString"
                },
                "parameters": {
                    "KeyVaultURL": {
                        "type": "=",
                        "defaultValue": "|:defaultValue:secureString"
                    }
                }
            }
        }
    },
    "Microsoft.Synapse/workspaces/datasets": {
        "properties": {
            "typeProperties": {
                "*": "="
            }
        }
    },
    "Microsoft.Synapse/workspaces/credentials" : {
        "properties": {
            "typeProperties": {
                "resourceId": "="
            }
        }
    }
}
```

Here's an explanation of how the preceding template is constructed, by resource type.

`notebooks`

- Any property in the `properties/bigDataPool/referenceName` path is parameterized with its default value. You can parameterize an attached Spark pool for each notebook file.

`sqlscripts`

- In the `properties/content/currentConnection` path, both the `poolName` and the `databaseName` properties are parameterized as strings without the default values in the template.

`pipelines`

- Any property in the `activities/typeProperties/waitTimeInSeconds` path is parameterized. Any activity in a pipeline that has a code-level property named `waitTimeInSeconds` (for example, the `Wait` activity) is parameterized as a number, with a default name. The property won't have a default value in the Resource Manager template. Instead, the property will be required input during Resource Manager deployment.
- The `headers` property (for example, in a `Web` activity) is parameterized with the `object` type (Object). The `headers` property has a default value that is the same value as the source factory.

`integrationRuntimes`

- All properties in the `typeProperties` path are parameterized with their respective default values. For example, two properties are under `IntegrationRuntimes` type properties: `computeProperties` and `ssisProperties`. Both property types are created with their respective default values and types (Object).

`triggers`

- Under `typeProperties`, two properties are parameterized:
  - The `maxConcurrency` property has a default value and is the `string` type. The default parameter name of the `maxConcurrency` property is `<entityName>_properties_typeProperties_maxConcurrency`.
  - The `recurrence` property also is parameterized. All properties under the `recurrence` property are set

to be parameterized as strings, with default values and parameter names. An exception is the `interval` property, which is parameterized as the `int` type. The parameter name is suffixed with `<entityName>_properties_typeProperties_recurrence_triggerSuffix`. Similarly, the `freq` property is a string and is parameterized as a string. However, the `freq` property is parameterized without a default value. The name is shortened and suffixed, such as `<entityName>_freq`.

`linkedServices`

- Linked services are unique. Because linked services and datasets have a wide range of types, you can provide type-specific customization. In the preceding example, for all linked services of the `AzureDataLakeStore` type, a specific template is applied. For all others (identified through the use of the `*` character), a different template is applied.
- The `connectionString` property is parameterized as a `securestring` value. It doesn't have a default value. The parameter name is shortened and suffixed with `connectionString`.
- The `secretAccessKey` property is parameterized as an `AzureKeyVaultSecret` value (for example, in an Amazon S3 linked service). The property is automatically parameterized as an Azure Key Vault secret and fetched from the configured key vault. You also can parameterize the key vault itself.

`datasets`

- Although you can customize types in datasets, an explicit *-level configuration isn't required. In the preceding example, all dataset properties under `typeProperties` are parameterized.

Best practices for CI/CD

If you're using Git integration with your Azure Synapse workspace and you have a CI/CD pipeline that moves your changes from development to test, and then to production, we recommend these best practices:

- **Integrate only the development workspace with Git**. If you use Git integration, integrate only your *development* Azure Synapse workspace with Git. Changes to test and production workspaces are deployed via CI/CD and don't need Git integration.
- **Prepare pools before you migrate artifacts**. If you have a SQL script or notebook attached to pools in the development workspace, use the same name for pools in different environments.
- **Sync versioning in infrastructure as code scenarios**. To manage infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model, use the same versioning that the DevOps team uses for source code.
- **Review Azure Data Factory best practices**. If you use Data Factory, see the best practices for Data Factory artifacts.

Troubleshoot artifacts deployment

Use the Synapse workspace deployment task

In Azure Synapse, unlike in Data Factory, some artifacts aren't Resource Manager resources. You can't use the ARM template deployment task to deploy Azure Synapse artifacts. Instead, use the Synapse workspace deployment task.

Unexpected token error in release

If your parameter file has parameter values that aren't escaped, the release pipeline fails to parse the file and generates an `unexpected token` error. We suggest that you override parameters or use Key Vault to retrieve parameter values. You also can use double escape characters to resolve the issue.

Azure Policy Regulatory Compliance controls for Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

Regulatory Compliance in Azure Policy provides Microsoft created and managed initiative definitions, known as *built-ins*, for the **compliance domains** and **security controls** related to different compliance standards. This page lists the **compliance domains** and **security controls** for Azure App Configuration. You can assign the built-ins for a **security control** individually to help make your Azure resources compliant with the specific standard.

The title of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Policy Version** column to view the source on the Azure Policy GitHub repo.

> **IMPORTANT**
>
> Each control is associated with one or more Azure Policy definitions. These policies might help you assess compliance with the control. However, there often isn't a one-to-one or complete match between a control and one or more policies. As such, **Compliant** in Azure Policy refers only to the policies themselves. This doesn't ensure that you're fully compliant with all requirements of a control. In addition, the compliance standard includes controls that aren't addressed by any Azure Policy definitions at this time. Therefore, compliance in Azure Policy is only a partial view of your overall compliance status. The associations between controls and Azure Policy Regulatory Compliance definitions for these compliance standards can change over time.

CMMC Level 3

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see Azure Policy Regulatory Compliance - CMMC Level 3. For more information about this compliance standard, see Cybersecurity Maturity Model Certification (CMMC).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
System and Communications Protection	SC.3.177	Employ FIPS-validated cryptography when used to protect the confidentiality of CUI.	Azure Synapse workspaces should use customer-managed keys to encrypt data at rest	1.0.0

FedRAMP High

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see Azure Policy Regulatory Compliance - FedRAMP High. For more information about this compliance standard, see FedRAMP High.

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-4	Information Flow Enforcement	Azure Synapse workspaces should use private link	1.0.1

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Access Control	AC-17	Remote Access	Azure Synapse workspaces should use private link	1.0.1
Access Control	AC-17 (1)	Automated Monitoring / Control	Azure Synapse workspaces should use private link	1.0.1
Risk Assessment	RA-5	Vulnerability Scanning	Vulnerability assessment should be enabled on your Synapse workspaces	1.0.0
System and Communications Protection	SC-7	Boundary Protection	Azure Synapse workspaces should use private link	1.0.1
System and Communications Protection	SC-7 (3)	Access Points	Azure Synapse workspaces should use private link	1.0.1
System and Communications Protection	SC-12	Cryptographic Key Establishment and Management	Azure Synapse workspaces should use customer-managed keys to encrypt data at rest	1.0.0

FedRAMP Moderate

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see Azure Policy Regulatory Compliance - FedRAMP Moderate. For more information about this compliance standard, see FedRAMP Moderate.

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-4	Information Flow Enforcement	Azure Synapse workspaces should use private link	1.0.1
Access Control	AC-17	Remote Access	Azure Synapse workspaces should use private link	1.0.1
Access Control	AC-17 (1)	Automated Monitoring / Control	Azure Synapse workspaces should use private link	1.0.1
Risk Assessment	RA-5	Vulnerability Scanning	Vulnerability assessment should be enabled on your Synapse workspaces	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC-7	Boundary Protection	Azure Synapse workspaces should use private link	1.0.1
System and Communications Protection	SC-7 (3)	Access Points	Azure Synapse workspaces should use private link	1.0.1
System and Communications Protection	SC-12	Cryptographic Key Establishment and Management	Azure Synapse workspaces should use customer-managed keys to encrypt data at rest	1.0.0

NIST SP 800-53 Rev. 5

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see Azure Policy Regulatory Compliance - NIST SP 800-53 Rev. 5. For more information about this compliance standard, see NIST SP 800-53 Rev. 5.

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-4	Information Flow Enforcement	Azure Synapse workspaces should use private link	1.0.1
Access Control	AC-17	Remote Access	Azure Synapse workspaces should use private link	1.0.1
Access Control	AC-17 (1)	Monitoring and Control	Azure Synapse workspaces should use private link	1.0.1
Risk Assessment	RA-5	Vulnerability Monitoring and Scanning	Vulnerability assessment should be enabled on your Synapse workspaces	1.0.0
System and Communications Protection	SC-7	Boundary Protection	Azure Synapse workspaces should use private link	1.0.1
System and Communications Protection	SC-7 (3)	Access Points	Azure Synapse workspaces should use private link	1.0.1
System and Communications Protection	SC-12	Cryptographic Key Establishment and Management	Azure Synapse workspaces should use customer-managed keys to encrypt data at rest	1.0.0

Reserve Bank of India - IT Framework for NBFC

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see Azure Policy Regulatory Compliance - Reserve Bank of India - IT Framework for NBFC. For more information about this compliance standard, see Reserve Bank of India - IT Framework for NBFC.

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Information and Cyber Security	RBI IT Framework 3.3	Vulnerability Management-3.3	Vulnerability assessment should be enabled on your Synapse workspaces	1.0.0
IS Audit	RBI IT Framework 5	Policy for Information System Audit (IS Audit)-5	IP firewall rules on Azure Synapse workspaces should be removed	1.0.0

Next steps

- Learn more about Azure Policy Regulatory Compliance.
- See the built-ins on the Azure Policy GitHub repo.

Encryption for Azure Synapse Analytics workspaces

8/19/2022 • 5 minutes to read • Edit Online

This article will describe:

- Encryption of data at rest in Synapse Analytics workspaces.
- Configuration of Synapse workspaces to enable encryption with a customer managed key.
- Managing keys used to encrypt data in workspaces.

Encryption of data at rest

A complete Encryption-at-Rest solution ensures the data is never persisted in unencrypted form. Double encryption of data at rest mitigates threats with two, separate layers of encryption to protect against compromises of any single layer. Azure Synapse Analytics offers a second layer of encryption for the data in your workspace with a customer-managed key. This key is safeguarded in your Azure Key Vault, which allows you to take ownership of key management and rotation.

The first layer of encryption for Azure services is enabled with platform-managed keys. By default, Azure Disks, and data in Azure Storage accounts are automatically encrypted at rest. Learn more about how encryption is used in Microsoft Azure in the Azure Encryption Overview.

> **NOTE**
> Some items considered customer content, such as table names, object names, and index names, may be transmitted in log files for support and troubleshooting by Microsoft.

Azure Synapse encryption

This section will help you better understand how customer-managed key encryption is enabled and enforced in Synapse workspaces. This encryption uses existing keys or new keys generated in Azure Key Vault. A single key is used to encrypt all the data in a workspace. Synapse workspaces support RSA 2048 and 3072 byte-sized keys, and RSA-HSM keys.

> **NOTE**
> Synapse workspaces do not support the use of EC, EC-HSM, and oct-HSM keys for encryption.

The data in the following Synapse components is encrypted with the customer-managed key configured at the workspace level:

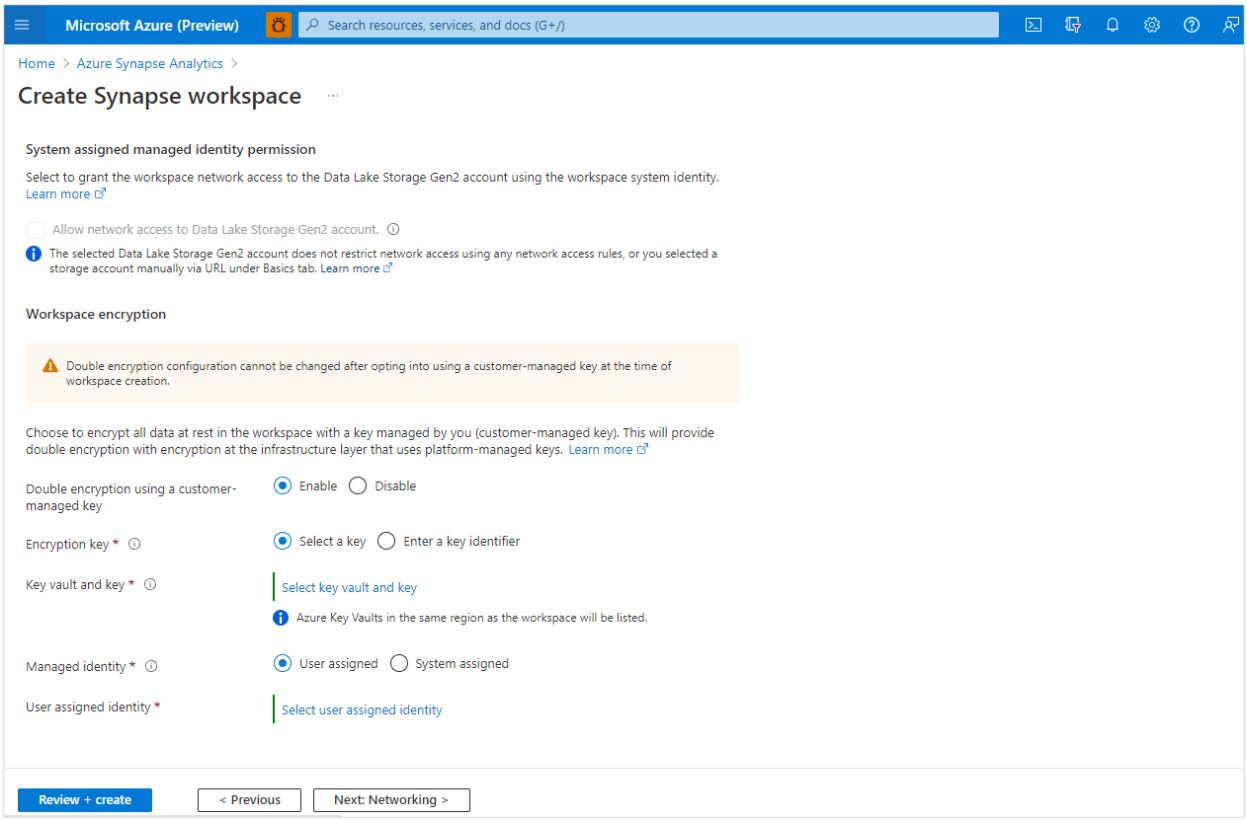
- SQL pools
- Dedicated SQL pools
- Serverless SQL pools
- Apache Spark pools
- Azure Data Factory integration runtimes, pipelines, datasets.

Workspace encryption configuration

Workspaces can be configured to enable double encryption with a customer-managed key at the time of

workspace creation. Enable double encryption using a customer-managed key on the "Security" tab when creating your new workspace. You can choose to enter a key identifier URI or select from a list of key vaults in the **same region** as the workspace. The Key Vault itself needs to have **purge protection enabled**.

> **IMPORTANT**
>
> The configuration setting for double encryption cannot be changed after the workspace is created.



Key access and workspace activation

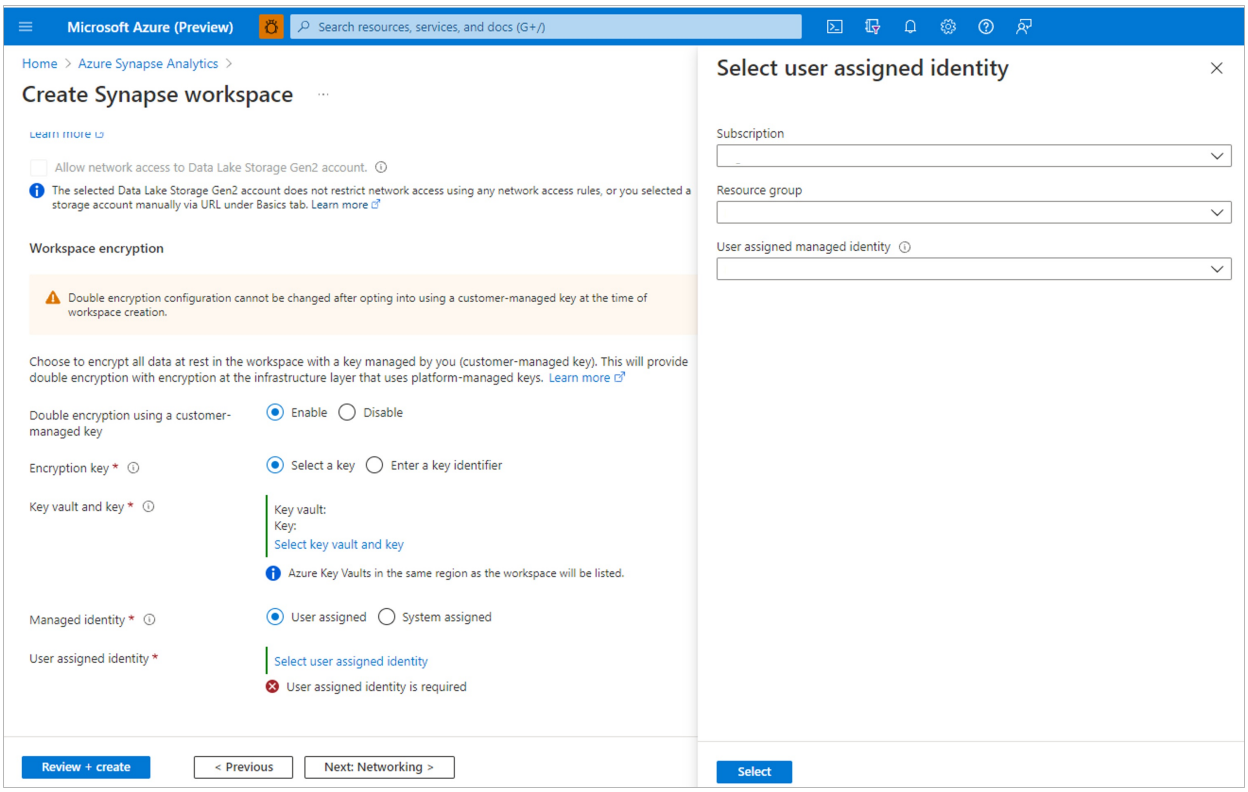
The Azure Synapse encryption model with customer-managed keys involves the workspace accessing the keys in Azure Key Vault to encrypt and decrypt as needed. The keys are made accessible to the workspace either through an access policy or Azure Key Vault RBAC. When granting permissions via an Azure Key Vault access policy, choose the "Application-only" option during policy creation (select the workspaces managed identity and do not add it as an authorized application).

The workspace managed identity must be granted the permissions it needs on the key vault before the workspace can be activated. This phased approach to workspace activation ensures that data in the workspace is encrypted with the customer-managed key. Encryption can be enabled or disabled for individual dedicated SQL Pools. Each dedicated pool is not enabled for encryption by default.

Using a User-assigned Managed identity

Workspaces can be configured to use a User-assigned Managed identity to access your customer-managed key stored in Azure Key Vault. Configure a User-assigned Managed identity to avoid phased activation of your Azure Synapse workspace when using double encryption with customer-managed keys. The Managed Identity Contributor built-in role is required to assign a user-assigned managed identity to an Azure Synapse workspace.

> **NOTE**
>
> A User-assigned Managed Identity cannot be configured to access customer-managed key when Azure Key Vault is behind a firewall.



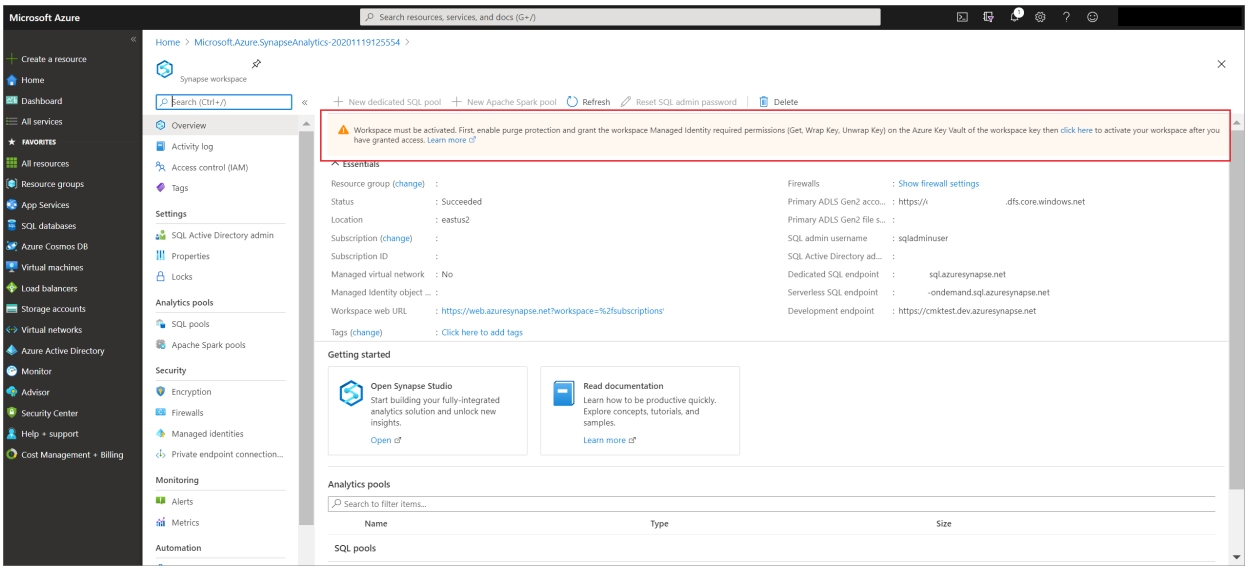
Permissions

To encrypt or decrypt data at rest, the managed identity must have the following permissions. Similarly, if you are using a Resource Manager template to create a new key, the 'keyOps' parameter of the template must have the following permissions:

- WrapKey (to insert a key into Key Vault when creating a new key).
- UnwrapKey (to get the key for decryption).
- Get (to read the public part of a key)

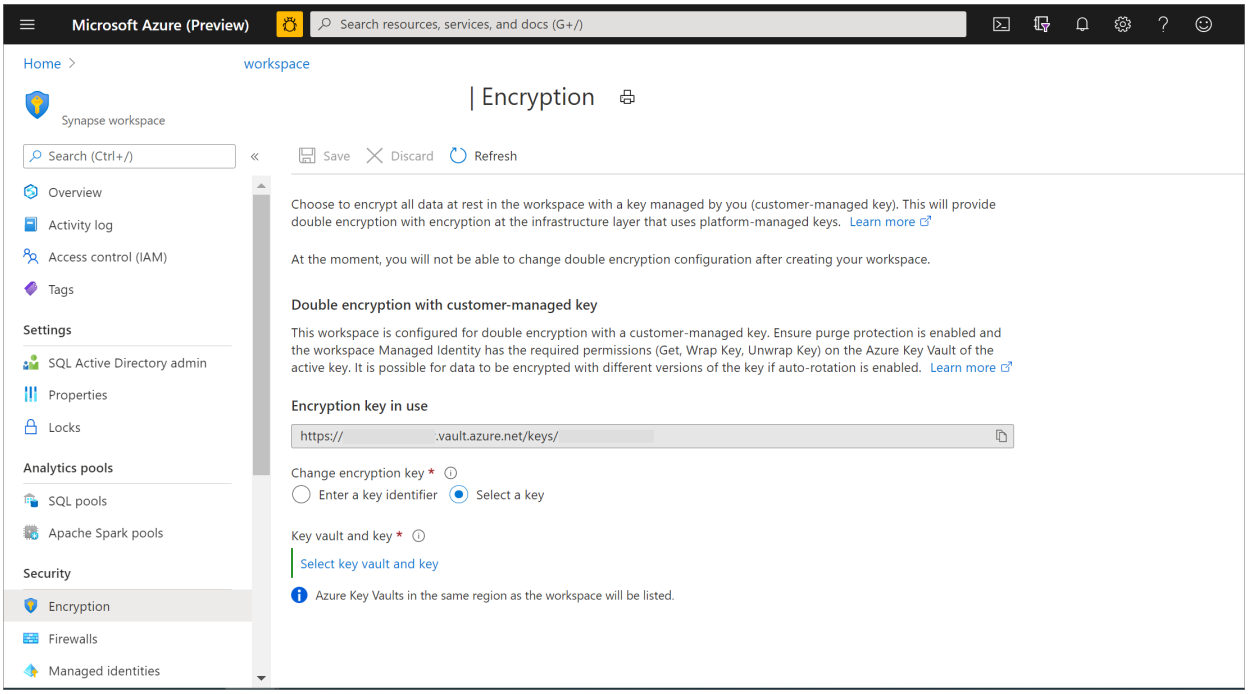
Workspace activation

If you do not configure a user-assigned managed identity to access customer managed keys during workspace creation, your workspace will remain in a "Pending" state until activation succeeds. The workspace must be activated before you can fully use all functionality. For example, you can only create a new dedicated SQL pool once activation succeeds. Grant the workspace managed identity access to the key vault and select the activation link in the workspace Azure portal banner. Once the activation completes successfully, your workspace is ready to use with the assurance that all data in it's protected with your customer-managed key. As previously noted, the key vault must have purge protection enabled for activation to succeed.



Manage the workspace customer-managed key

You can change the customer-managed key used to encrypt data from the **Encryption** page in the Azure portal. Here too, you can choose a new key using a key identifier or select from Key Vaults that you have access to in the same region as the workspace. If you choose a key in a different key vault from the ones previously used, grant the workspace-managed identity "Get", "Wrap", and "Unwrap" permissions on the new key vault. The workspace will validate its access to the new key vault and all data in the workspace will be re-encrypted with the new key.



> **IMPORTANT**
>
> When changing the encryption key of a workspace, retain the key until you replace it in the workspace with a new key. This is to allow decryption of data with the old key before it gets re-encrypted with the new key.

Azure Key Vaults policies for automatic, periodic rotation of keys, or actions on the keys can result in the creation of new key versions. You can choose to re-encrypt all the data in the workspace with the latest version of the active key. To-re-encrypt, change the key in the Azure portal to a temporary key and then switch back to the key you wish to use for encryption. As an example, to update data encryption using the latest version of active key Key1, change the workspace customer-managed key to temporary key, Key2. Wait for encryption with Key2 to finish. Then switch the workspace customer-managed key back to Key1-data in the workspace will be re-encrypted with the latest version of Key1.

> **NOTE**
>
> Azure Synapse Analytics does not automatically re-encrypt data when new key versions are created. To ensure consistency in your workspace, force the re-encryption of data using the process detailed above.

SQL Transparent Data Encryption with service-managed keys

SQL Transparent Data Encryption (TDE) is available for dedicated SQL Pools in workspaces *not* enabled for double encryption. In this type of workspace, a service-managed key is used to provide double encryption for the data in the dedicated SQL pools. TDE with the service-managed key can be enabled or disabled for individual dedicated SQL pools.

Next steps

- Use built-in Azure Policies to implement encryption protection for Synapse workspaces
- Create an Azure key vault and a key by using Resource Manager template

Azure Synapse Analytics IP firewall rules

8/19/2022 • 2 minutes to read • Edit Online

This article will explain IP firewall rules and teach you how to configure them in Azure Synapse Analytics.

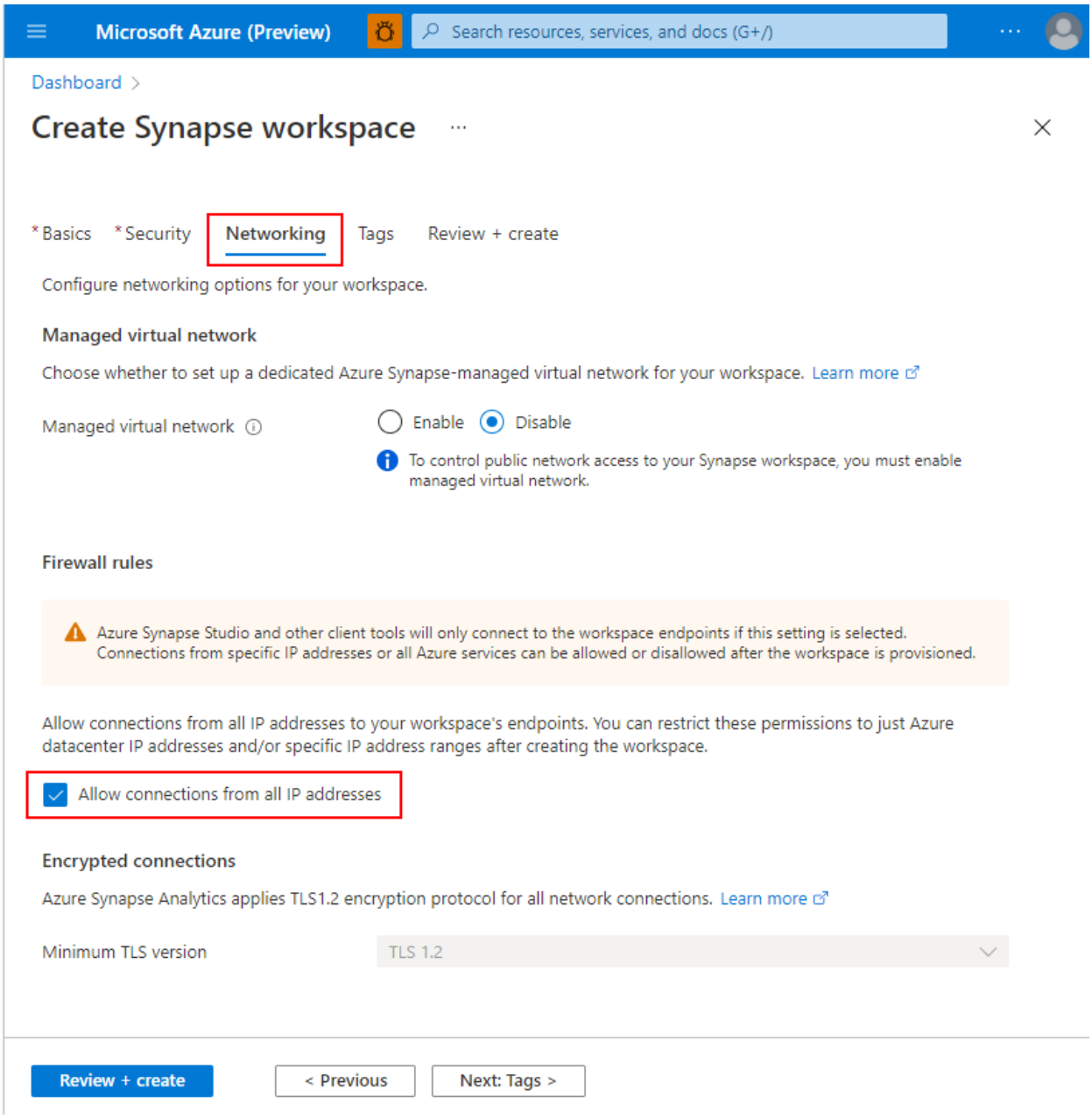
IP firewall rules

IP firewall rules grant or deny access to your Azure Synapse workspace based on the originating IP address of each request. You can configure IP firewall rules for your workspace. IP firewall rules configured at the workspace level apply to all public endpoints of the workspace (dedicated SQL pools, serverless SQL pool, and development).

Create and manage IP firewall rules

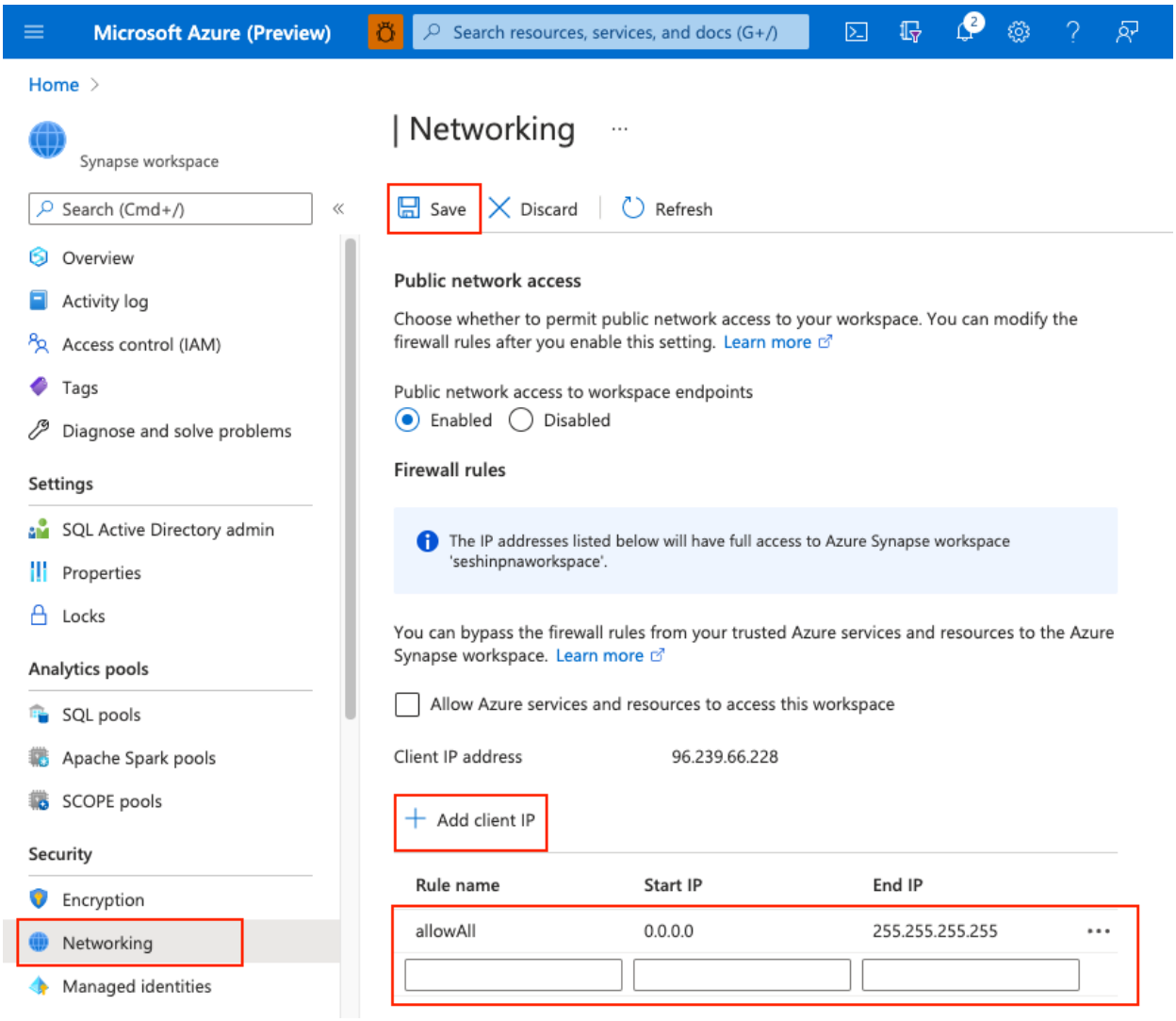
There are two ways IP firewall rules are added to an Azure Synapse workspace. To add an IP firewall to your workspace, select **Networking** and check **Allow connections from all IP addresses** during workspace creation.

> **IMPORTANT**
> This feature is only available to Azure Synapse workspaces not associated with a Managed VNet.



You can also add IP firewall rules to a Synapse workspace after the workspace is created. Select **Firewalls** under **Security** from Azure portal. To add a new IP firewall rule, give it a name, Start IP, and End IP. Select **Save** when done.

> **NOTE**
>
> The Public network access feature is only available to Azure Synapse workspaces associated with Azure Synapse Analytics Managed Virtual Network. However, you can still open your Azure Synapse workspaces to the public network regardless of its association with managed VNet. For more information, see Public network access.



Connect to Azure Synapse from your own network

You can connect to your Synapse workspace using Synapse Studio. You can also use SQL Server Management Studio (SSMS) to connect to the SQL resources (dedicated SQL pools and serverless SQL pool) in your workspace.

Make sure that the firewall on your network and local computer allows outgoing communication on TCP ports 80, 443 and 1443 for Synapse Studio.

Also, you need to allow outgoing communication on UDP port 53 for Synapse Studio. To connect using tools such as SSMS and Power BI, you must allow outgoing communication on TCP port 1433.

Manage the Azure Synapse workspace firewall

For more information on managing the firewall, see the Azure SQL documentation to manage server-level firewalls. Azure Synapse only supports server-level IP firewall rules. It doesn't support database-level IP firewall rules.

For more information on the methods to manage the firewall programmatically, see:

- API
- PowerShell
- Azure CLI

Next steps

- Create an Azure Synapse Workspace
- Create an Azure Synapse workspace with a Managed workspace Virtual Network
- Troubleshoot Azure Private Link connectivity problems
- Troubleshoot Azure Private Endpoint connectivity problems

Azure Synapse Analytics connectivity settings

8/19/2022 • 2 minutes to read • Edit Online

This article will explain connectivity settings in Azure Synapse Analytics and how to configure them where applicable.

Public network access

You can use the public network access feature to allow incoming public network connectivity to you Azure Synapse workspace.

- When public network access is **disabled**, you can connect to your workspace only using private endpoints.
- When public network access is **enabled**, you can connect to your workspace also from public networks. You can manage this feature both during and after your workspace creation.

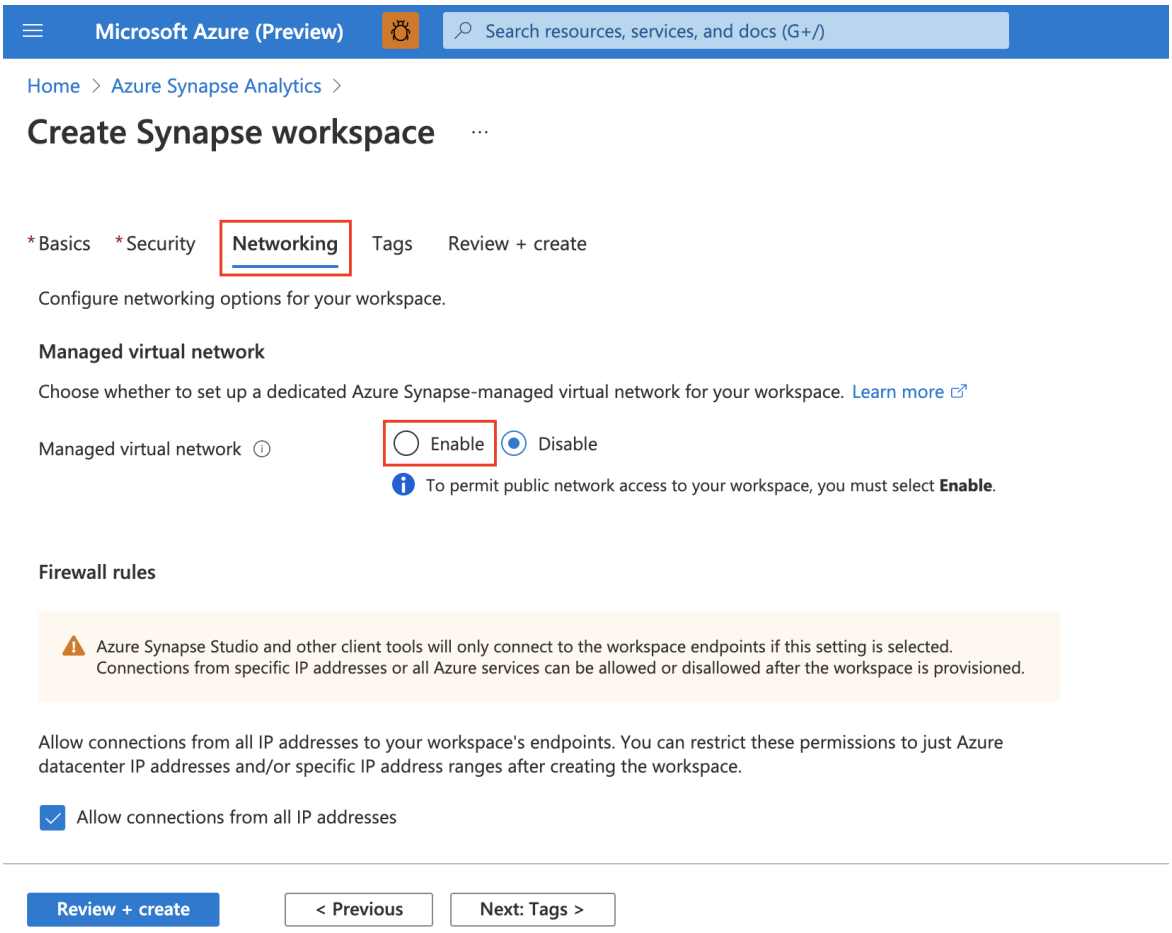
> **IMPORTANT**
> This feature is only available to Azure Synapse workspaces associated with Azure Synapse Analytics Managed Virtual Network. However, you can still open your Synapse workspaces to the public network regardless of its association with managed VNet.

Selecting the **Disable** option will not apply any firewall rules that you may configure. Additionally, your firewall rules will appear greyed out in the Network setting in Synapse portal. Your firewall configurations will be reapplied when you enable public network access again.

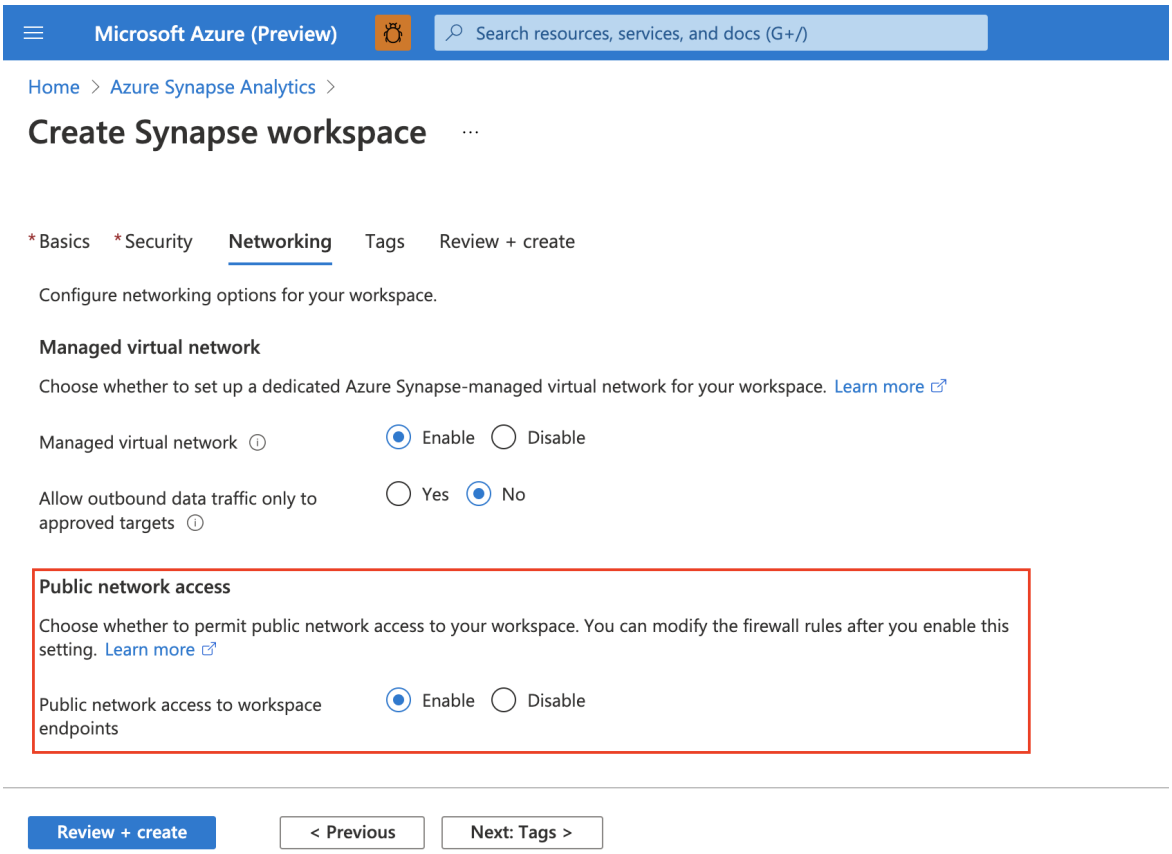
> **TIP**
> When you revert to enable, allow some time before editing the firewall rules.

Configure public network access when you create your workspace

1. Select the **Networking** tab when you create your workspace in Azure portal.
2. Under Managed virtual network, select **Enable** to associate your workspace with managed virtual network and permit public network access.



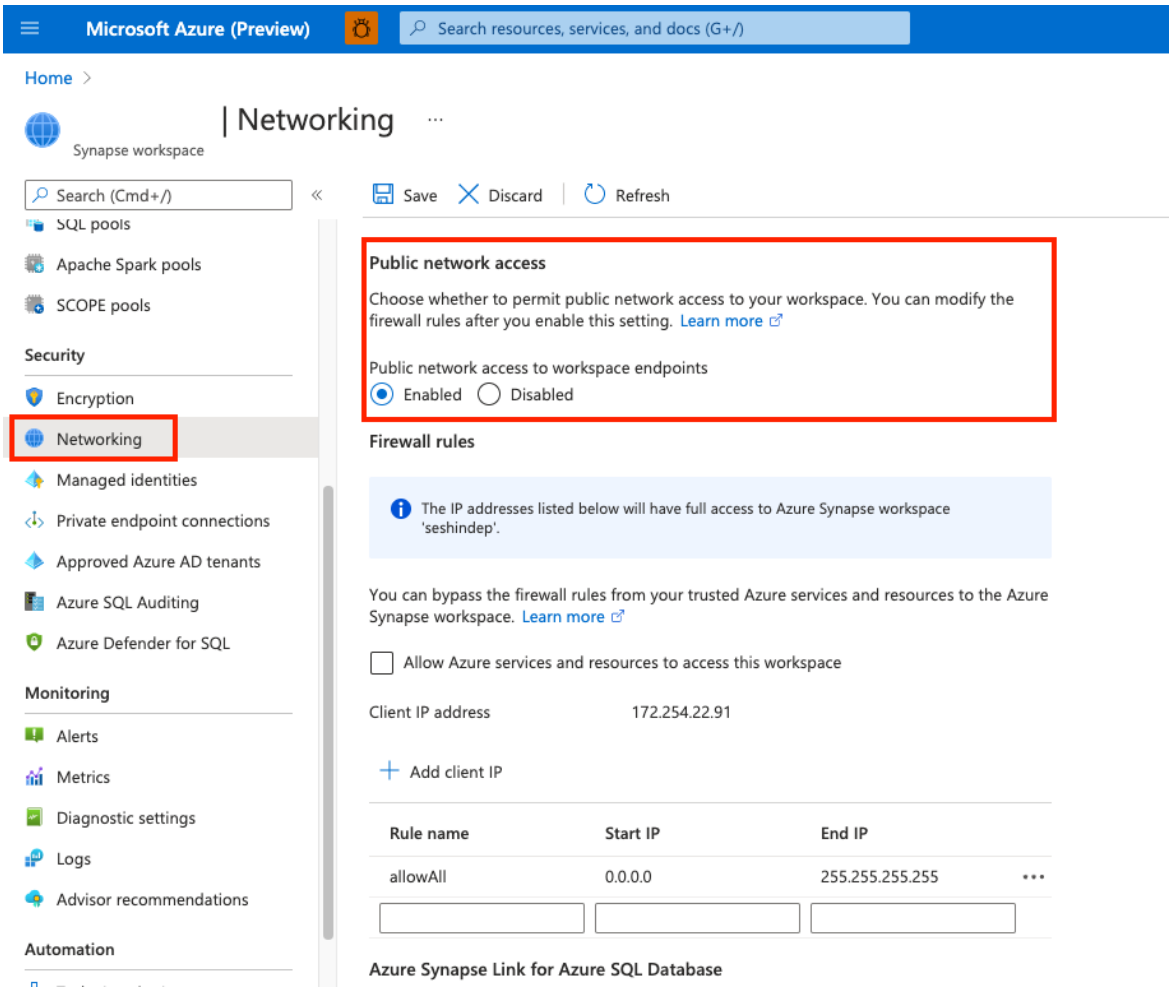
3. Under **Public network access**, select **Disable** to deny public access to your workspace. Select **Enable** if you want to allow public access to your workspace.



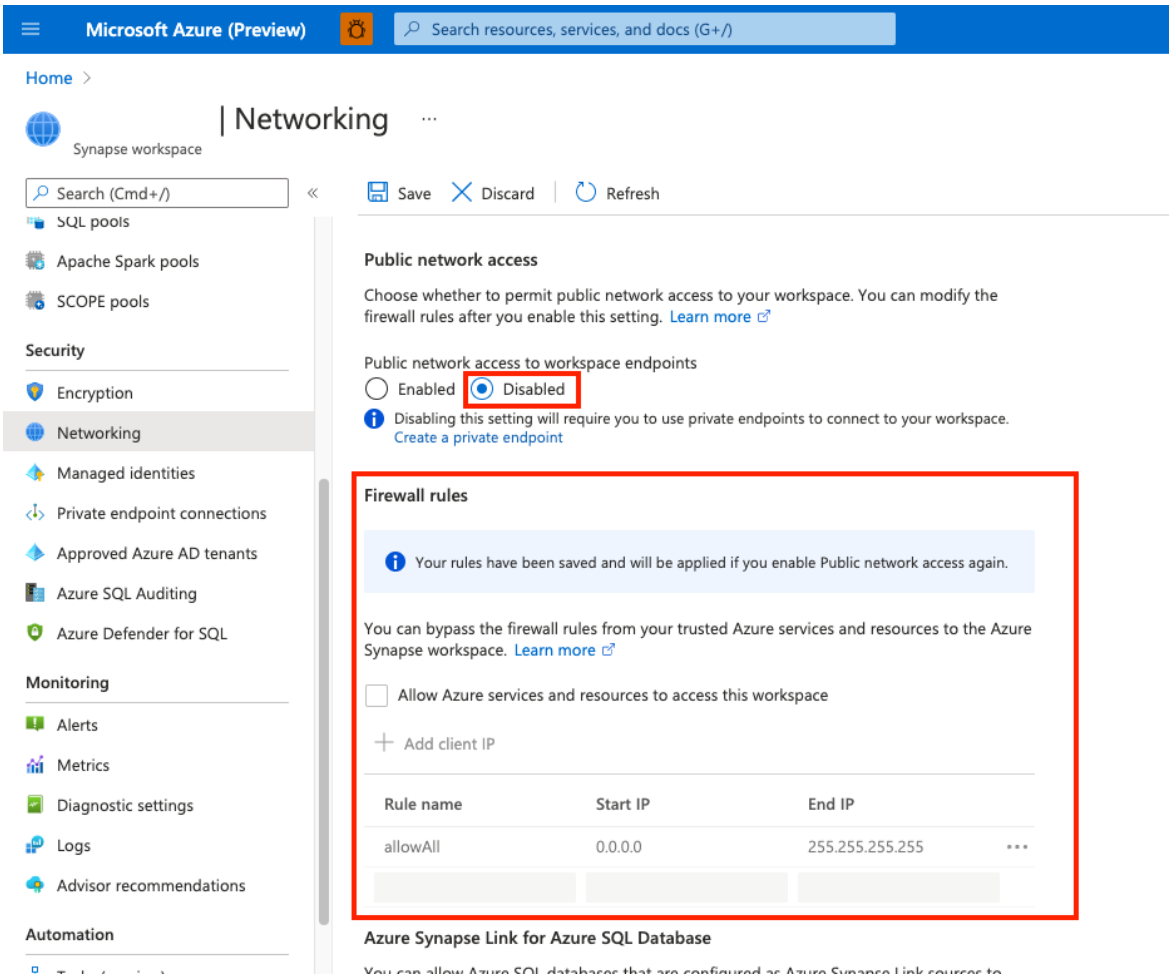
4. Complete the rest of the workspace creation flow.

Configure public network access after you create your workspace

1. Select your Synapse workspace in Azure portal.
2. Select **Networking** from the left navigation.
3. Select **Disabled** to deny public access to your workspace. Select **Enabled** if you want to allow public access to your workspace.



4. When disabled, the **Firewall rules** gray out to indicate that firewall rules are not in effect. Firewall rule configurations will be retained.



5. Select **Save** to save the change. A notification will confirm that the network setting was successfully saved.

Connection policy

The connection policy for Synapse SQL in Azure Synapse Analytics is set to *Default*. You cannot change this in Azure Synapse Analytics. You can learn more about how that affects connections to Synapse SQL in Azure Synapse Analytics here.

Minimal TLS version

The serverless SQL endpoint and development endpoint only accept TLS 1.2 and above.

Starting in December 2021, a requirement for TLS 1.2 has been implemented for workspace-managed dedicated SQL pools in new Synapse workspaces. Login attempts from connections using a TLS version lower than 1.2 will fail. Customers can raise or lower the minimal TLS version using the API, for both new Synapse workspaces or existing workspaces, so users who need to use a lower client version in the workspaces can connect. Customers can also raise the minimum TLS version to meet their security needs. Learn more by reading minimal TLS REST API.

Next steps

- Create an Azure Synapse Workspace.

Azure Synapse Analytics Managed Virtual Network

8/19/2022 • 3 minutes to read • Edit Online

This article will explain Managed Virtual Network in Azure Synapse Analytics.

Managed workspace Virtual Network

When you create your Azure Synapse workspace, you can choose to associate it to a Microsoft Azure Virtual Network. The Virtual Network associated with your workspace is managed by Azure Synapse. This Virtual Network is called a *Managed workspace Virtual Network*.

Managed workspace Virtual Network provides you value in four ways:

- With a Managed workspace Virtual Network you can offload the burden of managing the Virtual Network to Azure Synapse.
- You don't have to configure inbound NSG rules on your own Virtual Networks to allow Azure Synapse management traffic to enter your Virtual Network. Misconfiguration of these NSG rules causes service disruption for customers.
- You don't need to create a subnet for your Spark clusters based on peak load.
- Managed workspace Virtual Network along with Managed private endpoints protects against data exfiltration. You can only create Managed private endpoints in a workspace that has a Managed workspace Virtual Network associated with it.

Creating a workspace with a Managed workspace Virtual Network associated with it ensures that your workspace is network isolated from other workspaces. Azure Synapse provides various analytic capabilities in a workspace: Data integration,serverless Apache Spark pool, dedicated SQL pool, and serverless SQL pool.

If your workspace has a Managed workspace Virtual Network, Data integration and Spark resources are deployed in it. A Managed workspace Virtual Network also provides user-level isolation for Spark activities because each Spark cluster is in its own subnet.

Dedicated SQL pool and serverless SQL pool are multi-tenant capabilities and therefore reside outside of the Managed workspace Virtual Network. Intra-workspace communication to dedicated SQL pool and serverless SQL pool use Azure private links. These private links are automatically created for you when you create a workspace with a Managed workspace Virtual Network associated to it.

> **IMPORTANT**
> You cannot change this workspace configuration after the workspace is created. For example, you cannot reconfigure a workspace that does not have a Managed workspace Virtual Network associated with it and associate a Virtual Network to it. Similarly, you cannot reconfigure a workspace with a Managed workspace Virtual Network associated to it and disassociate the Virtual Network from it.

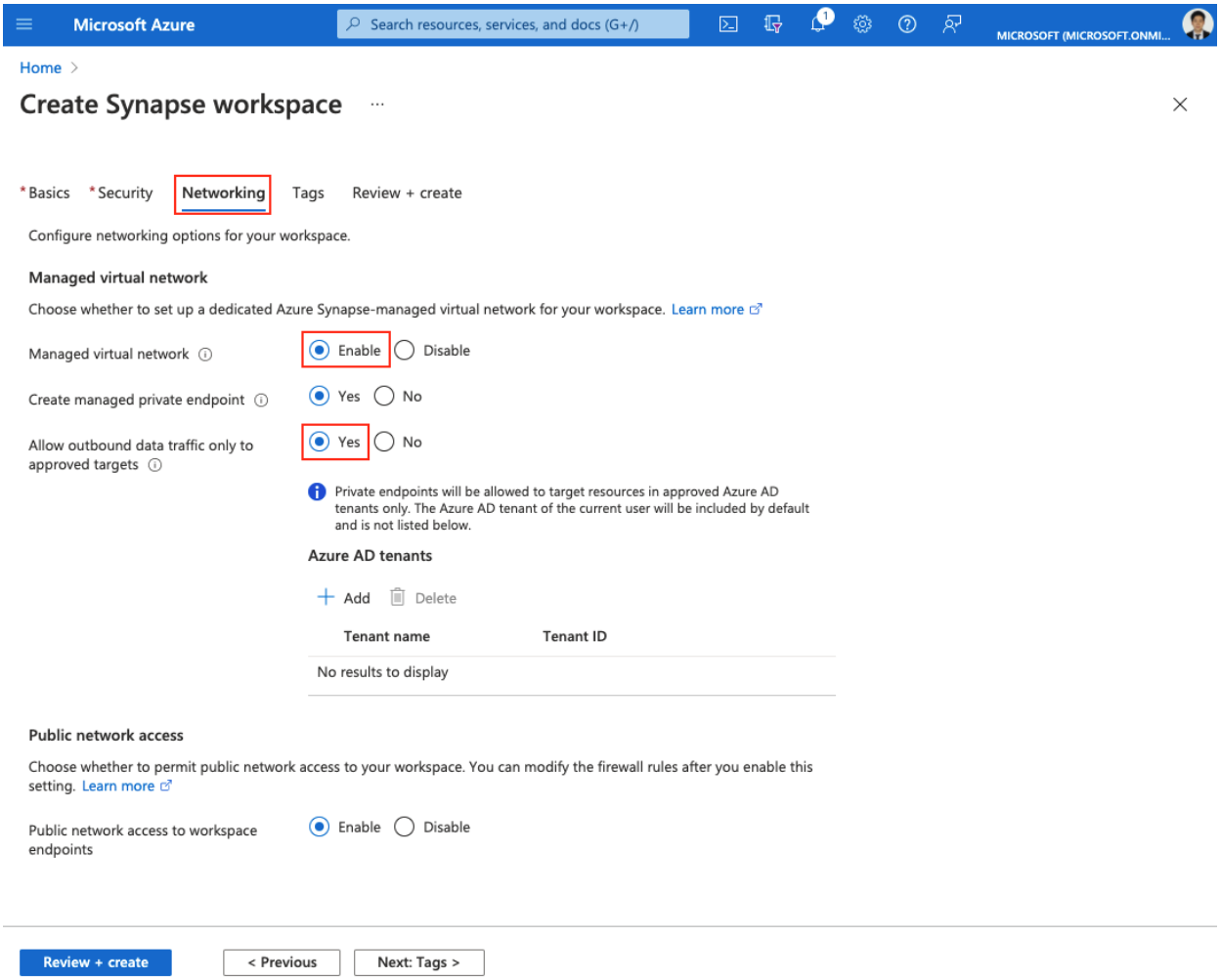
Create an Azure Synapse workspace with a Managed workspace Virtual Network

If you have not already done so, register the Network resource provider. Registering a resource provider configures your subscription to work with the resource provider. Choose *Microsoft.Network* from the list of resource providers when you register.

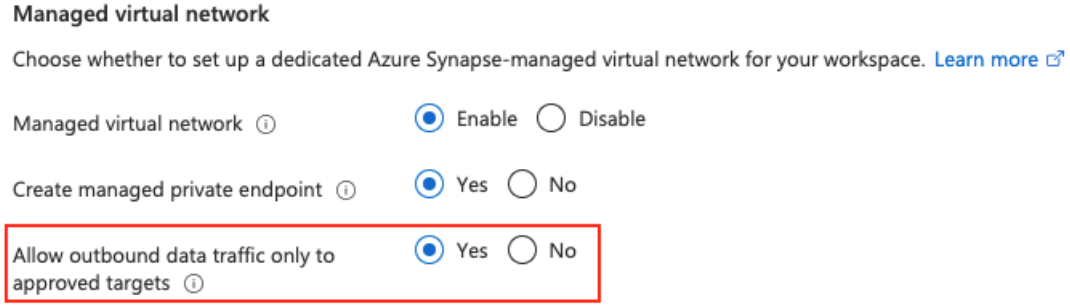
To create an Azure Synapse workspace that has a Managed workspace Virtual Network associated with it, select the **Networking** tab in Azure portal and check the **Enable managed virtual network** checkbox.

If you leave the checkbox unchecked, then your workspace won't have a Virtual Network associated with it.

> **IMPORTANT**
> You can only use private links in a workspace that has a Managed workspace Virtual Network.



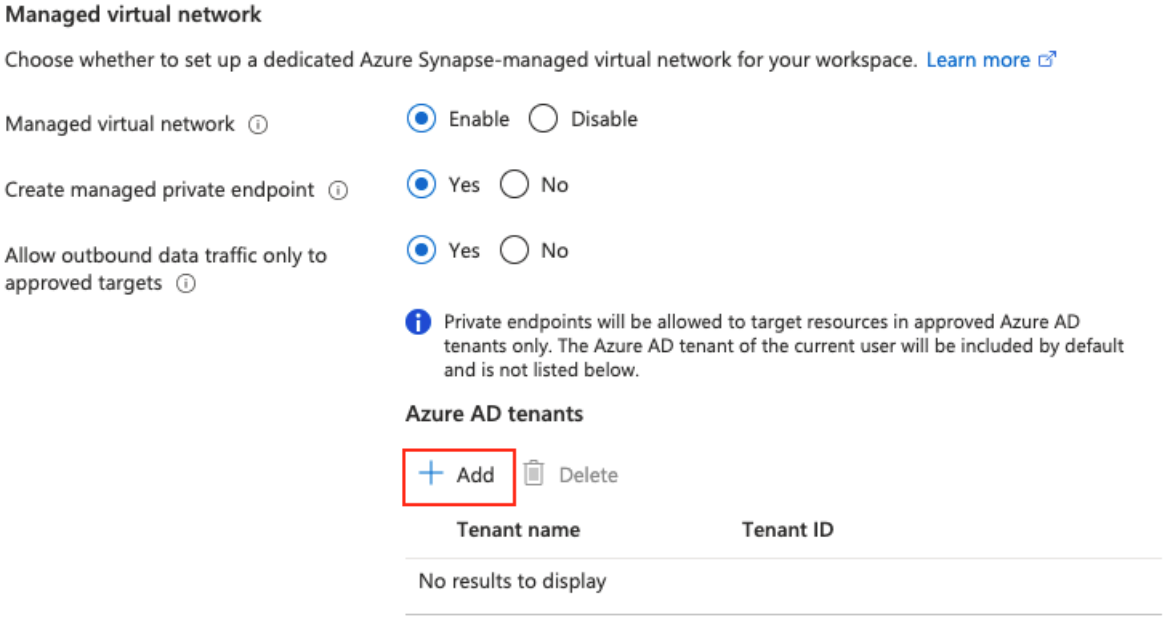
After you choose to associate a Managed workspace Virtual Network with your workspace, you can protect against data exfiltration by allowing outbound connectivity from the Managed workspace Virtual Network only to approved targets using Managed private endpoints. Select **Yes** to limit outbound traffic from the Managed workspace Virtual Network to targets through Managed private endpoints.



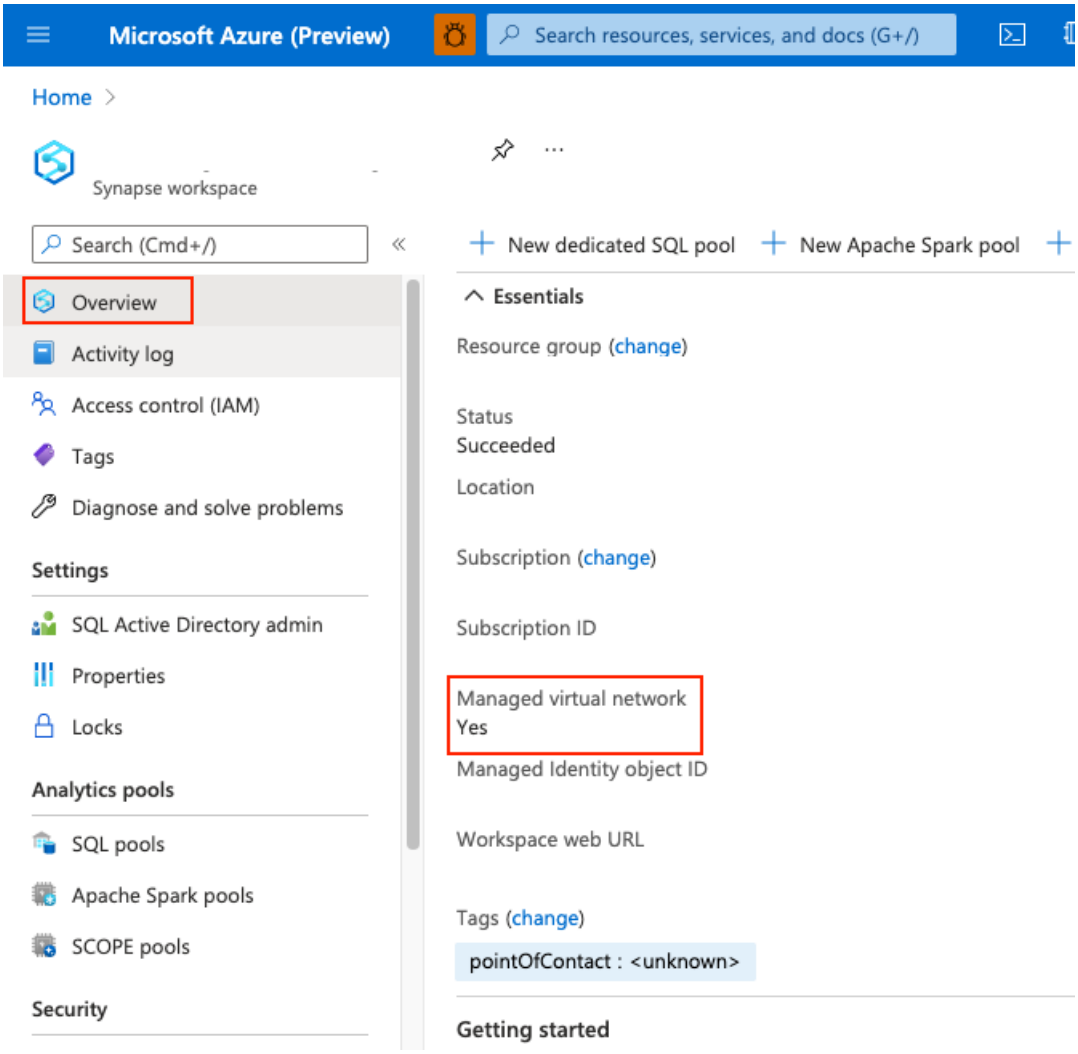
Select **No** to allow outbound traffic from the workspace to any target.

You can also control the targets to which Managed private endpoints are created from your Azure Synapse workspace. By default, Managed private endpoints to resources in the same AAD tenant that your subscription

belongs to are allowed. If you want to create a Managed private endpoint to a resource in an AAD tenant that is different from the one that your subscription belongs to, then you can add that AAD tenant by selecting **+ Add**. You can either select the AAD tenant from the dropdown or manually enter the AAD tenant ID.



After the workspace is created, you can check whether your Azure Synapse workspace is associated to a Managed workspace Virtual Network by selecting **Overview** from Azure portal.



Next steps

Create an Azure Synapse Workspace

Learn more about Managed private endpoints

Create Managed private endpoints to your data sources

Synapse Managed private endpoints

8/19/2022 • 2 minutes to read • Edit Online

This article will explain Managed private endpoints in Azure Synapse Analytics.

Managed private endpoints

Managed private endpoints are private endpoints created in a Managed Virtual Network associated with your Azure Synapse workspace. Managed private endpoints establish a private link to Azure resources. Azure Synapse manages these private endpoints on your behalf. You can create Managed private endpoints from your Azure Synapse workspace to access Azure services (such as Azure Storage or Azure Cosmos DB) and Azure hosted customer/partner services.

When you use Managed private endpoints, traffic between your Azure Synapse workspace and other Azure resources traverse entirely over the Microsoft backbone network. Managed private endpoints protect against data exfiltration. A Managed private endpoint uses private IP address from your Managed Virtual Network to effectively bring the Azure service that your Azure Synapse workspace is communicating into your Virtual Network. Managed private endpoints are mapped to a specific resource in Azure and not the entire service. Customers can limit connectivity to a specific resource approved by their organization.

Learn more about private links and private endpoints.

> **IMPORTANT**
>
> Managed private endpoints are only supported in Azure Synapse workspaces with a Managed workspace Virtual Network.

> **NOTE**
>
> When creating an Azure Synapse workspace, you can choose to associate a Managed Virtual Network to it. If you choose to have a Managed Virtual Network associated to your workspace, you can also choose to limit outbound traffic from your workspace to only approved targets. You must create Managed private endpoints to these targets.

A private endpoint connection is created in a "Pending" state when you create a Managed private endpoint in Azure Synapse. An approval workflow is started. The private link resource owner is responsible to approve or reject the connection. If the owner approves the connection, the private link is established. But, if the owner doesn't approve the connection, then the private link won't be established. In either case, the Managed private endpoint will be updated with the status of the connection. Only a Managed private endpoint in an approved state can be used to send traffic to the private link resource that is linked to the Managed private endpoint.

Managed private endpoints for dedicated SQL pool and serverless SQL pool

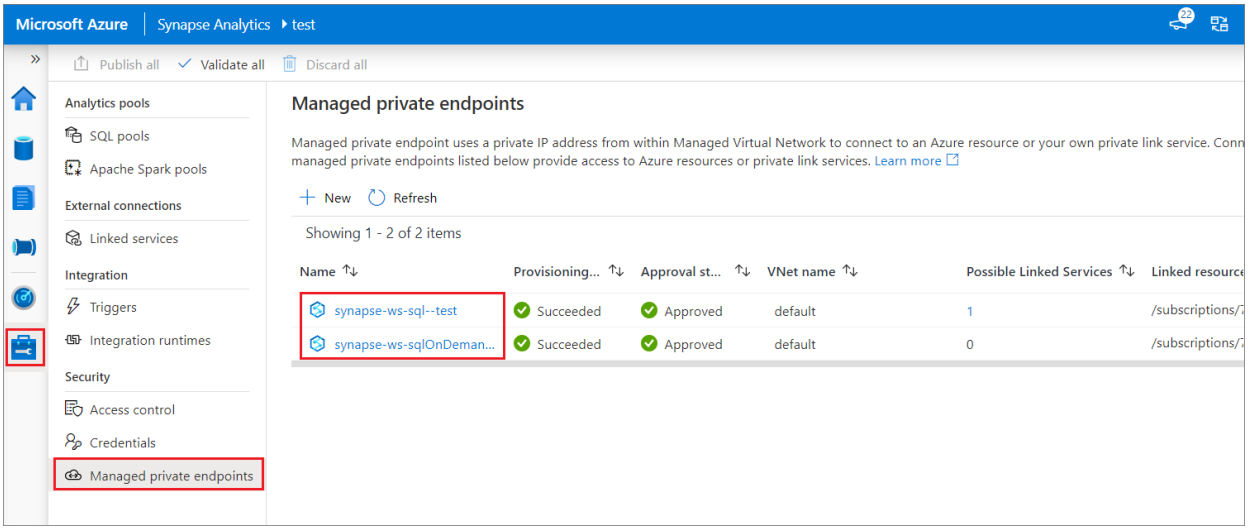
Dedicated SQL pool and serverless SQL pool are analytic capabilities in your Azure Synapse workspace. These capabilities use multi-tenant infrastructure that isn't deployed into the Managed workspace Virtual Network.

When a workspace is created, Azure Synapse creates two Managed private endpoints in the workspace, one for dedicated SQL pool and one for serverless SQL pool.

These two Managed private endpoints are listed in Synapse Studio. Select **Manage** in the left navigation, then

select **Managed private endpoints** to see them in the Studio.

The Managed private endpoint that targets SQL pool is called *synapse-ws-sql--<workspacename>* and the one that targets serverless SQL pool is called *synapse-ws-sqlOnDemand--<workspacename>*.



These two Managed private endpoints are automatically created for you when you create your Azure Synapse workspace. You aren't charged for these two Managed private endpoints.

Next steps

To learn more, advance to the Create Managed private endpoints to your data sources article.

Data exfiltration protection for Azure Synapse Analytics workspaces

8/19/2022 • 2 minutes to read • Edit Online

This article will explain data exfiltration protection in Azure Synapse Analytics

Securing data egress from Synapse workspaces

Azure Synapse Analytics workspaces support enabling data exfiltration protection for workspaces. With exfiltration protection, you can guard against malicious insiders accessing your Azure resources and exfiltrating sensitive data to locations outside of your organization's scope. At the time of workspace creation, you can choose to configure the workspace with a managed virtual network and additional protection against data exfiltration. When a workspace is created with a managed virtual network, Data integration and Spark resources are deployed in the managed virtual network. The workspace's dedicated SQL pools and serverless SQL pools have multi-tenant capabilities and as such, need to exist outside the managed virtual network. For workspaces with data exfiltration protection, resources within the managed virtual network always communicate over managed private endpoints and the Synapse SQL resources can only connect to authorized Azure resources (targets of approved managed private endpoint connections from the workspace).

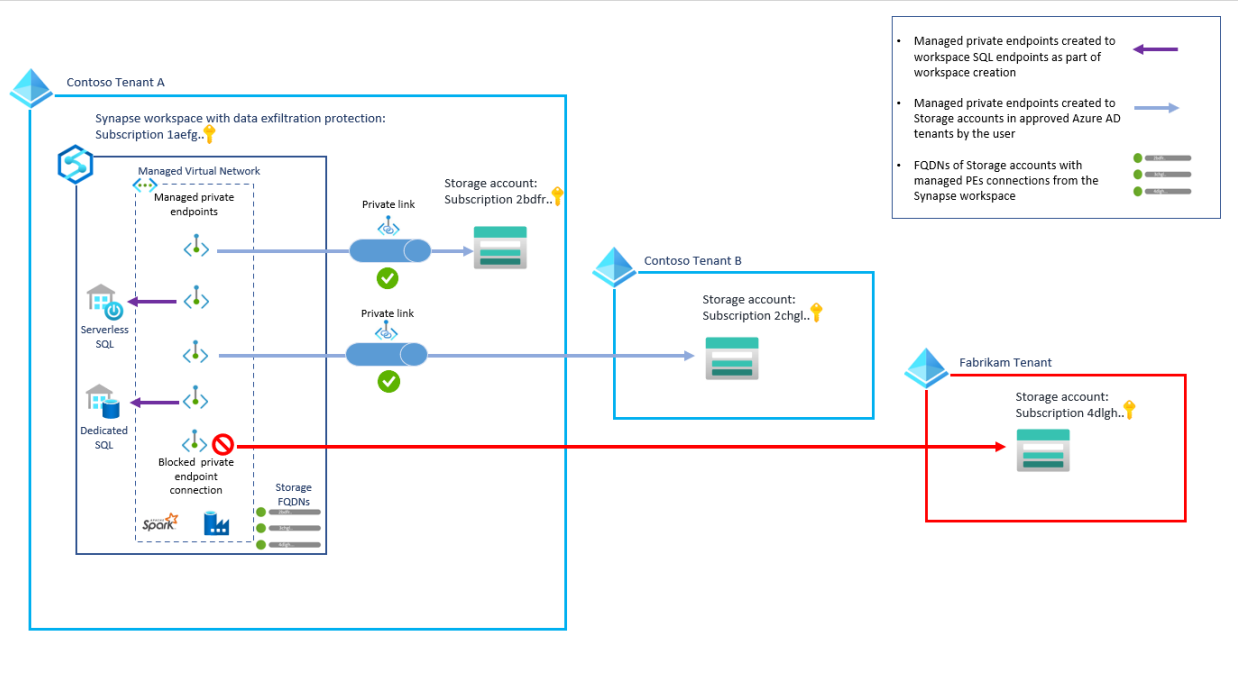
> **NOTE**
> You cannot change the workspace configuration for managed virtual network and data exfiltration protection after the workspace is created.

Managing Synapse workspace data egress to approved targets

After the workspace is created with data exfiltration protection enabled, the owners of the workspace resource can manage the list of approved Azure AD tenants for the workspace. Users with the right permissions on the workspace can use the Synapse Studio to create managed private endpoint connection requests to resources in the workspace's approved Azure AD tenants. Managed private endpoint creation will be blocked if the user attempts to create a private endpoint connection to a resource in an unapproved tenant.

Sample workspace with data exfiltration protection enabled

Let us use an example to illustrate data exfiltration protection for Synapse workspaces. Contoso has Azure resources in Tenant A and Tenant B and there is a need for these resources to connect securely. A Synapse workspace has been created in Tenant A with Tenant B added as an approved Azure AD tenant. The diagram shows private endpoint connections to Azure Storage accounts in Tenant A and Tenant B that have been approved by the Storage account owners. The diagram also shows blocked private endpoint creation. The creation of this private endpoint was blocked as it targeted an Azure Storage account in the Fabrikam Azure AD tenant, which is not an approved Azure AD tenant for Contoso's workspace.



IMPORTANT

Resources in tenants other than the workspace's tenant must not have blocking firewall rules in place for the SQL pools to connect to them. Resources within the workspace's managed virtual network, such as Spark clusters, can connect over managed private links to firewall-protected resources.

Next Steps

Learn how to create a workspace with data exfiltration protection enabled

Learn more about Managed workspace Virtual Network

Learn more about Managed private endpoints

Create Managed private endpoints to your data sources

Connect to Azure Synapse Studio using Azure Private Link Hubs

8/19/2022 • 2 minutes to read • Edit Online

This article explains how Azure Synapse Analytics Private Link Hubs private links are used to securely connect to Synapse Studio.

Azure Private Link Hubs

You can securely connect to Azure Synapse Studio from your Azure virtual network using private links. Azure Synapse Analytics private link hubs are Azure resources which act as connectors between your secured network and the Synapse Studio web experience.

There are two steps to connect to Synapse Studio using private links. First, you must create a private link hubs resource. Second, you must create a private endpoint from your Azure virtual network to this private link hub. You can then use private endpoints to securely communicate with Synapse Studio. You must integrate the private endpoints with your DNS solution, either your on-premises solution or Azure Private DNS.

Azure Private Links Hubs and Azure Synapse Studio

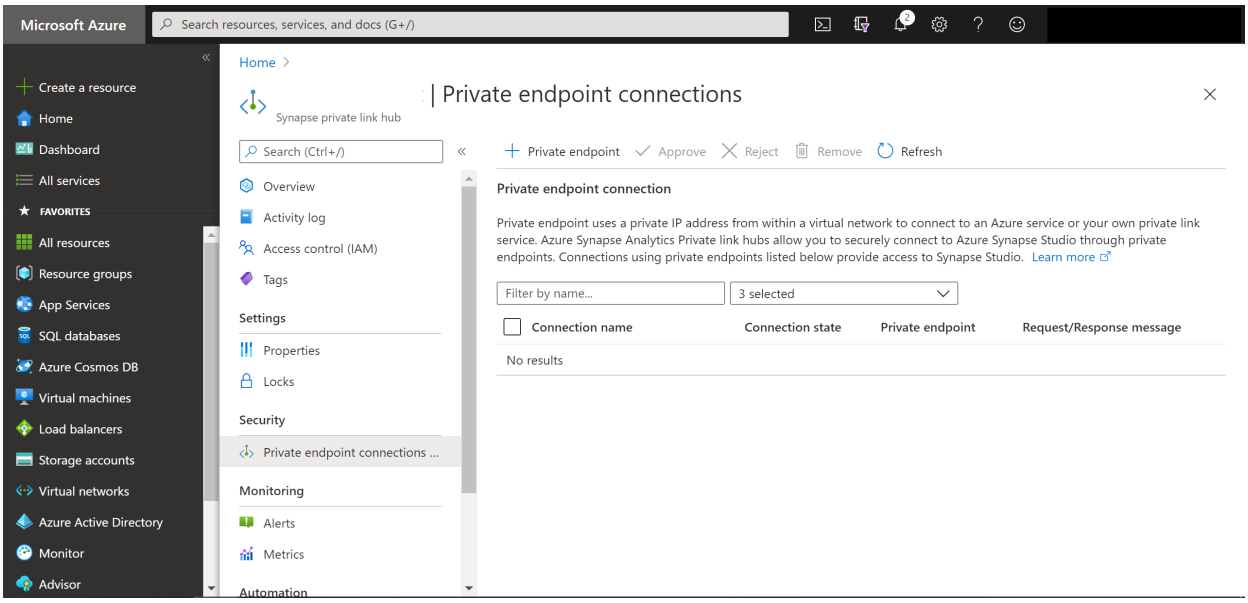
You can use a single Azure Synapse private link hub resource to privately connect to all your Azure Synapse Analytics workspaces using Azure Synapse Studio. The workspaces do not have to be in the same region as the Azure Synapse Private link hub. The Azure Synapse Private link hub resource can also be used for connections to Synapse workspaces in different subscriptions or Azure AD tenants.

You can create your private link hub by searching for *Synapse private link hubs* in the Azure portal and selecting **Azure Synapse Analytics (private link hubs)** from Services. Follow the steps in the guide for how to connect to workspace resources from a restricted network for details.

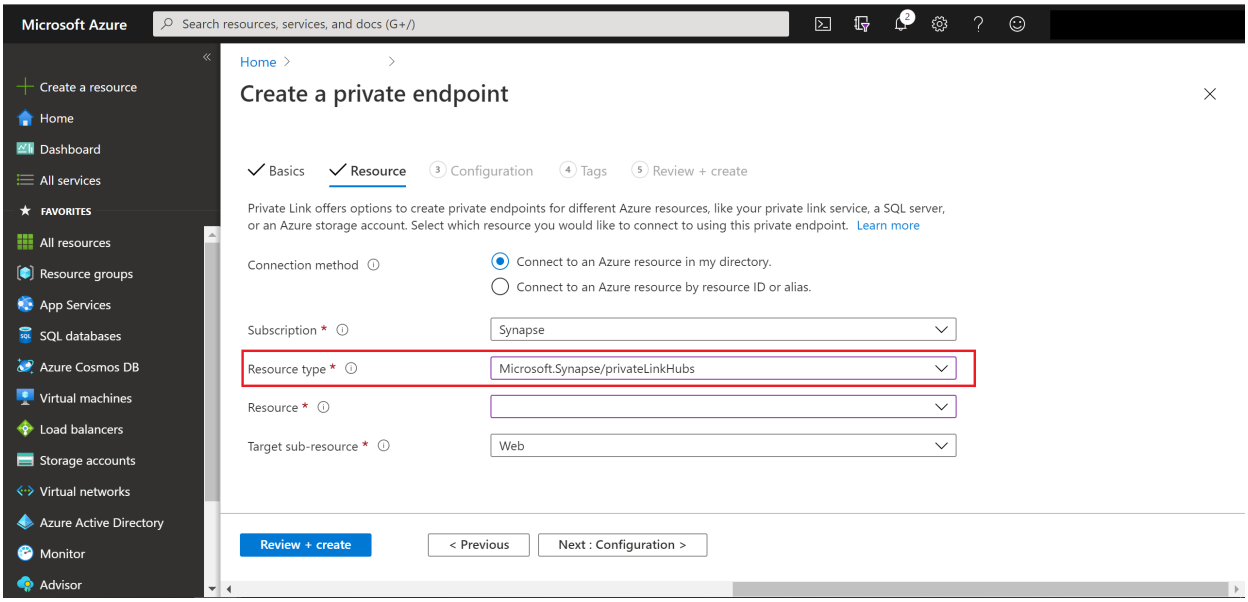
> **NOTE**
> Private link hubs are intended for securely loading the static content Synapse Studio over private links. You must create **separate, private endpoints** to the resources you wish to connect to within the workspace, such as provisioned/dedicated SQL pools, or Spark pools.

Azure Private Links Hubs and Azure Virtual Network

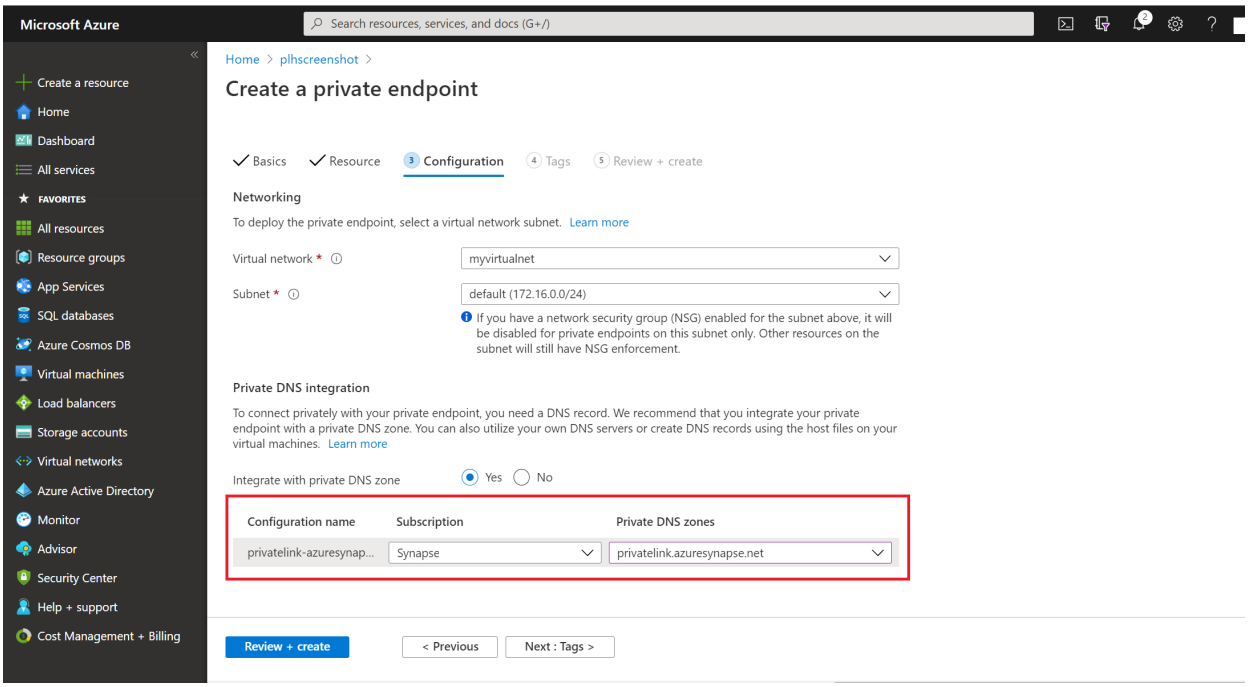
You must connect your Azure virtual network to the Synapse private link hub resource to secure the end-to-end connection to Synapse Studio. For this, you must create a private endpoint from your virtual network to the private link hub you created. You can use the Azure portal for your private link hub and go to the Private endpoint section. Select "+ Private endpoint" to create a new private endpoint that connects to your private link hub.



Make sure to choose the "Microsoft.Synapse/privateLinkHubs" Resource type on the "Resource" tab.



On "Configuration" tab, select "privatelink.azuresynapse.net" for Private DNS Zones when integrating with your virtual network and private DNS zone.



Next steps

Connect to workspace resources from a restricted network

Learn more about Managed workspace Virtual Network

Learn more about Managed private endpoints

Create Managed private endpoints to your data sources

Connect to Synapse workspace using private endpoints

Connect to a secure Azure storage account from your Synapse workspace

8/19/2022 • 3 minutes to read • Edit Online

This article will teach you how to connect to a secure Azure storage account from your Azure Synapse workspace. You can link an Azure storage account to your Synapse workspace when you create your workspace. You can link more storage accounts after you create your workspace.

Secured Azure storage accounts

Azure storage provides a layered security model that enables you to secure and control access to your storage accounts. You can configure IP firewall rules to grant traffic from selected public IP address ranges access to your storage account. You can also configure network rules to grant traffic from selected virtual networks access to your storage account. You can combine IP firewall rules that allow access from selected IP address ranges and network rules that grant access from selected virtual networks on the same storage account. These rules apply to the public endpoint of a storage account. You do not need any access rules to allow traffic from Managed private endpoints created in your workspace to a storage account. Storage firewall rules can be applied to existing storage accounts, or to new storage accounts when you create them. You can learn more about securing your storage account here.

Synapse workspaces and virtual networks

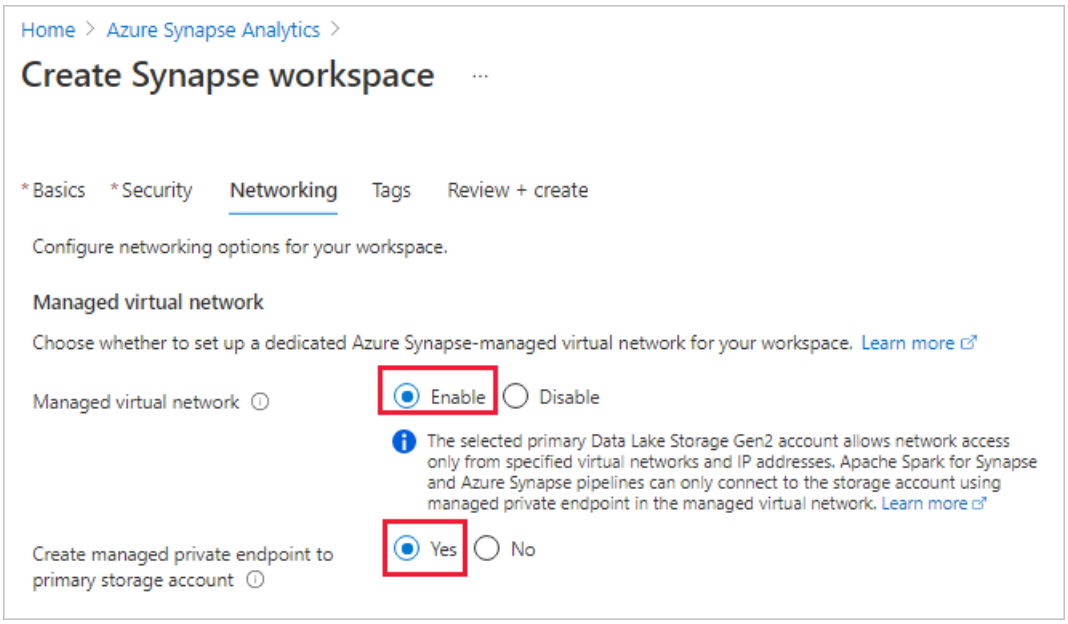
When you create a Synapse workspace, you can choose to enable a Managed virtual network to be associated with it. If you do not enable Managed virtual network for your workspace when you create it, your workspace is in a shared virtual network along with other Synapse workspaces that do not have a Managed virtual network associated with it. If you enabled Managed virtual network when you created the workspace, then your workspace is associated with a dedicated virtual network managed by Azure Synapse. These virtual networks are not created in your customer subscription. Therefore, you will not be able to grant traffic from these virtual networks access to your secured storage account using network rules described above.

Access a secured storage account

Synapse operates from networks that cannot be included in your network rules. The following needs to be done to enable access from your workspace to your secure storage account.

- Create an Azure Synapse workspace with a managed virtual network associated with it and create managed private endpoints from it to the secure storage account.

  If you use Azure Portal to create your workspace, you can enable managed virtual network under the **Networking** tab as shown below. If you enable managed virtual network or Synapse determines that the primary storage account is a secure storage account, then you have the option to create a managed private endpoint connection request to the secure storage account as shown below. The storage account owner will need to approve the connection request to establish the private link. Alternatively, Synapse will approve this connection request if the user creating an Apache Spark pool in the workspace has sufficient privileges to approve the connection request.



- Grant your Azure Synapse workspace access to your secure storage account as a trusted Azure service. As a trusted service, Azure Synapse will then use strong authentication to securely connect to your storage account.

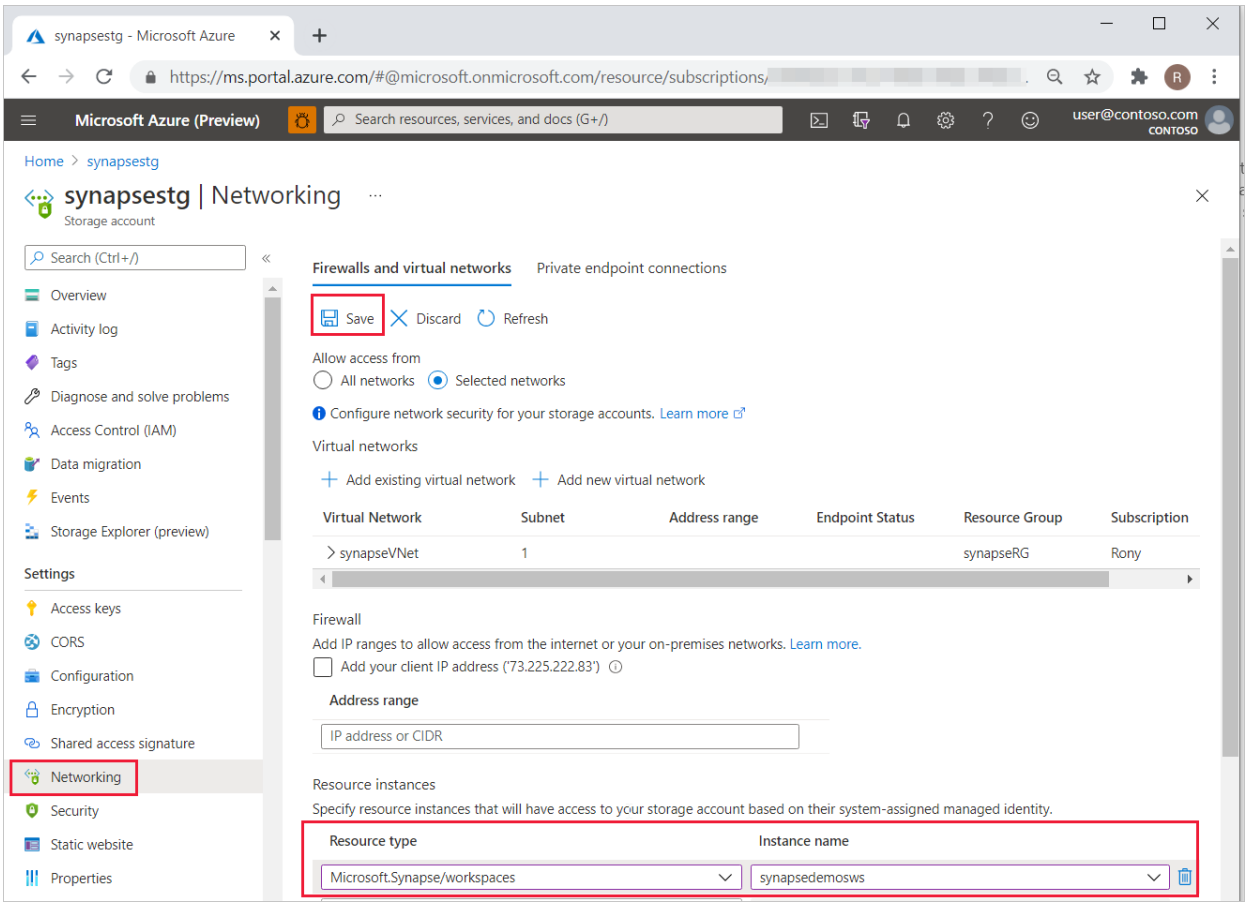
Create a Synapse workspace with a managed virtual network and create managed private endpoints to your storage account

You can follow these steps to create a Synapse workspace that has a managed virtual network associated with it. Once the workspace with an associated managed virtual network is created, you can create a managed private endpoint to your secure storage account by following the steps listed here.

Grant your Azure Synapse workspace access to your secure storage account as a trusted Azure service

Analytic capabilities such as Dedicated SQL pool and Serverless SQL pool use multi-tenant infrastructure that is not deployed into the managed virtual network. In order for traffic from these capabilities to access the secured storage account, you must configure access to your storage account based on the workspace's system-assigned managed identity by following the steps below.

In Azure portal, navigate to your secured storage account. Select **Networking** from the left navigation pane. In the **Resource instances** section, select *Microsoft.Synapse/workspaces* as the **Resource type** and enter your workspace name for **Instance name**. Select **Save**.



You should now be able to access your secured storage account from the workspace.

Next steps

Learn more about Managed workspace virtual network.

Learn more about Managed private endpoints.

Use Azure Active Directory Authentication for authentication with Synapse SQL

8/19/2022 • 7 minutes to read • Edit Online

Azure Active Directory authentication is a mechanism that connects to Azure Synapse Analytics by using identities in Azure Active Directory (Azure AD).

With Azure AD authentication, you can centrally manage user identities that have access to Azure Synapse to simplify permission management. Benefits include the following:

- It provides an alternative to regular username and password authentication.
- Helps stop the proliferation of user identities across servers.
- Allows password rotation in a single place.
- Customers can manage permissions using external (Azure AD) groups.
- It can eliminate storing passwords by enabling integrated Windows authentication and other forms of authentication supported by Azure Active Directory.
- Azure AD supports token-based authentication for applications connecting to Azure Synapse.
- Azure AD authentication supports ADFS (domain federation) or native user/password authentication for a local Azure Active Directory without domain synchronization.
- Azure AD supports connections from SQL Server Management Studio that use Active Directory Universal Authentication, which includes multi-factor authentication (MFA). MFA includes strong authentication with a range of easy verification options, including phone call, text message, smart cards with pin, or mobile app notification. For more information, see SSMS support for Azure AD MFA with Synapse SQL.
- Azure AD supports similar connections from SQL Server Data Tools (SSDT) that use Active Directory Interactive Authentication. For more information, see Azure Active Directory support in SQL Server Data Tools (SSDT).

The configuration steps include the following procedures to configure and use Azure Active Directory authentication.

1. Create and populate Azure AD.
2. Create an Azure Active Directory identity
3. Assign role to created Azure Active Directory identity in Synapse workspace
4. Connect to Synapse Studio by using Azure AD identities.

Azure AD pass-through in Azure Synapse Analytics

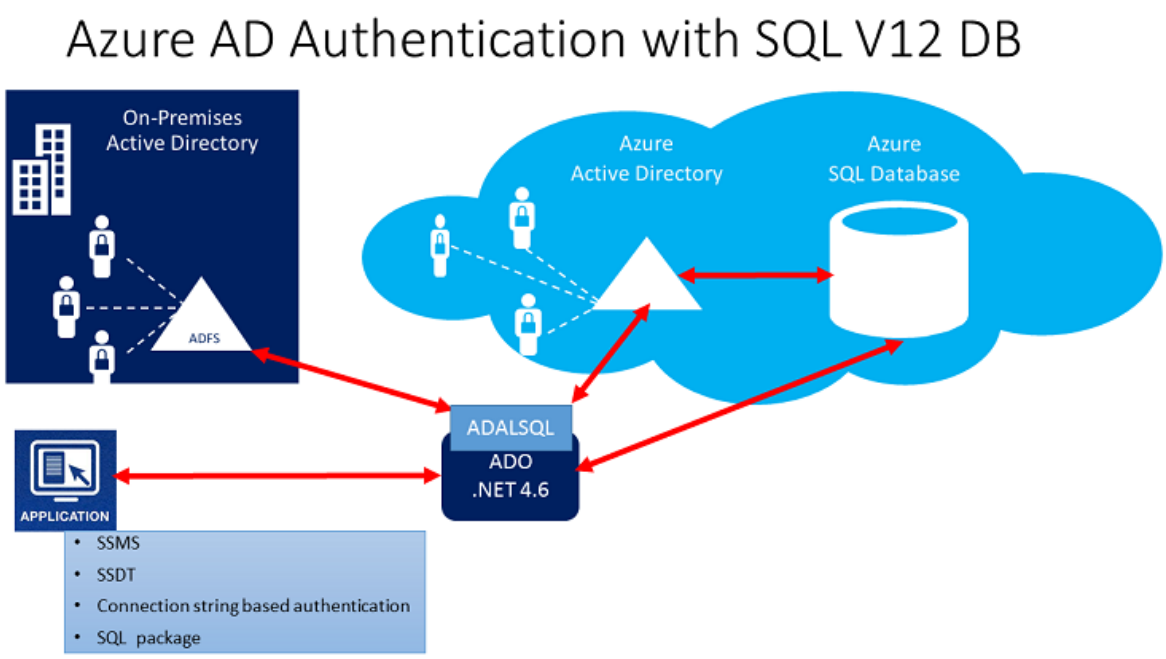
Azure Synapse Analytics enables you to access the data in the data lake using your Azure Active Directory identity.

Defining access rights on the files and data that is respected in different data engines enables you to simplify your data lake solutions by having a single place where the permissions are defined instead of having to define them in multiple places.

Trust architecture

The following high-level diagram summarizes the solution architecture of using Azure AD authentication with Synapse SQL. To support Azure AD native user password, only the Cloud portion and Azure AD/Synapse Synapse SQL is considered. To support Federated authentication (or user/password for Windows credentials),

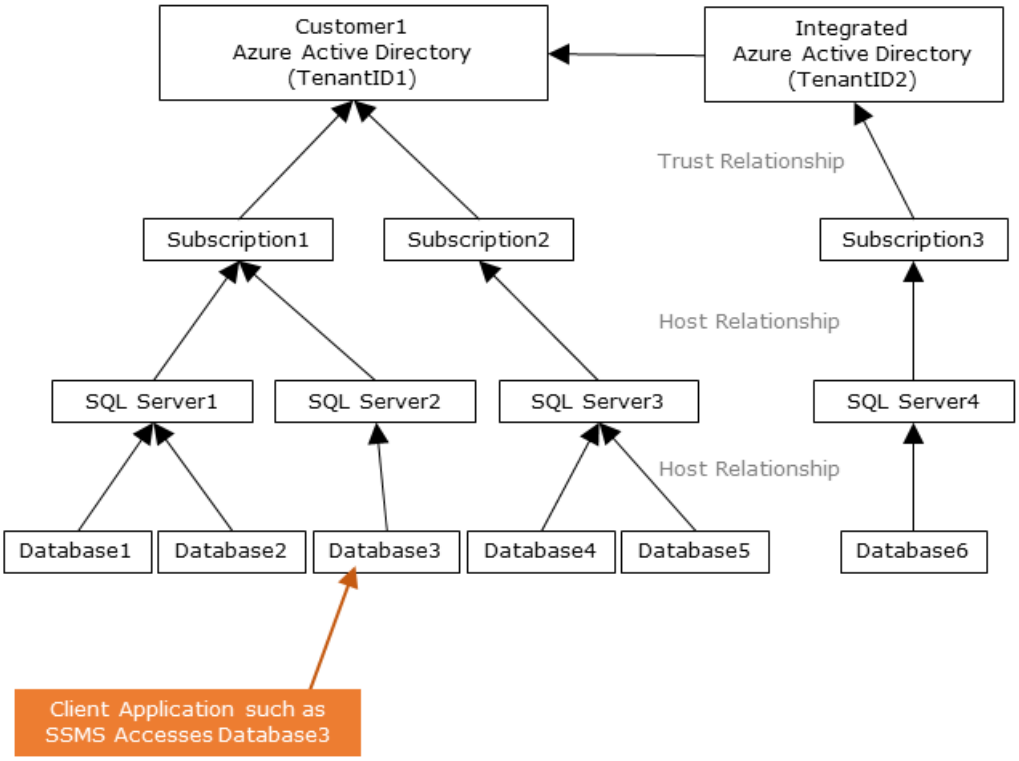
the communication with ADFS block is required. The arrows indicate communication pathways.



The following diagram indicates the federation, trust, and hosting relationships that allow a client to connect to a database by submitting a token. The token is authenticated by an Azure AD, and is trusted by the database.

Customer 1 can represent an Azure Active Directory with native users or an Azure AD with federated users. Customer 2 represents a possible solution including imported users; in this example coming from a federated Azure Active Directory with ADFS being synchronized with Azure Active Directory.

It's important to understand that access to a database using Azure AD authentication requires that the hosting subscription is associated to the Azure AD. The same subscription must be used to create the SQL Server hosting the Azure SQL Database or dedicated SQL pool.

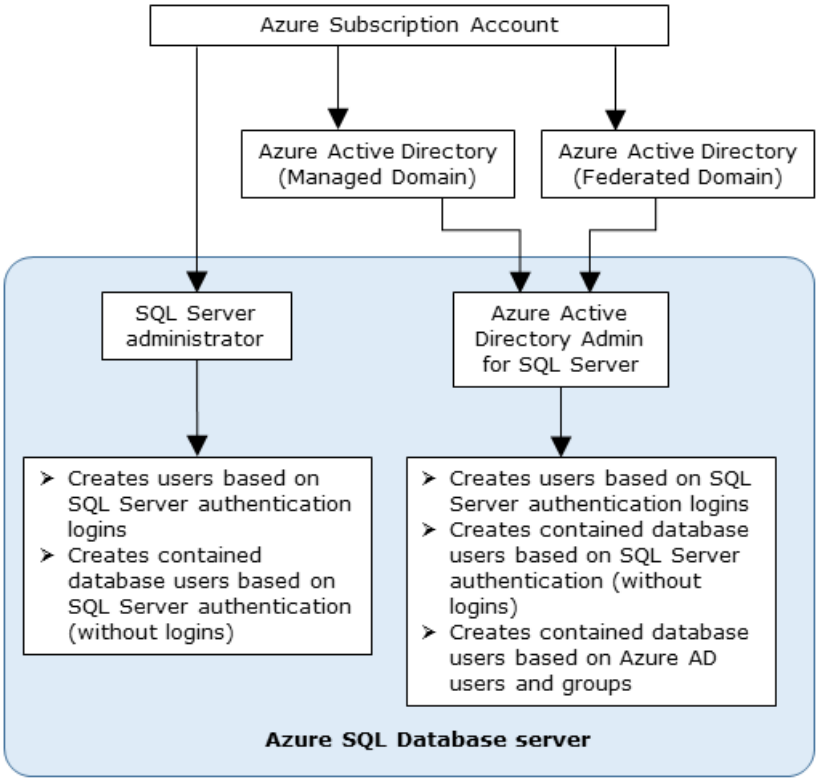


Administrator structure

When using Azure AD authentication, there are two Administrator accounts for the Synapse SQL; the original SQL administrator (using SQL authentication) and the Azure AD administrator. Only the administrator based on an Azure AD account can create the first Azure AD contained database user in a user database.

The Azure AD administrator login can be an Azure AD user or an Azure AD group. When the administrator is a group account, it can be used by any group member, enabling multiple Azure AD administrators for the Synapse SQL instance.

Using group account as an administrator enhances manageability by allowing you to centrally add and remove group members in Azure AD without changing the users or permissions in Azure Synapse Analytics workspace. Only one Azure AD administrator (a user or group) can be configured at any time.



Permissions

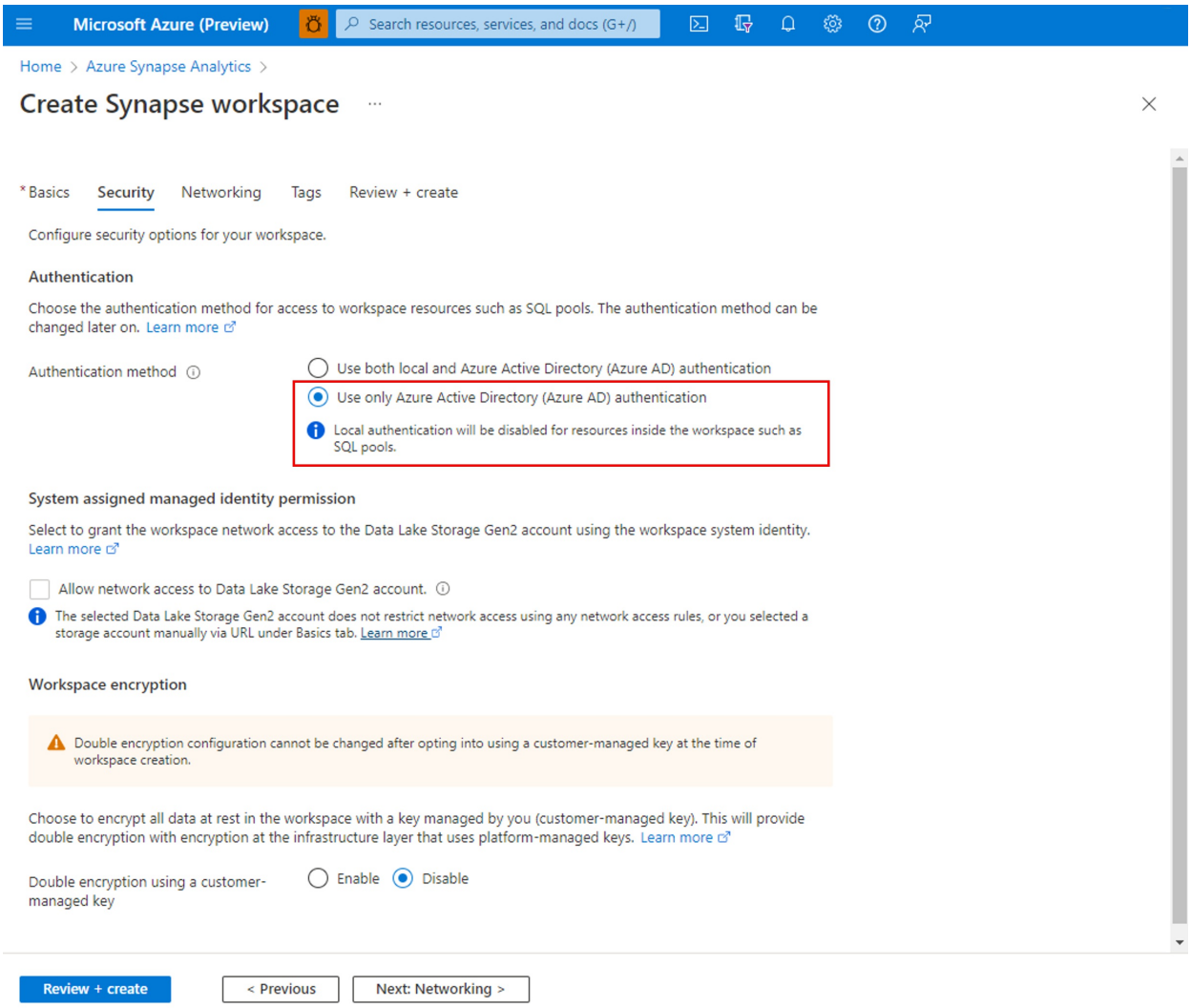
To create new users, you must have the `ALTER ANY USER` permission in the database. The `ALTER ANY USER` permission can be granted to any database user. The `ALTER ANY USER` permission is also held by the SQL administrator and Azure AD administrator accounts, and database users with the `CONTROL ON DATABASE` or `ALTER ON DATABASE` permission for that database, and by members of the `db_owner` database role.

To create a contained database user in Synapse SQL, you must connect to the database or instance using an Azure AD identity. To create the first contained database user, you must connect to the database by using an Azure AD administrator (who is the owner of the database).

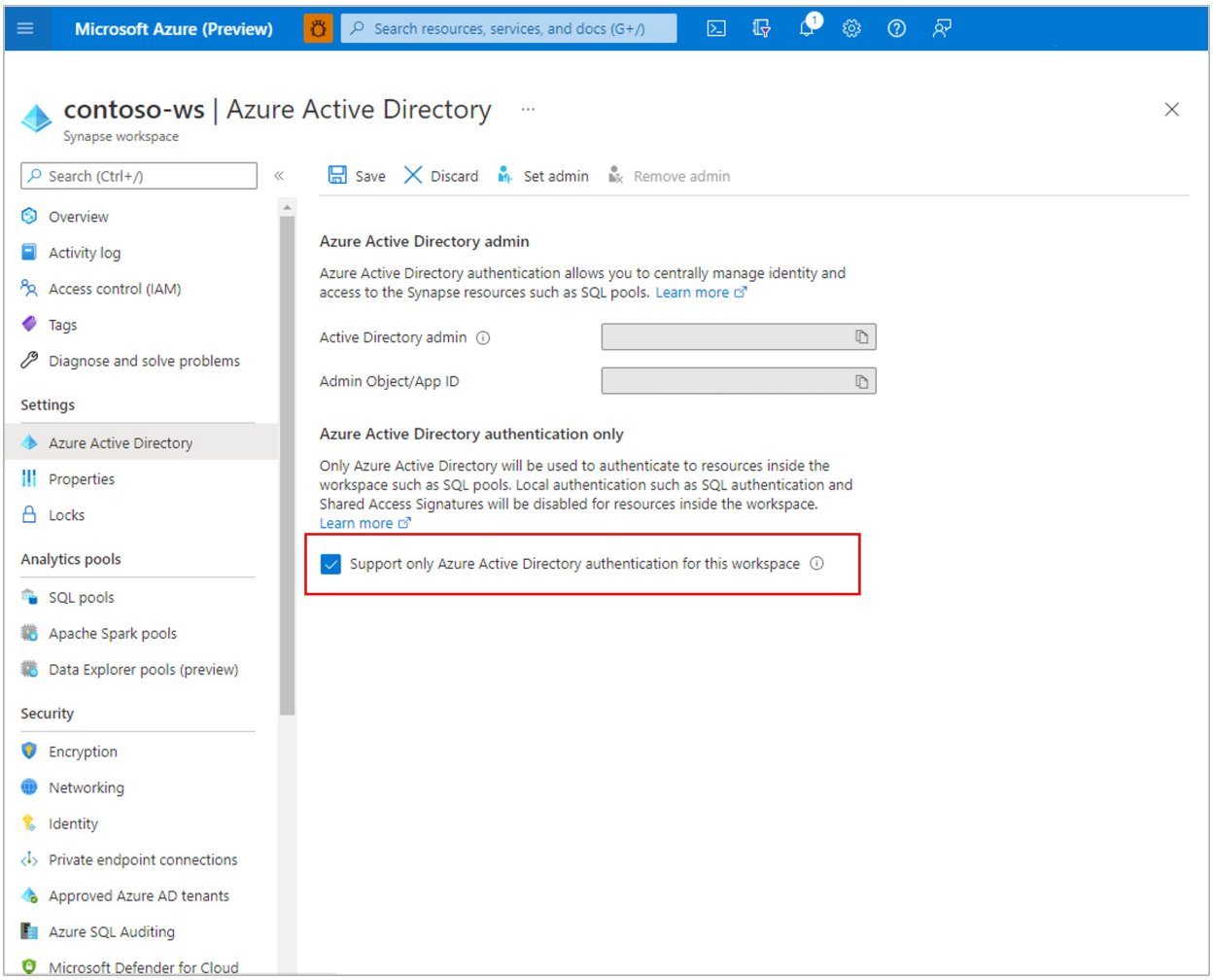
Any Azure AD authentication is only possible if the Azure AD admin was created for Synapse SQL. If the Azure Active Directory admin was removed from the server, existing Azure Active Directory users created previously inside Synapse SQL can no longer connect to the database using their Azure Active Directory credentials.

Disable local authentication

By allowing only Azure Active Directory authentication, centrally manage access to Azure Synapse resources, such as SQL pools. To disable local authentication in Synapse during workspace creation, select **Use only Azure Active Directory (Azure AD) authentication** as the authentication method. A SQL Administrator login will still be created but it will be disabled. Local authentication can be enabled later by an Azure Owner or Contributor of the Synapse workspace.



You can also disable local authentication after a workspace is created through the Azure portal. Local authentication cannot be disabled until an Azure Active Directory admin is created for the Azure Synapse workspace.



Azure AD features and limitations

- The following members of Azure AD can be provisioned in Synapse SQL:
  - Native members: A member created in Azure AD in the managed domain or in a customer domain. For more information, see Add your own domain name to Azure AD.
  - Federated domain members: A member created in Azure AD with a federated domain. For more information, see Microsoft Azure now supports federation with Windows Server Active Directory.
  - Imported members from other Azure ADs who are native or federated domain members.
  - Active Directory groups created as security groups.
- Azure AD users that are part of a group that has `db_owner` server role can't use the **CREATE DATABASE SCOPED CREDENTIAL** syntax in Synapse SQL. You will see the following error:

  ```
  SQL Error [2760] [S0001]: The specified schema name 'user@mydomain.com' either does not exist or you
  do not have permission to use it.
  ```

  Grant the `db_owner` role directly to the individual Azure AD user to mitigate the **CREATE DATABASE SCOPED CREDENTIAL** issue.

- These system functions return NULL values when executed under Azure AD principals:
  - `SUSER_ID()`
  - `SUSER_NAME(<admin ID>)`
  - `SUSER_SNAME(<admin SID>)`
  - `SUSER_ID(<admin name>)`
  - `SUSER_SID(<admin name>)`

Connect using Azure AD identities

Azure Active Directory authentication supports the following methods of connecting to a database using Azure AD identities:

- Azure Active Directory Password
- Azure Active Directory Integrated
- Azure Active Directory Universal with MFA
- Using Application token authentication

The following authentication methods are supported for Azure AD server principals (logins):

- Azure Active Directory Password
- Azure Active Directory Integrated
- Azure Active Directory Universal with MFA

Additional considerations

- To enhance manageability, we recommend you provision a dedicated Azure AD group as an administrator.
- Only one Azure AD administrator (a user or group) can be configured for Synapse SQL pool at any time.
  - The addition of Azure AD server principals (logins) for Synapse SQL allows the possibility of creating multiple Azure AD server principals (logins) that can be added to the `sysadmin` role.
- Only an Azure AD administrator for Synapse SQL can initially connect to Synapse SQL using an Azure Active Directory account. The Active Directory administrator can configure subsequent Azure AD database users.
- We recommend setting the connection timeout to 30 seconds.
- SQL Server 2016 Management Studio and SQL Server Data Tools for Visual Studio 2015 (version 14.0.60311.1April 2016 or later) support Azure Active Directory authentication. (Azure AD authentication is supported by the **.NET Framework Data Provider for SqlServer**; at least version .NET Framework 4.6). So, the newest versions of these tools and data-tier applications (DAC and .BACPAC) can use Azure AD authentication.
- Beginning with version 15.0.1, sqlcmd utility and bcp utility support Active Directory Interactive authentication with MFA.
- SQL Server Data Tools for Visual Studio 2015 requires at least the April 2016 version of the Data Tools (version 14.0.60311.1). Currently, Azure AD users aren't shown in SSDT Object Explorer. As a workaround, view the users in sys.database_principals.
- Microsoft JDBC Driver 6.0 for SQL Server supports Azure AD authentication. Also, see Setting the Connection Properties.
- The Azure Active Directory admin account controls access to dedicated SQL pools, while Synapse RBAC roles are used to control access to serverless pools, for example, the **Synapse Administrator** role. Configure Synapse RBAC roles via Synapse Studio, for more information, see How to manage Synapse RBAC role assignments in Synapse Studio.

Next steps

- For an overview of access and control in Synapse SQL, see Synapse SQL access control.
- For more information about database principals, see Principals.
- For more information about database roles, see Database roles.

Use Multi-factor AAD authentication with Synapse SQL (SSMS support for MFA)

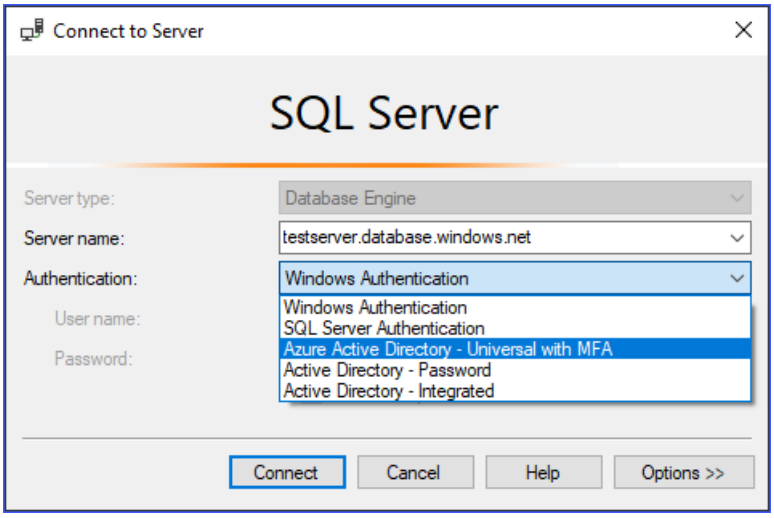
8/19/2022 • 4 minutes to read • Edit Online

Synapse SQL support connections from SQL Server Management Studio (SSMS) using *Active Directory Universal Authentication*.

This article discusses the differences between the various authentication options, and also the limitations associated with using Universal Authentication.

Download the latest SSMS - On the client computer, download the latest version of SSMS, from Download SQL Server Management Studio (SSMS).

For all the features discussed in this article, use at least July 2017, version 17.2. The most recent connection dialog box, should look similar to the following image:



The five authentication options

Active Directory Universal Authentication supports the two non-interactive authentication methods: - `Active Directory - Password` authentication - `Active Directory - Integrated` authentication

There are two non-interactive authentication models as well, which can be used in many different applications (ADO.NET, JDCB, ODC, etc.). These two methods never result in pop-up dialog boxes:

- `Active Directory - Password`
- `Active Directory - Integrated`

The interactive method is that also supports Azure AD Multi-Factor Authentication (MFA) is:

- `Active Directory - Universal with MFA`

Azure AD MFA helps safeguard access to data and applications while meeting user demand for a simple sign-in process. It delivers strong authentication with a range of easy verification options (phone call, text message, smart cards with pin, or mobile app notification), allowing users to choose the method they prefer. Interactive MFA with Azure AD can result in a pop-up dialog box for validation.

For a description of Multi-Factor Authentication, see Multi-Factor Authentication.

Azure AD domain name or tenant ID parameter

Beginning with SSMS version 17, users that are imported into the current Active Directory from other Azure Active Directories as guest users, can provide the Azure AD domain name, or tenant ID when they connect.

Guest users include users invited from other Azure ADs, Microsoft accounts such as outlook.com, hotmail.com, live.com, or other accounts like gmail.com. This information, allows **Active Directory Universal with MFA Authentication** to identify the correct authenticating authority. This option is also required to support Microsoft accounts (MSA) such as outlook.com, hotmail.com, live.com, or non-MSA accounts.

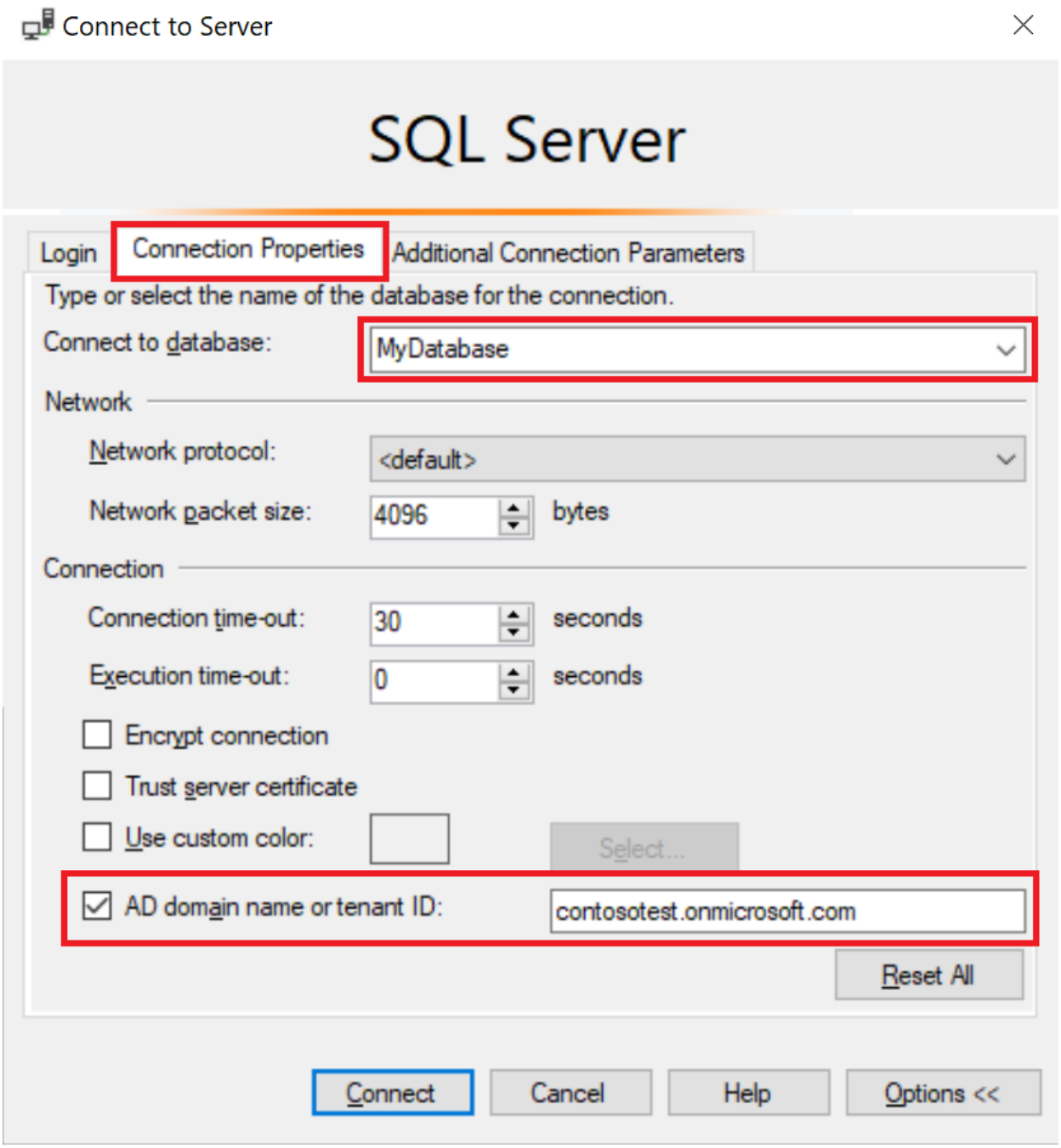
All these users who want to be authenticated using Universal Authentication must enter their Azure AD domain name or tenant ID. This parameter represents the current Azure AD domain name/tenant ID the Azure Server is linked with.

For example, if Azure Server is associated with Azure AD domain `contosotest.onmicrosoft.com` where user `joe@contosodev.onmicrosoft.com` is hosted as an imported user from Azure AD domain `contosodev.onmicrosoft.com`, the domain name required to authenticate this user is `contosotest.onmicrosoft.com`.

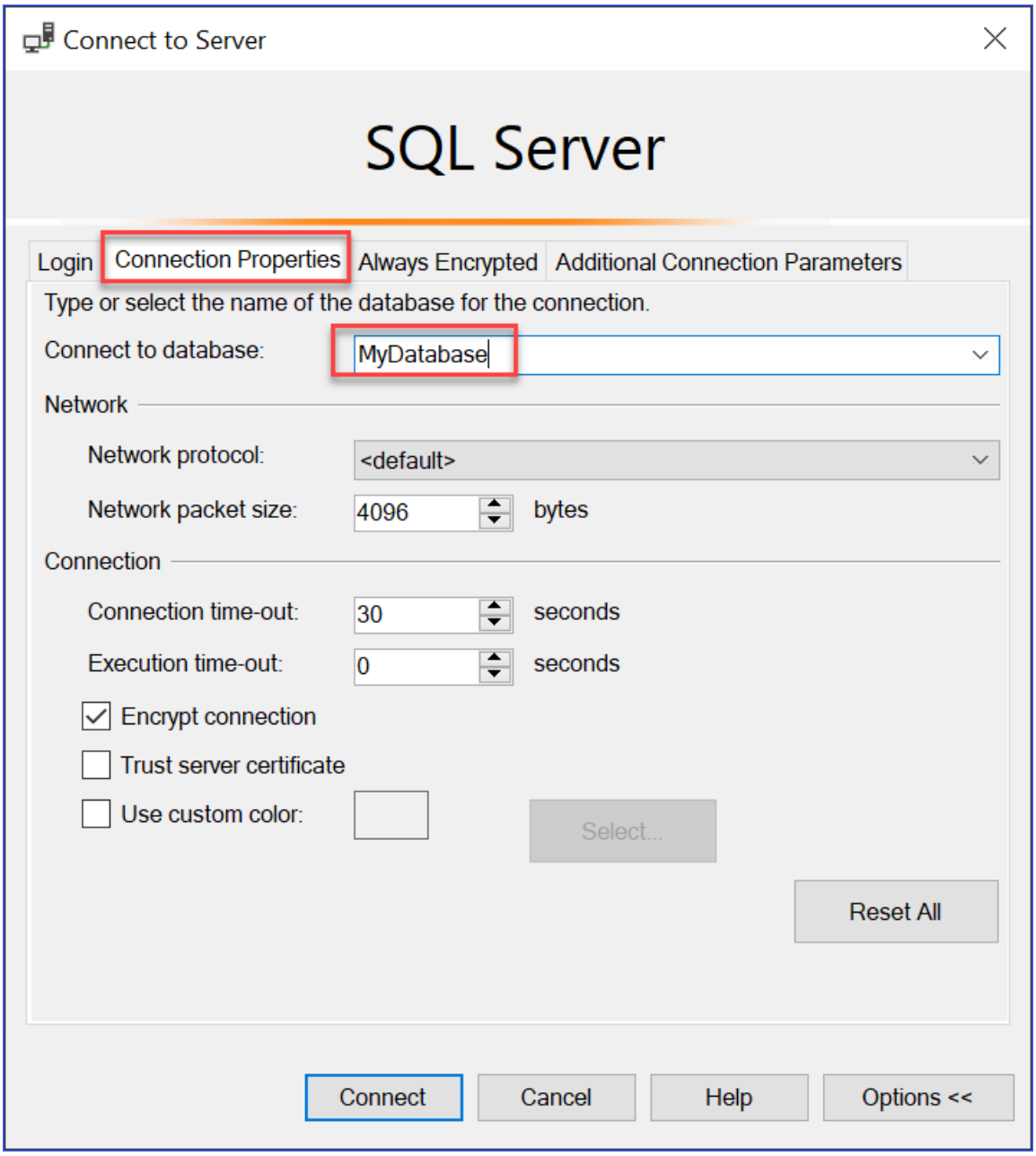
When the user is a native user of the Azure AD linked to Azure Server, and is not an MSA account, no domain name or tenant ID is required.

To enter the parameter (beginning with SSMS version 17.2), in the **Connect to Database** dialog box, complete the dialog box, selecting **Active Directory - Universal with MFA** authentication, select **Options**, complete the **User name** box, and then select the **Connection Properties** tab.

Check the **AD domain name or tenant ID** box, and provide authenticating authority, such as the domain name (**contosotest.onmicrosoft.com**) or the GUID of the tenant ID.



If you are running SSMS 18.x or later, then the AD domain name or tenant ID is no longer needed for guest users because 18.x or later automatically recognizes it.



Azure AD business to business support

Azure AD users supported for Azure AD B2B scenarios as guest users (see What is Azure B2B collaboration can connect to Synapse SQL only as part of members of a group created in current Azure AD and mapped manually using the Transact-SQL `CREATE USER` statement in a given database.

For example, if `steve@gmail.com` is invited to Azure AD `contosotest` (with the Azure Ad domain `contosotest.onmicrosoft.com`), an Azure AD group, such as `usergroup` must be created in the Azure AD that contains the `steve@gmail.com` member. Then, this group must be created for a specific database (that is, MyDatabase) by Azure AD SQL admin or Azure AD DBO by executing a Transact-SQL `CREATE USER [usergroup] FROM EXTERNAL PROVIDER` statement.

After the database user is created, then the user `steve@gmail.com` can log in to `MyDatabase` using the SSMS authentication option `Active Directory - Universal with MFA support`.

The usergroup, by default, has only the connect permission and any further data access that will need to be granted in the normal way.

As a guest user, `steve@gmail.com` must check the box and add the AD domain name `contosotest.onmicrosoft.com` in the SSMS **Connection Property** dialog box. The **AD domain name or tenant ID** option is only supported for the Universal with MFA connection options, otherwise it is greyed out.

Universal Authentication limitations for Synapse SQL

- SSMS and SqlPackage.exe are the only tools currently enabled for MFA through Active Directory Universal Authentication.
- SSMS version 17.2, supports multi-user concurrent access using Universal Authentication with MFA. Version 17.0 and 17.1, restricted a login for an instance of SSMS using Universal Authentication to a single Azure Active Directory account. To log in as another Azure AD account, you must use another instance of SSMS. (This restriction is limited to Active Directory Universal Authentication; you can log in to different servers using Active Directory Password Authentication, Active Directory Integrated Authentication, or SQL Server Authentication).
- SSMS supports Active Directory Universal Authentication for Object Explorer, Query Editor, and Query Store visualization.
- SSMS version 17.2 provides DacFx Wizard support for Export/Extract/Deploy Data database. Once a specific user is authenticated through the initial authentication dialog using Universal Authentication, the DacFx Wizard functions the same way it does for all other authentication methods.
- The SSMS Table Designer doesn't support Universal Authentication.
- There are no additional software requirements for Active Directory Universal Authentication except that you must use a supported version of SSMS.
- The Active Directory Authentication Library (ADAL) version for Universal authentication was updated to its latest ADAL.dll 3.13.9 available released version. See Active Directory Authentication Library 3.14.1.

Next steps

For more information, see the Connect to Synapse SQL with SQL Server Management Studio article.

SQL Authentication in Azure Synapse Analytics

8/19/2022 • 10 minutes to read • Edit Online

Azure Synapse Analytics has two SQL form-factors that enable you to control your resource consumption. This article explains how the two form-factors control the user authentication.

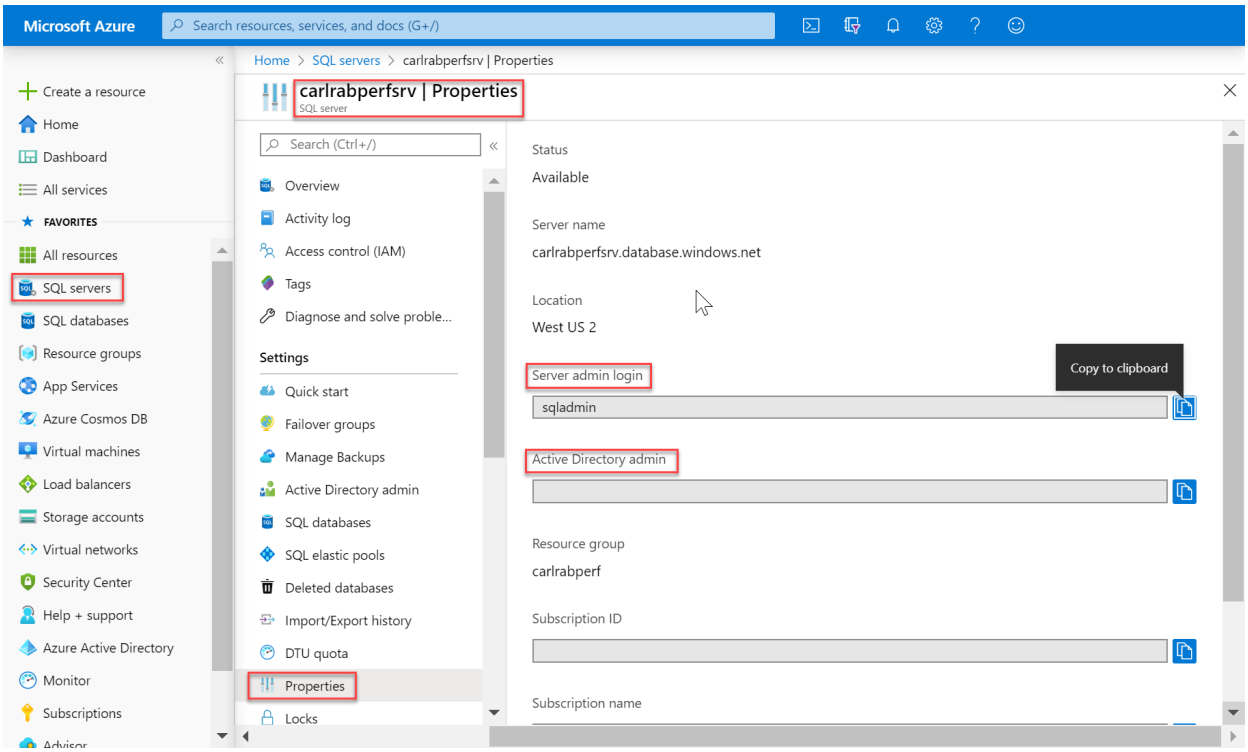
To authorize to Synapse SQL, you can use two authorization types:

- Azure Active Directory authorization
- SQL authorization

SQL authorization enables legacy applications to connect to Azure Synapse SQL in a familiar way. However, Azure Active Directory authentication allows you to centrally manage access to Azure Synapse resources, such as SQL pools. Azure Synapse Analytics supports disabling local authentication, such as SQL authentication, both during and after workspace creation. Once disabled, local authentication can be enabled at any time by authorized users. For more information on Azure AD-only authentication, see Disabling local authentication in Azure Synapse Analytics.

Administrative accounts

There are two administrative accounts (**SQL admin username** and **SQL Active Directory admin**) that act as administrators. To identify these administrator accounts for your SQL pools open the Azure portal, and navigate to the Properties tab of your Synapse workspace.



- **SQL admin username**

  When you create an Azure Synapse Analytics, you must name a **Server admin login**. SQL server creates that account as a login in the `master` database. This account connects using SQL Server authentication (user name and password). Only one of these accounts can exist.

- **SQL Active Directory admin**

One Azure Active Directory account, either an individual or security group account, can also be configured as an administrator. It's optional to configure an Azure AD administrator, but an Azure AD administrator **must** be configured if you want to use Azure AD accounts to connect to Synapse SQL.

- The Azure Active Directory admin account controls access to dedicated SQL pools, while Synapse RBAC roles are used to control access to serverless pools, for example, the **Synapse Administrator** role. Changing the Azure Active Directory administrator account will only affect the account's access to dedicated SQL pools.

The **SQL admin username** and **SQL Active Directory admin** accounts have the following characteristics:

- Are the only accounts that can automatically connect to any SQL Database on the server. (To connect to a user database, other accounts must either be the owner of the database, or have a user account in the user database.)
- These accounts enter user databases as the `dbo` user and they have all the permissions in the user databases. (The owner of a user database also enters the database as the `dbo` user.)
- Don't enter the `master` database as the `dbo` user, and have limited permissions in the `master` database.
- Are **not** members of the standard SQL Server `sysadmin` fixed server role, which is not available in SQL Database.
- Can create, alter, and drop databases, logins, users in the `master` database, and server-level IP firewall rules.
- Can add and remove members to the `dbmanager` and `loginmanager` roles.
- Can view the `sys.sql_logins` system table.

- Serverless SQL pool
- Dedicated SQL pool

To manage the users having access to serverless SQL pool, you can use the instructions below.

To create a login to serverless SQL pool, use the following syntax:

```
CREATE LOGIN Mary WITH PASSWORD = '<strong_password>';
-- or
CREATE LOGIN [Mary@domainname.net] FROM EXTERNAL PROVIDER;
```

When the login exists, you can create users in the individual databases within the serverless SQL pool endpoint and grant required permissions to these users. To create a user, you can use the following syntax:

```
CREATE USER Mary FROM LOGIN Mary;
-- or
CREATE USER Mary FROM LOGIN Mary@domainname.net;
-- or
CREATE USER [mike@contoso.com] FROM EXTERNAL PROVIDER;
```

Once login and user are created, you can use the regular SQL Server syntax to grant rights.

Non-administrator users

Generally, non-administrator accounts don't need access to the `master` database. Create contained database users at the database level using the CREATE USER (Transact-SQL) statement.

The user can be an Azure Active Directory authentication contained database user (if you have configured your environment for Azure AD authentication), or a SQL Server authentication contained database user, or a SQL Server authentication user based on a SQL Server authentication login (created in the previous step.)

To create users, connect to the database, and execute statements similar to the following examples:

```
CREATE USER Mary FROM LOGIN Mary;
CREATE USER [mike@contoso.com] FROM EXTERNAL PROVIDER;
```

Initially, only one of the administrators or the owner of the database can create users. To authorize additional users to create new users, grant that selected user the `ALTER ANY USER` permission, by using a statement such as:

```
GRANT ALTER ANY USER TO Mary;
```

To give additional users full control of the database, make them a member of the **db_owner** fixed database role.

In Azure SQL Database or synapse serverless, use the `ALTER ROLE` statement.

```
ALTER ROLE db_owner ADD MEMBER Mary;
```

In dedicated SQL pool use EXEC sp_addrolemember.

```
EXEC sp_addrolemember 'db_owner', 'Mary';
```

> **NOTE**
> One common reason to create a database user based on a server login is for users that need access to multiple databases. Since contained database users are individual entities, each database maintains its own user and its own password. This can cause overhead as the user must then remember each password for each database, and it can become untenable when having to change multiple passwords for many databases.

Groups and roles

Efficient access management uses permissions assigned to groups and roles instead of individual users.

- When using Azure Active Directory authentication, put Azure Active Directory users into an Azure Active Directory group. Create a contained database user for the group. Place one or more database users into a database role and then assign permissions to the database role.
- When using SQL Server authentication, create contained database users in the database. Place one or more database users into a database role and then assign permissions to the database role.

The database roles can be the built-in roles such as **db_owner**, **db_ddladmin**, **db_datawriter**, **db_datareader**, **db_denydatawriter**, and **db_denydatareader**. **db_owner** is commonly used to grant full permission to only a few users. The other fixed database roles are useful for getting a simple database in development quickly, but are not recommended for most production databases.

For example, the **db_datareader** fixed database role grants read access to every table in the database, which is usually more than is strictly necessary.

It's far better to use the CREATE ROLE statement to create your own user-defined database roles and carefully grant each role the least permissions necessary for the business need. When a user is a member of multiple roles, they aggregate the permissions of them all.

Permissions

There are over 100 permissions that can be individually granted or denied in SQL Database. Many of these permissions are nested. For example, the `UPDATE` permission on a schema includes the `UPDATE` permission on

each table within that schema. As in most permission systems, the denial of a permission overrides a grant.

Because of the nested nature and the number of permissions, it can take careful study to design an appropriate permission system to properly protect your database.

Start with the list of permissions at Permissions (Database Engine) and review the poster size graphic of database engine permissions.

Considerations and restrictions

When managing logins and users in SQL Database, consider the following points:

- You must be connected to the `master` database when executing the `CREATE/ALTER/DROP DATABASE` statements.
- The database user corresponding to the **Server admin** login can't be altered or dropped.
- **Server admin** will be disabled if Azure AD-only authentication is enabled.
- US-English is the default language of the **Server admin** login.
- Only the administrators (**Server admin** login or Azure AD administrator) and the members of the **dbmanager** database role in the `master` database have permission to execute the `CREATE DATABASE` and `DROP DATABASE` statements.
- You must be connected to the `master` database when executing the `CREATE/ALTER/DROP LOGIN` statements. However, using logins is discouraged. Use contained database users instead. For more information, see Contained Database Users - Making Your Database Portable.
- To connect to a user database, you must provide the name of the database in the connection string.
- Only the server-level principal login and the members of the **loginmanager** database role in the `master` database have permission to execute the `CREATE LOGIN`, `ALTER LOGIN`, and `DROP LOGIN` statements.
- When executing the `CREATE/ALTER/DROP LOGIN` and `CREATE/ALTER/DROP DATABASE` statements in an ADO.NET application, using parameterized commands isn't allowed. For more information, see Commands and Parameters.
- When executing the `CREATE USER` statement with the `FOR/FROM LOGIN` option, it must be the only statement in a Transact-SQL batch.
- When executing the `ALTER USER` statement with the `WITH LOGIN` option, it must be the only statement in a Transact-SQL batch.
- `CREATE/ALTER/DROP LOGIN` and `CREATE/ALTER/DROP USER` statements are not supported when Azure AD-only authentication is enabled for the Azure Synapse workspace.
- To `CREATE/ALTER/DROP` a user requires the `ALTER ANY USER` permission on the database.
- When the owner of a database role tries to add or remove another database user to or from that database role, the following error may occur: **User or role 'Name' does not exist in this database.** This error occurs because the user isn't visible to the owner. To resolve this issue, grant the role owner the `VIEW DEFINITION` permission on the user.

Next steps

For more information, see Contained Database Users - Making Your Database Portable.

Conditional Access in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

You can now configure conditional access policies for Azure Synapse workspaces. Conditional access is a tool provided by Azure Active Directory to bring several signals such as device type and device IP location together to make decisions to grant access, block access, or enforce multi-factor authentication for a resource. Conditional access policies are configured in Azure Active Directory. Learn more about conditional access.

Configure conditional access

The following steps show how to configure a conditional access policy for Azure Synapse workspaces.

1. Sign in to the Azure portal using an account with *global administrator permissions*, select **Azure Active Directory**, choose **Security** from the menu.
2. Select **Conditional Access**, then choose **+ New Policy**, and provide a name for the policy.
3. Under **Assignments**, select **Users and groups**, check the **Select users and groups** option, and then select an Azure AD user or group for Conditional access. Click Select, and then click Done.
4. Select **Cloud apps**, click **Select apps**. Select **Microsoft Azure Synapse Gateway**. Then click Select and Done.
5. Under **Access Controls**, select **Grant** and then check the policy you want to apply, and select Done.
6. Set the **Enable policy** toggle to **On**, then select Create.

Next steps

Learn more about conditional access policies and their components.

- Common conditional access policies
- Building a conditional access policy

Azure Synapse access control

8/19/2022 • 6 minutes to read • Edit Online

This article provides an overview of the mechanisms available to control access to Azure Synapse compute resources and data.

Overview

Azure Synapse provides a comprehensive and fine-grained access control system, that integrates:

- **Azure roles** for resource management and access to data in storage,
- **Synapse roles** for managing live access to code and execution,
- **SQL roles** for data plane access to data in SQL pools, and
- **Git permissions** for source code control, including continuous integration and deployment support.

Azure Synapse roles provide sets of permissions that can be applied at different scopes. This granularity makes it easy to grant appropriate access to administrators, developers, security personnel, and operators to compute resources and data.

Access control can be simplified by using security groups that are aligned with people's job roles. You only need to add and remove users from appropriate security groups to manage access.

Access control elements

Create and manage Azure Synapse compute resources

Azure roles are used to control management of:

- Dedicated SQL pools
- Data Explorer pools
- Apache Spark pools
- Integration runtimes

To *create* these resources, you need to be an Azure Owner or Contributor on the resource group. To *manage* them once created, you need to be an Azure Owner or Contributor on either the resource group or the individual resources.

An Azure Owner or Contributor can enable or disable Azure AD-only authentication for Azure Synapse workspaces. For more information on Azure AD-only authentication, see Disabling local authentication in Azure Synapse Analytics.

Develop and execute code in Azure Synapse

Synapse supports two development models.

- **Synapse live development**. You develop and debug code in Synapse Studio and then **publish** it to save and execute. The Synapse service is the source of truth for code editing and execution. Any unpublished work is lost when you close Synapse Studio.
- **Git-enabled development**. You develop and debug code in Synapse Studio and **commit** changes to a working branch of a Git repo. Work from one or more branches is integrated into a collaboration branch, from where you **publish** it to the service. The Git repo is the source of truth for code editing, while the service is the source of truth for execution. Changes must be committed to the Git repo or published to the service before closing Synapse Studio. Learn more about using Synapse Analytics with Git.

In both development models, any user with access to Synapse Studio can create code artifacts. However, you need additional permissions to publish artifacts to the service, read published artifacts, to commit changes to Git, to execute code, and to access linked data protected by credentials.

Azure Synapse roles

Azure Synapse roles are used to control access to the Synapse service. Different roles can permit you to:

- List published code artifacts,
- Publish code artifacts, linked services, and credential definitions,
- Execute code or pipelines that use Synapse compute resources,
- Execute code or pipelines that access linked data protected by credentials,
- View outputs associated with published code artifacts,
- Monitor compute resource status, and view runtime logs.

Azure Synapse roles can be assigned at the workspace scope or at finer-grained scopes to limit the permissions granted to specific Azure Synapse resources.

Git permissions

When using Git-enabled development in Git mode, you need Git permissions in addition to the Synapse User or Synapse RBAC (role-based access control) roles to read code artifacts, including linked service and credential definitions. To commit changes to code artifacts in Git mode, you need Git permissions, and the Synapse Artifact Publisher (Synapse RBAC) role.

Access data in SQL

When working with dedicated and serverless SQL pools, data plane access is controlled using SQL permissions.

The creator of a workspace is assigned as the Active Directory Admin on the workspace. After creation, this role can be assigned to a different user or to a security group in the Azure portal.

Serverless SQL pools: Synapse Administrators are granted `db_owner` (`DBO`) permissions on the serverless SQL pool, 'Built-in'. To grant other users access to serverless SQL pools, Synapse administrators need to run SQL scripts on each serverless pool.

Dedicated SQL pools: Synapse Administrators have full access to data in dedicated SQL pools, and the ability to grant access to other users. Synapse Administrators can also perform configuration and maintenance activities on dedicated pools, except for dropping databases. Active Directory Admin permission is granted to the creator of the workspace and the workspace MSI. Permission to access dedicated SQL pools isn't otherwise granted automatically. To grant other users or groups access to dedicated SQL pools, the Active Directory Admin or Synapse Administrator must run SQL scripts against each dedicated SQL pool.

See How to set up Synapse Access Control for examples of SQL scripts for granting SQL permissions in SQL pools.

Accessing data in Data Explorer pools

When working with Data Explorer pools, data plane access is controlled through Data Explorer permissions. Synapse Administrators are granted `All Database admin` permissions on Data Explorer pools. To grant other users or groups access to Data Explorer pools, Synapse administrators should refer to Security roles management. For more information on data plane access, see Data Explorer access control overview.

Accessing system-managed data in storage

Serverless SQL pools and Apache Spark tables store their data in an ADLS Gen2 container associated with the workspace. User-installed Apache Spark libraries are also managed in the same storage account. To enable these use cases, users and the workspace MSI must be granted **Storage Blob Data Contributor** access to this workspace ADLS Gen2 storage container.

Using security groups as a best practice

To simplify managing access control, you can use security groups to assign roles to individuals and groups. Security groups can be created to mirror personas or job functions in your organization that need access to Synapse resources or artifacts. These persona-based security groups can then be assigned one or more Azure roles, Synapse roles, SQL permissions, or Git permissions. With well-chosen security groups, it's easy to assign a user the required permissions by adding them to the appropriate security group.

> **NOTE**
> If using security groups to manage access, there is additional latency introduced by Azure Active Directory before changes take effect.

Access control enforcement in Synapse Studio

Synapse Studio will behave differently based on your permissions and the current mode:

- **Synapse live mode:** Synapse Studio will prevent you from seeing published content, publishing content, or taking other actions if you don't have the required permission. In some cases, you'll be prevented from creating code artifacts that you can't use or save.
- **Git-mode:** If you have Git permissions that let you commit changes to the current branch, then the commit action will be permitted if you have permission to publish changes to the live service (Synapse Artifact Publisher role).

In some cases, you're allowed to create code artifacts even without permission to publish or commit. This allows you to execute code (with the required execution permissions). For more information on the roles required for common tasks, see Understand the roles required to perform common tasks in Azure Synapse.

If a feature is disabled in Synapse Studio, a tooltip will indicate the required permission. Use the Synapse RBAC roles guide to look up which role is required to provide the missing permission.

Next steps

- Learn more about Synapse RBAC
- Learn more about Synapse RBAC roles
- Learn how to set up access control for a Synapse Workspace using security groups.
- Learn how to review Synapse RBAC role assignments
- Learn how to manage Synapse RBAC role assignments

What is Synapse role-based access control (RBAC)?

8/19/2022 • 3 minutes to read • Edit Online

Synapse RBAC extends the capabilities of Azure RBAC for Synapse workspaces and their content.

Azure RBAC is used to manage who can create, update, or delete the Synapse workspace and its SQL pools, Apache Spark pools, and Integration runtimes.

Synapse RBAC is used to manage who can:

- Publish code artifacts and list or access published code artifacts,
- Execute code on Apaches Spark pools and Integration runtimes,
- Access linked (data) services protected by credentials
- Monitor or cancel job execution, review job output, and execution logs.

> **NOTE**
> While Synapse RBAC is used to manage access to published SQL scripts, it provides only limited access control to serverless and dedicated SQL pools. Access to SQL pools is primarily controlled using SQL security.

What can I do with Synapse RBAC?

Here are some examples of what you can do with Synapse RBAC:

- Allow a user to publish changes made to Apache Spark notebooks and jobs to the live service.
- Allow a user to run and cancel notebooks and spark jobs on a specific Apache Spark pool.
- Allow a user to use specific credentials so they can run pipelines secured by the workspace system identity and access data in linked services secured with credentials.
- Allow an administrator to manage, monitor, and cancel job execution on specific Spark Pools.

How Synapse RBAC works

Like Azure RBAC, Synapse RBAC works by creating role assignments. A role assignment consists of three elements: a security principal, a role definition, and a scope.

Security Principals

A *security principal* is a user, group, service principal, or managed identity.

Roles

A *role* is a collection of permissions or actions that can be performed on specific resource types or artifact types.

Synapse provides built-in roles that define collections of actions that match the needs of different personas:

- Administrators can get full access to create and configure a workspace
- Developers can create, update and debug SQL scripts, notebooks, pipelines, and dataflows, but not be able to publish or execute this code on production compute resources/data
- Operators can monitor and manage system status, application execution and review logs, without access to code or the outputs from execution.
- Security staff can manage and configure endpoints without having access to code, compute resources or data.

Learn more about the built-in Synapse roles.

Scopes

A *scope* defines the resources or artifacts that the access applies to. Azure Synapse supports hierarchical scopes. Permissions granted at a higher-level scope are inherited by objects at a lower level. In Synapse RBAC, the top-level scope is a workspace. Assigning a role with workspace scope grants permissions to all applicable objects in the workspace.

Current supported scopes within a workspace are:

- Apache Spark pool
- Integration runtime
- linked service
- credential

Access to code artifacts is granted with workspace scope. Granting access to collections of artifacts within a workspace will be supported in a later release.

Resolving role assignments to determine permissions

A role assignment grants a principal the permissions defined by the role at the specified scope.

Synapse RBAC is an additive model like Azure RBAC. Multiple roles may be assigned to a single principal and at different scopes. When computing the permissions of a security principal, the system considers all roles assigned to the principal and to groups that directly or indirectly include the principal. It also considers the scope of each assignment in determining the permissions that apply.

Enforcing assigned permissions

In Synapse Studio, specific buttons or options may be grayed out or a permissions error may be returned when attempting an action if you don't have the required permissions.

If a button or option is disabled, hovering over the button or option shows a tooltip with the required permission. Contact a Synapse Administrator to assign a role that grants the required permission. You can see the roles that provide specific actions, see Synapse RBAC Roles.

Who can assign Synapse RBAC roles?

Synapse Administrators can assign Synapse RBAC roles. A Synapse Administrator at the workspace level can grant access at any scope. A Synapse Administrator at a lower-level scope can only grant access at that scope.

When a new workspace is created, the creator is automatically given the Synapse Administrator role at workspace scope.

To help you regain access to a workspace in the event that no Synapse Administrators are assigned or available to you, users with permissions to manage Azure RBAC role assignments on the workspace can also manage Synapse RBAC role assignments, allowing the addition of Synapse Administrator or other Synapse role assignments.

Where do I manage Synapse RBAC?

Synapse RBAC is managed from within Synapse Studio using the access control tools in the **Manage** hub.

Next steps

Understand the built-in Synapse RBAC roles.

Learn how to review Synapse RBAC role assignments for a workspace.

Learn how to assign Synapse RBAC roles

Synapse RBAC Roles

8/19/2022 • 8 minutes to read • Edit Online

The article describes the built-in Synapse RBAC (role-based access control) roles, the permissions they grant, and the scopes at which they can be used.

For more information on reviewing and assigning Synapse role memberships, see how to review Synapse RBAC role assignments and how to assign Synapse RBAC roles.

What's changed since the preview?

For users familiar with the Synapse RBAC roles provided during the preview, the following changes apply:

- Workspace Admin is renamed **Synapse Administrator**
- Apache Spark Admin is renamed **Synapse Apache Spark Administrator** and has permission to see all published code artifacts, including SQL scripts. This role no longer gives permission to use the workspace MSI, which requires the Synapse Credential User role. This permission is required to run pipelines.
- SQL Admin is renamed **Synapse SQL Administrator** and has permission to see all published code artifacts, including Spark notebooks and jobs. This role no longer gives permission to use the workspace MSI, which requires the Synapse Credential User role. This permission is required to run pipelines.
- **New finer-grained Synapse RBAC roles** are introduced that focus on supporting development and operations personas rather than specific analytics runtimes.
- **New lower-level scopes** are introduced for several roles. These scopes allow roles to be restricted to specific resources or objects.

Built-in Synapse RBAC roles and scopes

The following table describes the built-in roles and the scopes at which they can be used.

> **NOTE**
> Users with any Synapse RBAC role at any scope automatically have the Synapse User role at workspace scope.

> **IMPORTANT**
> Synapse RBAC roles do not grant permissions to create or manage SQL pools, Apache Spark pools, and Integration runtimes in Azure Synapse workspaces. Azure Owner or Azure Contributor roles on the resource group are required for these actions.

ROLE	PERMISSIONS	SCOPES

ROLE	PERMISSIONS	SCOPES
Synapse Administrator	Full Synapse access to SQL pools, Data Explorer pools, Apache Spark pools, and Integration runtimes. Includes create, read, update, and delete access to all published code artifacts. Includes Compute Operator, Linked Data Manager, and Credential User permissions on the workspace system identity credential. Includes assigning Synapse RBAC roles. In addition to Synapse Administrator, Azure Owners can also assign Synapse RBAC roles. Azure permissions are required to create, delete, and manage compute resources. *Can read and write artifacts* *Can do all actions on Spark activities.* *Can view Spark pool logs* *Can view saved notebook and pipeline output* *Can use the secrets stored by linked services or credentials* *Can assign and revoke Synapse RBAC roles at current scope*	Workspace Spark pool Integration runtime Linked service Credential
Synapse Apache Spark Administrator	Full Synapse access to Apache Spark Pools. Create, read, update, and delete access to published Spark job definitions, notebooks and their outputs, and to libraries, linked services, and credentials. Includes read access to all other published code artifacts. Doesn't include permission to use credentials and run pipelines. Doesn't include granting access. *Can do all actions on Spark artifacts* *Can do all actions on Spark activities*	Workspace Spark pool
Synapse SQL Administrator	Full Synapse access to serverless SQL pools. Create, read, update, and delete access to published SQL scripts, credentials, and linked services. Includes read access to all other published code artifacts. Doesn't include permission to use credentials and run pipelines. Doesn't include granting access. *Can do all actions on SQL scripts* *Can connect to SQL serverless endpoints with SQL* `db_datareader`, `db_datawriter`, `connect`, *and* `grant` *permissions*	Workspace

ROLE	PERMISSIONS	SCOPES
Synapse Contributor	Full Synapse access to Apache Spark pools and Integration runtimes. Includes create, read, update, and delete access to all published code artifacts and their outputs, including credentials and linked services. Includes compute operator permissions. Doesn't include permission to use credentials and run pipelines. Doesn't include granting access. *Can read and write artifacts* *Can view saved notebook and pipeline output* *Can do all actions on Spark activities* *Can view Spark pool logs*	Workspace Spark pool Integration runtime
Synapse Artifact Publisher	Create, read, update, and delete access to published code artifacts and their outputs. Doesn't include permission to run code or pipelines, or to grant access. *Can read published artifacts and publish artifacts* *Can view saved notebook, Spark job, and pipeline output*	Workspace
Synapse Artifact User	Read access to published code artifacts and their outputs. Can create new artifacts but can't publish changes or run code without additional permissions.	Workspace
Synapse Compute Operator	Submit Spark jobs and notebooks and view logs. Includes canceling Spark jobs submitted by any user. Requires additional use credential permissions on the workspace system identity to run pipelines, view pipeline runs and outputs. *Can submit and cancel jobs, including jobs submitted by others* *Can view Spark pool logs*	Workspace Spark pool Integration runtime
Synapse Monitoring Operator	Read published code artifacts, including logs and outputs for notebooks and pipeline runs. Includes ability to list and view details of serverless SQL pools, Apache Spark pools, Data Explorer pools, and Integration runtimes. Requires additional permissions to run/cancel pipelines, Spark notebooks, and Spark jobs.	Workspace

ROLE	PERMISSIONS	SCOPES
Synapse Credential User	Runtime and configuration-time use of secrets within credentials and linked services in activities like pipeline runs. To run pipelines, this role is required, scoped to the workspace system identity. *Scoped to a credential, permits access to data via a linked service that is protected by the credential (may also require compute use permission)* *Allows execution of pipelines protected by the workspace system identity credential*	Workspace Linked Service Credential
Synapse Linked Data Manager	Creation and management of managed private endpoints, linked services, and credentials. Can create managed private endpoints that use linked services protected by credentials	Workspace
Synapse User	List and view details of SQL pools, Apache Spark pools, Integration runtimes, and published linked services and credentials. Doesn't include other published code artifacts. Can create new artifacts but can't run or publish without additional permissions. *Can list and read Spark pools, Integration runtimes.*	Workspace, Spark pool Linked service Credential

Synapse RBAC roles and the actions they permit

NOTE

- All actions listed in the tables below are prefixed, "Microsoft.Synapse/..."
- All artifact read, write, and delete actions are with respect to published artifacts in the live service. These permissions do not affect access to artifacts in a connected Git repo.

The following table lists the built-in roles and the actions/permissions that each support.

ROLE	ACTIONS

ROLE	ACTIONS
Synapse Administrator	workspaces/read workspaces/roleAssignments/write, delete workspaces/managedPrivateEndpoint/write, delete workspaces/bigDataPools/useCompute/action workspaces/bigDataPools/viewLogs/action workspaces/integrationRuntimes/useCompute/action workspaces/integrationRuntimes/viewLogs/action workspaces/artifacts/read workspaces/notebooks/write, delete workspaces/sparkJobDefinitions/write, delete workspaces/sqlScripts/write, delete workspaces/kqlScripts/write, delete workspaces/dataFlows/write, delete workspaces/pipelines/write, delete workspaces/triggers/write, delete workspaces/datasets/write, delete workspaces/libraries/write, delete workspaces/linkedServices/write, delete workspaces/credentials/write, delete workspaces/notebooks/viewOutputs/action workspaces/pipelines/viewOutputs/action workspaces/linkedServices/useSecret/action workspaces/credentials/useSecret/action workspaces/linkConnections/read workspaces/linkConnections/write workspaces/linkConnections/delete workspaces/linkConnections/useCompute/action
Synapse Apache Spark Administrator	workspaces/read workspaces/bigDataPools/useCompute/action workspaces/bigDataPools/viewLogs/action workspaces/notebooks/viewOutputs/action workspaces/artifacts/read workspaces/notebooks/write, delete workspaces/sparkJobDefinitions/write, delete workspaces/libraries/write, delete workspaces/linkedServices/write, delete workspaces/credentials/write, delete
Synapse SQL Administrator	workspaces/read workspaces/artifacts/read workspaces/sqlScripts/write, delete workspaces/linkedServices/write, delete workspaces/credentials/write, delete

ROLE	ACTIONS
Synapse Contributor	workspaces/read workspaces/bigDataPools/useCompute/action workspaces/bigDataPools/viewLogs/action workspaces/integrationRuntimes/useCompute/action workspaces/integrationRuntimes/viewLogs/action workspaces/artifacts/read workspaces/notebooks/write, delete workspaces/sparkJobDefinitions/write, delete workspaces/sqlScripts/write, delete workspaces/kqlScripts/write, delete workspaces/dataFlows/write, delete workspaces/pipelines/write, delete workspaces/triggers/write, delete workspaces/datasets/write, delete workspaces/libraries/write, delete workspaces/linkedServices/write, delete workspaces/credentials/write, delete workspaces/notebooks/viewOutputs/action workspaces/pipelines/viewOutputs/action workspaces/linkConnections/read workspaces/linkConnections/write workspaces/linkConnections/delete workspaces/linkConnections/useCompute/action
Synapse Artifact Publisher	workspaces/read workspaces/artifacts/read workspaces/notebooks/write, delete workspaces/sparkJobDefinitions/write, delete workspaces/sqlScripts/write, delete workspaces/kqlScripts/write, delete workspaces/dataFlows/write, delete workspaces/pipelines/write, delete workspaces/triggers/write, delete workspaces/datasets/write, delete workspaces/libraries/write, delete workspaces/linkedServices/write, delete workspaces/credentials/write, delete workspaces/notebooks/viewOutputs/action workspaces/pipelines/viewOutputs/action
Synapse Artifact User	workspaces/read workspaces/artifacts/read workspaces/notebooks/viewOutputs/action workspaces/pipelines/viewOutputs/action
Synapse Compute Operator	workspaces/read workspaces/bigDataPools/useCompute/action workspaces/bigDataPools/viewLogs/action workspaces/integrationRuntimes/useCompute/action workspaces/integrationRuntimes/viewLogs/action workspaces/linkConnections/read workspaces/linkConnections/useCompute/action
Synapse Monitoring Operator	workspaces/read workspaces/artifacts/read workspaces/notebooks/viewOutputs/action workspaces/pipelines/viewOutputs/action workspaces/integrationRuntimes/viewLogs/action workspaces/bigDataPools/viewLogs/action

ROLE	ACTIONS
Synapse Credential User	workspaces/read workspaces/linkedServices/useSecret/action workspaces/credentials/useSecret/action
Synapse Linked Data Manager	workspaces/read workspaces/managedPrivateEndpoint/write, delete workspaces/linkedServices/write, delete workspaces/credentials/write, delete
Synapse User	workspaces/read

Synapse RBAC actions and the roles that permit them

The following table lists Synapse actions and the built-in roles that permit these actions:

ACTION	ROLE
workspaces/read	Synapse Administrator Synapse Apache Spark Administrator Synapse SQL Administrator Synapse Contributor Synapse Artifact Publisher Synapse Artifact User Synapse Compute Operator Synapse Monitoring Operator Synapse Credential User Synapse Linked Data Manager Synapse User
workspaces/roleAssignments/write, delete	Synapse Administrator
workspaces/managedPrivateEndpoint/write, delete	Synapse Administrator Synapse Linked Data Manager
workspaces/bigDataPools/useCompute/action	Synapse Administrator Synapse Apache Spark Administrator Synapse Contributor Synapse Compute Operator Synapse Monitoring Operator
workspaces/bigDataPools/viewLogs/action	Synapse Administrator Synapse Apache Spark Administrator Synapse Contributor Synapse Compute Operator
workspaces/integrationRuntimes/useCompute/action	Synapse Administrator Synapse Contributor Synapse Compute Operator Synapse Monitoring Operator
workspaces/integrationRuntimes/viewLogs/action	Synapse Administrator Synapse Contributor Synapse Compute Operator Synapse Monitoring Operator

ACTION	ROLE
workspaces/linkConnections/read	Synapse Administrator Synapse Contributor Synapse Compute Operator
workspaces/linkConnections/useCompute/action	Synapse Administrator Synapse Contributor Synapse Compute Operator
workspaces/artifacts/read	Synapse Administrator Synapse Apache Spark Administrator Synapse SQL Administrator Synapse Contributor Synapse Artifact Publisher Synapse Artifact User
workspaces/notebooks/write, delete	Synapse Administrator Synapse Apache Spark Administrator Synapse Contributor Synapse Artifact Publisher
workspaces/sparkJobDefinitions/write, delete	Synapse Administrator Synapse Apache Spark Administrator Synapse Contributor Synapse Artifact Publisher
workspaces/sqlScripts/write, delete	Synapse Administrator Synapse SQL Administrator Synapse Contributor Synapse Artifact Publisher
workspaces/kqlScripts/write, delete	Synapse Administrator Synapse Contributor Synapse Artifact Publisher
workspaces/dataFlows/write, delete	Synapse Administrator Synapse Contributor Synapse Artifact Publisher
workspaces/pipelines/write, delete	Synapse Administrator Synapse Contributor Synapse Artifact Publisher
workspaces/linkConnections/write, delete	Synapse Administrator Synapse Contributor
workspaces/triggers/write, delete	Synapse Administrator Synapse Contributor Synapse Artifact Publisher
workspaces/datasets/write, delete	Synapse Administrator Synapse Contributor Synapse Artifact Publisher

ACTION	ROLE
workspaces/libraries/write, delete	Synapse Administrator Synapse Apache Spark Administrator Synapse Contributor Synapse Artifact Publisher
workspaces/linkedServices/write, delete	Synapse Administrator Synapse Apache Spark Administrator Synapse SQL Administrator Synapse Contributor Synapse Artifact Publisher Synapse Linked Data Manager
workspaces/credentials/write, delete	Synapse Administrator Synapse Apache Spark Administrator Synapse SQL Administrator Synapse Contributor Synapse Artifact Publisher Synapse Linked Data Manager
workspaces/notebooks/viewOutputs/action	Synapse Administrator Synapse Apache Spark Administrator Synapse Contributor Synapse Artifact Publisher Synapse Artifact User
workspaces/pipelines/viewOutputs/action	Synapse Administrator Synapse Contributor Synapse Artifact Publisher Synapse Artifact User
workspaces/linkedServices/useSecret/action	Synapse Administrator Synapse Credential User
workspaces/credentials/useSecret/action	Synapse Administrator Synapse Credential User

Synapse RBAC scopes and their supported roles

The table below lists Synapse RBAC scopes and the roles that can be assigned at each scope.

> **NOTE**
> To create or delete an object you must have permissions at a higher-level scope.

SCOPE	ROLES

SCOPE	ROLES
Workspace	Synapse Administrator Synapse Apache Spark Administrator Synapse SQL Administrator Synapse Contributor Synapse Artifact Publisher Synapse Artifact User Synapse Compute Operator Synapse Monitoring Operator Synapse Credential User Synapse Linked Data Manager Synapse User
Apache Spark pool	Synapse Administrator Synapse Contributor Synapse Compute Operator
Integration runtime	Synapse Administrator Synapse Contributor Synapse Compute Operator
Linked service	Synapse Administrator Synapse Credential User
Credential	Synapse Administrator Synapse Credential User

NOTE

All artifact roles and actions are scoped at the workspace level.

Next steps

- Learn how to review Synapse RBAC role assignments for a workspace.
- Learn how to assign Synapse RBAC roles

Understand the roles required to perform common tasks in Azure Synapse

8/19/2022 • 7 minutes to read • Edit Online

This article will help you understand which Synapse RBAC (role-based access control) roles or Azure RBAC roles you need to get work done in Synapse Studio. To manage role membership, see Manage Synapse RBAC role assignments.

Synapse Studio access control and workflow summary

Access Synapse Studio

You can open Synapse Studio and view details of the workspace and list any of its Azure resources such as SQL pools, Spark pools, or Integration runtimes. You will see if you've been assigned any Synapse RBAC role or have the Azure Owner, Contributor, or Reader role on the workspace.

Resource management

You can create SQL pools, Data Explorer pools, Apache Spark pools, and Integration runtimes if you're an Azure Owner or Contributor on the workspace. When using ARM templates for automated deployment, you need to be an Azure Contributor on the resource group.

You can pause or scale a dedicated SQL pool, configure a Spark pool, or an integration runtime if you're an Azure Owner or Contributor on the workspace or that resource.

View and edit code artifacts

With access to Synapse Studio, you can create new code artifacts, such as SQL scripts, KQL scripts, notebooks, spark jobs, linked services, pipelines, dataflows, triggers, and credentials. These artifacts can be published or saved with additional permissions.

If you're a Synapse Artifact User, Synapse Artifact Publisher, Synapse Contributor, or Synapse Administrator you can list, open, and edit already published code artifacts.

Execute your code

You can execute SQL scripts on SQL pools if you have the necessary SQL permissions defined in the SQL pools. You can execute KQL scripts on Data Explorer pools if you have the necessary permissions.

You can run notebooks and Spark jobs if you have Synapse Compute Operator permissions on the workspace or specific Apache Spark pools.

With Compute Operator permissions on the workspace or specific integration runtimes, and appropriate credential permissions you can execute pipelines.

Monitor and manage execution

You can review the status of running notebooks and jobs in Apache Spark pools if you're a Synapse User.

You can review logs and cancel running jobs and pipelines if you're a Synapse Compute Operator at the workspace or for a specific Spark pool or pipeline.

Publish and save your code

You can publish new or updated code artifacts to the service if you're a Synapse Artifact Publisher, Synapse Contributor, or Synapse Administrator.

You can commit code artifacts to a working branch of a Git repository if the workspace is Git-enabled and you have Git permissions. With Git enabled, publishing is only allowed from the collaboration branch.

If you close Synapse Studio without publishing or committing changes to code artifacts, then those changes will be lost.

Tasks and required roles

The table below lists common tasks and for each task, the Synapse RBAC, or Azure RBAC roles required.

> **NOTE**
> Synapse Administrator is not listed for each task unless it is the only role that provides the necessary permission. A Synapse Administrator can perform all tasks enabled by other Synapse RBAC roles.

> **NOTE**
> Guest users from another tenant are also able to review, add, or change role assignments once they have been assigned as Synapse Administrator.

The minimum Synapse RBAC role required is shown.

All Synapse RBAC roles at any scope provide you Synapse User permissions at the workspace.

All Synapse RBAC permissions/actions shown in the table are prefixed `Microsoft/Synapse/workspaces/...`.

TASK (I WANT TO...)	ROLE (I NEED TO BE...)	SYNAPSE RBAC PERMISSION/ACTION
Open Synapse Studio on a workspace	Synapse User or	read
	Azure Owner or Contributor, or Reader on the workspace	none
List SQL pools or Data Explorer pools or Apache Spark pools, or Integration runtimes and access their configuration details	Synapse User or	read
	Azure Owner or Contributor, or Reader on the workspace	none
List linked services or credentials or managed private endpoints	Synapse User	read
SQL POOLS		
Create a dedicated SQL pool or a serverless SQL pool	Azure Owner or Contributor on the workspace	none
Manage (pause or scale, or delete) a dedicated SQL pool	Azure Owner or Contributor on the SQL pool or workspace	none

TASK (I WANT TO...)	ROLE (I NEED TO BE...)	SYNAPSE RBAC PERMISSION/ACTION
Create a SQL script	Synapse User or Azure Owner or Contributor on the workspace. *Additional SQL permissions are required to run a SQL script, publish, or commit changes*.	
List and open any published SQL script	Synapse Artifact User or Artifact Publisher, or Synapse Contributor	artifacts/read
Run a SQL script on a serverless SQL pool	SQL permissions on the pool (granted automatically to a Synapse Administrator)	none
Run a SQL script on a dedicated SQL pool	SQL permissions on the pool (granted automatically to a Synapse Administrator)	none
Publish a new or updated, or deleted SQL script	Synapse Artifact Publisher or Synapse Contributor	sqlScripts/write, delete
Commit changes to a SQL script to the Git repo	Requires Git permissions on the repo	
Assign Active Directory Admin on the workspace (via workspace properties in the Azure Portal)	Azure Owner or Contributor on the workspace	
DATA EXPLORER POOLS		
Create a Data Explorer pool	Azure Owner or Contributor on the workspace	none
Manage (pause or scale, or delete) a Data Explorer pool	Azure Owner or Contributor on the Data Explorer pool or workspace	none
Create a KQL script	Synapse User. *Additional Data Explorer permissions are required to run a script, publish, or commit changes*.	
List and open any published KQL script	Synapse Artifact User or Artifact Publisher, or Synapse Contributor	artifacts/read
Run a KQL script on a Data Explorer pool	Data Explorer permissions on the pool (granted automatically to a Synapse Administrator)	none
Publish new, update, or delete KQL script	Synapse Artifact Publisher or Synapse Contributor	kqlScripts/write, delete
Commit changes to a KQL script to the Git repo	Requires Git permissions on the repo	

TASK (I WANT TO...)	ROLE (I NEED TO BE...)	SYNAPSE RBAC PERMISSION/ACTION
APACHE SPARK POOLS		
Create an Apache Spark pool	Azure Owner or Contributor on the workspace	
Monitor Apache Spark applications	Synapse User	read
View the logs for notebook and job execution	Synapse Monitoring Operator	
Cancel any notebook or Spark job running on an Apache Spark pool	Synapse Compute Operator on the Apache Spark pool.	bigDataPools/useCompute
Create a notebook or job definition	Synapse User or Azure Owner or Contributor, or Reader on the workspace *Additional permissions are required to run, publish, or commit changes*	read
List and open a published notebook or job definition, including reviewing saved outputs	Synapse Artifact User or Synapse Monitoring Operator on the workspace	artifacts/read
Run a notebook and review its output, or submit a Spark job	Synapse Apache Spark Administrator or Synapse Compute Operator on the selected Apache Spark pool	bigDataPools/useCompute
Publish or delete a notebook or job definition (including output) to the service	Artifact Publisher on the workspace or Synapse Apache Spark Administrator	notebooks/write, delete
Commit changes to a notebook or job definition to the Git repo	Git permissions	none
PIPELINES, INTEGRATION RUNTIMES, DATAFLOWS, DATASETS & TRIGGERS		
Create, update, or delete an Integration runtime	Azure Owner or Contributor on the workspace	
Monitor Integration runtime status	Synapse Monitoring Operator	read, integrationRuntimes/viewLogs
Review pipeline runs	Synapse Monitoring Operator	read, pipelines/viewOutputs
Create a pipeline	Synapse User *Additional Synapse permissions are required to debug, add triggers, publish, or commit changes*	read

TASK (I WANT TO...)	ROLE (I NEED TO BE...)	SYNAPSE RBAC PERMISSION/ACTION
Create a dataflow or dataset	Synapse User *Additional Synapse permissions are required to publish, or commit changes*	read
List and open a published pipeline	Synapse Artifact User or Synapse Monitoring Operator	artifacts/read
Preview dataset data	Synapse User and Synapse Credential User on the WorkspaceSystemIdentity	
Debug a pipeline using the default Integration runtime	Synapse User and Synapse Credential User on the WorkspaceSystemIdentity credential	read, credentials/useSecret
Create a trigger, including trigger now (requires permission to execute the pipeline)	Synapse User and Synapse Credential User on the WorkspaceSystemIdentity	read, credentials/useSecret/action
Execute/run a pipeline	Synapse User and Synapse Credential User on the WorkspaceSystemIdentity	read, credentials/useSecret/action
Copy data using the Copy Data tool	Synapse User and Synapse Credential User on the Workspace System Identity	read, credentials/useSecret/action
Ingest data (using a schedule)	Synapse Author and Synapse Credential User on the Workspace System Identity	read, credentials/useSecret/action
Publish a new, updated, or deleted pipeline, dataflow, or trigger to the service	Synapse Artifact Publisher on the workspace	pipelines/write, delete dataflows/write, delete triggers/write, delete
Commit changes to pipelines, dataflows, datasets, or triggers to the Git repo	Git permissions	none
LINKED SERVICES		
Create a linked service (includes assigning a credential)	Synapse User *Additional permissions are required to use a linked service with credentials, or to publish, or commit changes*	read
List and open a published linked service	Synapse Artifact User	linkedServices/write, delete
Test connection on a linked service secured by a credential	Synapse User and Synapse Credential User	credentials/useSecret/action
Publish a linked service	Synapse Artifact Publisher or Synapse Linked Data Manager	linkedServices/write, delete

TASK (I WANT TO...)	ROLE (I NEED TO BE...)	SYNAPSE RBAC PERMISSION/ACTION
Commit linked service definitions to the Git repo	Git permissions	none
ACCESS MANAGEMENT		
Review Synapse RBAC role assignments at any scope	Synapse User	read
Assign and remove Synapse RBAC role assignments for users, groups, and service principals	Synapse Administrator at the workspace or at a specific workspace item scope	roleAssignments/write, delete

Next steps

- Review Synapse RBAC role assignments
- Manage Synapse RBAC role assignments

Managed identity for Azure Synapse

8/19/2022 • 7 minutes to read • Edit Online

This article helps you understand managed identity (formerly known as Managed Service Identity/MSI) and how it works in Azure Synapse.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Overview

Managed identities eliminate the need to manage credentials. Managed identities provide an identity for the service instance when connecting to resources that support Azure Active Directory (Azure AD) authentication. For example, the service can use a managed identity to access resources like Azure Key Vault, where data admins can securely store credentials or access storage accounts. The service uses the managed identity to obtain Azure AD tokens.

There are two types of supported managed identities:

- **System-assigned:** You can enable a managed identity directly on a service instance. When you allow a system-assigned managed identity during the creation of the service, an identity is created in Azure AD tied to that service instance's lifecycle. By design, only that Azure resource can use this identity to request tokens from Azure AD. So when the resource is deleted, Azure automatically deletes the identity for you. Azure Synapse Analytics requires that a system-assigned managed identity must be created along with the Synapse workspace.
- **User-assigned:** You may also create a managed identity as a standalone Azure resource. You can create a user-assigned managed identity and assign it to one or more instances of a Synapse workspace. In user-assigned managed identities, the identity is managed separately from the resources that use it.

Managed identity provides the below benefits:

- Store credential in Azure Key Vault, in which case-managed identity is used for Azure Key Vault authentication.
- Access data stores or computes using managed identity authentication, including Azure Blob storage, Azure Data Explorer, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure SQL Database, Azure SQL Managed Instance, Azure Synapse Analytics, REST, Databricks activity, Web activity, and more. Check the connector and activity articles for details.
- Managed identity is also used to encrypt/decrypt data and metadata using the customer-managed key stored in Azure Key Vault, providing double encryption.

System-assigned managed identity

> **NOTE**
>
> System-assigned managed identity is also referred to as 'Managed identity' elsewhere in the documentation and in the Synapse Studio UI for backward compatibility purpose. We will explicitly mention 'User-assigned managed identity' when referring to it.

Generate system-assigned managed identity

System-assigned managed identity is generated as follows:

- When creating a Synapse workspace through **Azure portal or PowerShell**, managed identity will always be created automatically.
- When creating a workspace through **SDK**, managed identity will be created only if you specify Identity = new ManagedIdentity" in the Synapse workspace object for creation." See example in .NET Quickstart - Create data factory.
- When creating Synapse workspace through **REST API**, managed identity will be created only if you specify "identity" section in request body. See example in REST quickstart - create data factory.

If you find your service instance doesn't have a managed identity associated following retrieve managed identity instruction, you can explicitly generate one by updating it with identity initiator programmatically:

- Generate managed identity using PowerShell
- Generate managed identity using REST API
- Generate managed identity using an Azure Resource Manager template
- Generate managed identity using SDK

> **NOTE**
>
> - Managed identity cannot be modified. Updating a service instance which already has a managed identity won't have any impact, and the managed identity is kept unchanged.
> - If you update a service instance which already has a managed identity without specifying the "identity" parameter in the factory or workspace objects or without specifying "identity" section in REST request body, you will get an error.
> - When you delete a service instance, the associated managed identity will be deleted along.

Generate system-assigned managed identity using PowerShell

Call **New-AzSynapseWorkspace** command, then you see "Identity" fields being newly generated:

```
PS C:\> $creds = New-Object System.Management.Automation.PSCredential ("ContosoUser", $password)
PS C:\> New-AzSynapseWorkspace -ResourceGroupName <resourceGroupName> -Name <workspaceName> -Location
<region> -DefaultDataLakeStorageAccountName <storageAccountName> -DefaultDataLakeStorageFileSystem
<fileSystemName> -SqlAdministratorLoginCredential $creds

DefaultDataLakeStorage           : Microsoft.Azure.Commands.Synapse.Models.PSDataLakeStorageAccountDetails
ProvisioningState                : Succeeded
SqlAdministratorLogin            : ContosoUser
VirtualNetworkProfile            :
Identity                         : Microsoft.Azure.Commands.Synapse.Models.PSManagedIdentity
ManagedVirtualNetwork            :
PrivateEndpointConnections       : {}
WorkspaceUID                     : <workspaceUid>
ExtraProperties                  : {[WorkspaceType, Normal], [IsScopeEnabled, False]}
ManagedVirtualNetworkSettings    :
Encryption                       : Microsoft.Azure.Commands.Synapse.Models.PSEncryptionDetails
WorkspaceRepositoryConfiguration :
Tags                             :
TagsTable                        :
Location                         : <region>
Id                               : /subscriptions/<subsID>/resourceGroups/<resourceGroupName>/providers/
                                   Microsoft.Synapse/workspaces/<workspaceName>
Name                             : <workspaceName>
Type                             : Microsoft.Synapse/workspaces
```

Generate system-assigned managed identity using REST API

> **NOTE**
>
> If you attempt to update a service instance that already has a managed identity without either specifying the **identity** parameter in the workspace object or providing an **identity** section in the REST request body, you will get an error.

Call the API below with the "identity" section in the request body:

```
PATCH
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Mic
rosoft.Synapse/workspaces/{workspaceName}?api-version=2018-06-01
```

Request body: add "identity": { "type": "SystemAssigned" }.

```
{
    "name": "<workspaceName>",
    "location": "<region>",
    "properties": {},
    "identity": {
        "type": "SystemAssigned"
    }
}
```

Response: managed identity is created automatically, and "identity" section is populated accordingly.

```
{
    "name": "<workspaceName>",
    "tags": {},
    "properties": {
        "provisioningState": "Succeeded",
        "loggingStorageAccountKey": "**********",
        "createTime": "2021-09-26T04:10:01.1135678Z",
        "version": "2018-06-01"
    },
    "identity": {
        "type": "SystemAssigned",
        "principalId": "765ad4ab-XXXX-XXXX-XXXX-51ed985819dc",
        "tenantId": "72f988bf-XXXX-XXXX-XXXX-2d7cd011db47"
    },
    "id":
"/subscriptions/<subscriptionId>/resourceGroups/<resourceGroupName>/providers/Microsoft.Synapse/workspaces/<
workspaceName>",
    "type": "Microsoft.Synapse/workspaces",
    "location": "<region>"
}
```

Generate system-assigned managed identity using an Azure Resource Manager template

Template: add "identity": { "type": "SystemAssigned" }.

```
{
    "contentVersion": "1.0.0.0",
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "resources": [{
        "name": "<workspaceName>",
        "apiVersion": "2018-06-01",
        "type": "Microsoft.Synapse/workspaces",
        "location": "<region>",
        "identity": {
            "type": "SystemAssigned"
        }
    }]
}
```

Generate system-assigned managed identity using SDK

```
Workspace workspace = new Workspace
{
    Identity = new ManagedIdentity
    {
        Type = ResourceIdentityType.SystemAssigned
    },
    DefaultDataLakeStorage = new DataLakeStorageAccountDetails
    {
        AccountUrl = <defaultDataLakeStorageAccountUrl>,
        Filesystem = <DefaultDataLakeStorageFilesystem>
    },
    SqlAdministratorLogin = <SqlAdministratorLoginCredentialUserName>
    SqlAdministratorLoginPassword = <SqlAdministratorLoginCredentialPassword>,
    Location = <region>
};
client.Workspaces.CreateOrUpdate(resourceGroupName, workspaceName, workspace);
```

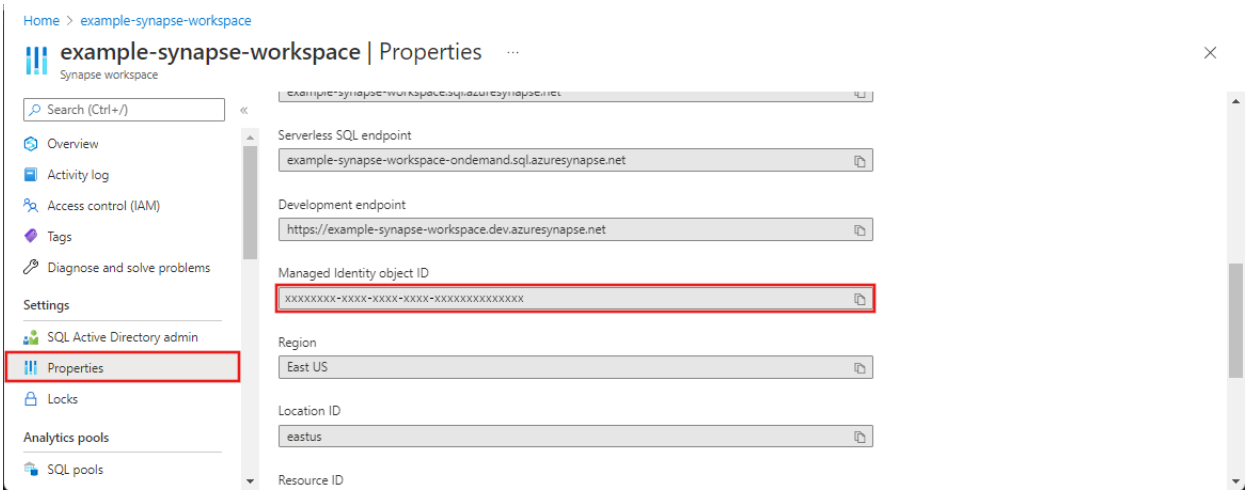
Retrieve system-assigned managed identity

You can retrieve the managed identity from Azure portal or programmatically. The following sections show some samples.

> **TIP**
>
> If you don't see the managed identity, generate managed identity by updating your service instance.

Retrieve system-assigned managed identity using Azure portal

You can find the managed identity information from Azure portal -> your Synapse workspace -> Properties.

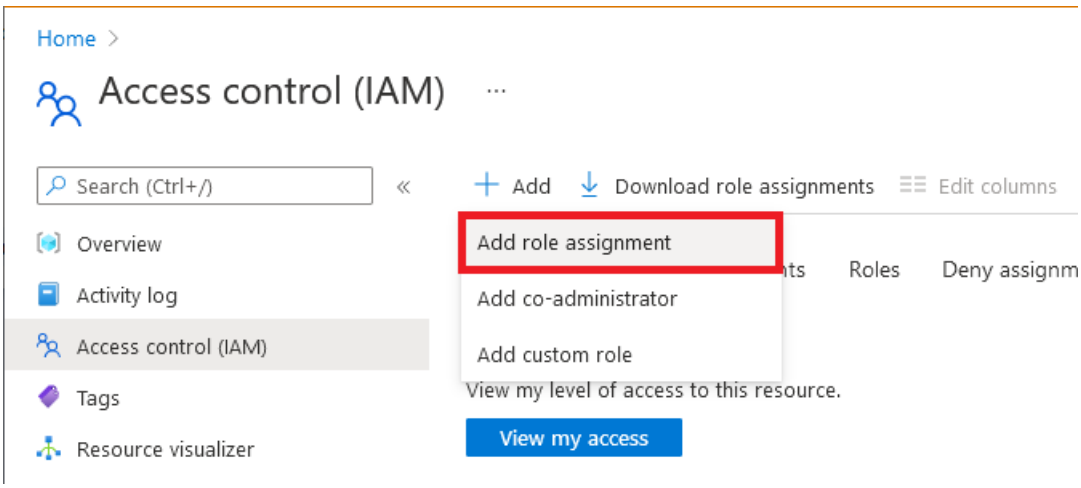


- Managed Identity Object ID

The managed identity information will also show up when you create linked service, which supports managed identity authentication, like Azure Blob, Azure Data Lake Storage, Azure Key Vault, etc.

To grant permissions, follow these steps. For detailed steps, see Assign Azure roles using the Azure portal.

1. Select **Access control (IAM)**.
2. Select **Add** > **Add role assignment**.



3. On the **Members** tab, select **Managed identity**, and then select **Select members**.
4. Select your Azure subscription.
5. Under **System-assigned managed identity**, select **Synapse workspace**, and then select a workspace. You can also use the object ID or workspace name (as the managed-identity name) to find this identity. To get the managed identity's application ID, use PowerShell.
6. On the **Review + assign** tab, select **Review + assign** to assign the role.

Retrieve system-assigned managed identity using PowerShell

The managed identity principal ID and tenant ID will be returned when you get a specific service instance as

follows. Use the **PrincipalId** to grant access:

```
PS C:\> (Get-AzSynapseWorkspace -ResourceGroupName <resourceGroupName> -Name <workspaceName>).Identity

IdentityType   PrincipalId                          TenantId
------------   -----------                          --------
SystemAssigned cadadb30-XXXX-XXXX-XXXX-ef3500e2ff05 72f988bf-XXXX-XXXX-XXXX-2d7cd011db47
```

You can get the application ID by copying above principal ID, then running below Azure Active Directory command with principal ID as parameter.

```
PS C:\> Get-AzADServicePrincipal -ObjectId cadadb30-XXXX-XXXX-XXXX-ef3500e2ff05

ServicePrincipalNames : {76f668b3-XXXX-XXXX-XXXX-1b3348c75e02,
https://identity.azure.net/P86P8g6nt1QxfPJx22om8MOooMf/Ag0Qf/nnREppHkU=}
ApplicationId         : 76f668b3-XXXX-XXXX-XXXX-1b3348c75e02
DisplayName           : <workspaceName>
Id                    : cadadb30-XXXX-XXXX-XXXX-ef3500e2ff05
Type                  : ServicePrincipal
```

Retrieve managed identity using REST API

The managed identity principal ID and tenant ID will be returned when you get a specific service instance as follows.

Call below API in the request:

```
GET
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Mic
rosoft.Synapse/workspaces/{workspaceName}?api-version=2018-06-01
```

Response: You will get response like shown in below example. The "identity" section is populated accordingly.

```
{
  "properties": {
    "defaultDataLakeStorage": {
      "accountUrl": "https://exampledatalakeaccount.dfs.core.windows.net",
      "filesystem": "examplefilesystem"
    },
    "encryption": {
      "doubleEncryptionEnabled": false
    },
    "provisioningState": "Succeeded",
    "connectivityEndpoints": {
      "web": "https://web.azuresynapse.net?
workspace=%2fsubscriptions%2{subscriptionId}%2fresourceGroups%2f{resourceGroupName}%2fproviders%2fMicrosoft.
Synapse%2fworkspaces%2f{workspaceName}",
      "dev": "https://{workspaceName}.dev.azuresynapse.net",
      "sqlOnDemand": "{workspaceName}-ondemand.sql.azuresynapse.net",
      "sql": "{workspaceName}.sql.azuresynapse.net"
    },
    "managedResourceGroupName": "synapseworkspace-managedrg-f77f7cf2-XXXX-XXXX-XXXX-c4cb7ac3cf4f",
    "sqlAdministratorLogin": "sqladminuser",
    "privateEndpointConnections": [],
    "workspaceUID": "e56f5773-XXXX-XXXX-XXXX-a0dc107af9ea",
    "extraProperties": {
      "WorkspaceType": "Normal",
      "IsScopeEnabled": false
    },
    "publicNetworkAccess": "Enabled",
    "cspWorkspaceAdminProperties": {
      "initialWorkspaceAdminObjectId": "3746a407-XXXX-XXXX-XXXX-842b6cf1fbcc"
    },
    "trustedServiceBypassEnabled": false
  },
  "type": "Microsoft.Synapse/workspaces",
  "id":
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Synapse/workspaces/{
workspaceName}",
  "location": "eastus",
  "name": "{workspaceName}",
  "identity": {
    "type": "SystemAssigned",
    "tenantId": "72f988bf-XXXX-XXXX-XXXX-2d7cd011db47",
    "principalId": "cadadb30-XXXX-XXXX-XXXX-ef3500e2ff05"
  },
  "tags": {}
}
```


TIP

To retrieve the managed identity from an ARM template, add an **outputs** section in the ARM JSON:

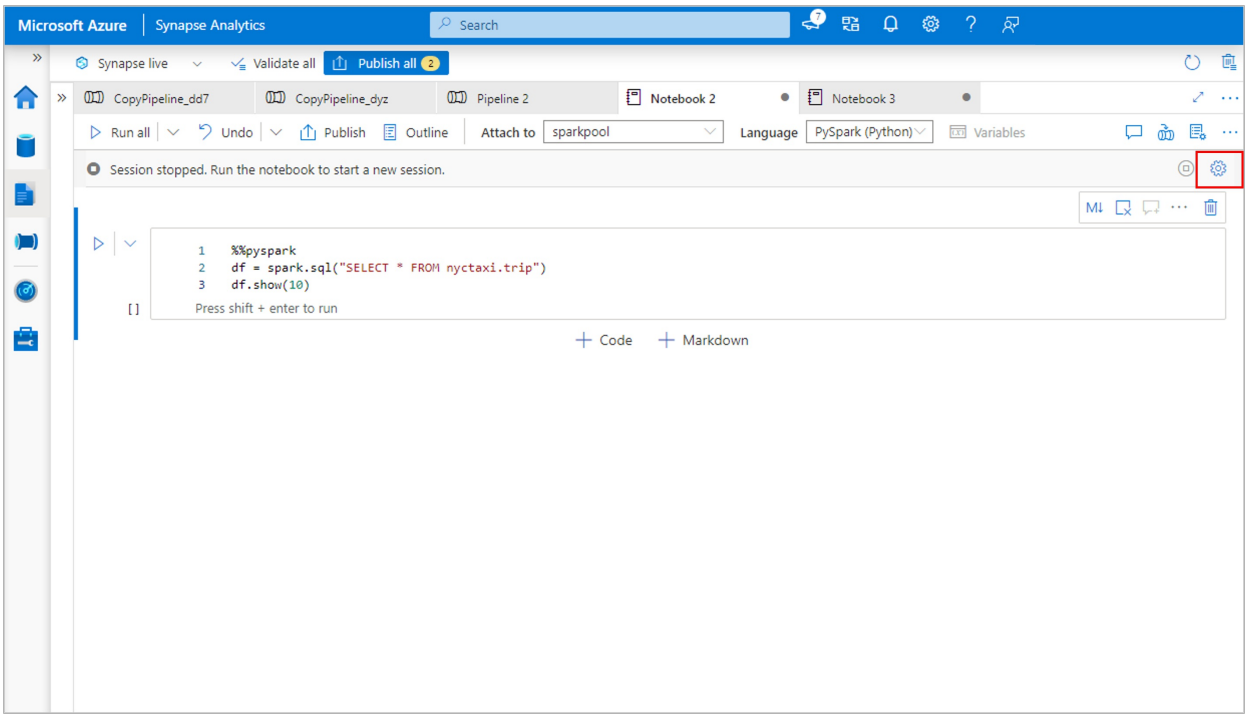

```
{
    "outputs":{
        "managedIdentityObjectId":{
            "type":"string",
            "value":"[reference(resourceId('Microsoft.Synapse/workspaces', parameters('<workspaceName>')),
'2018-06-01', 'Full').identity.principalId]"
        }
    }
}
```

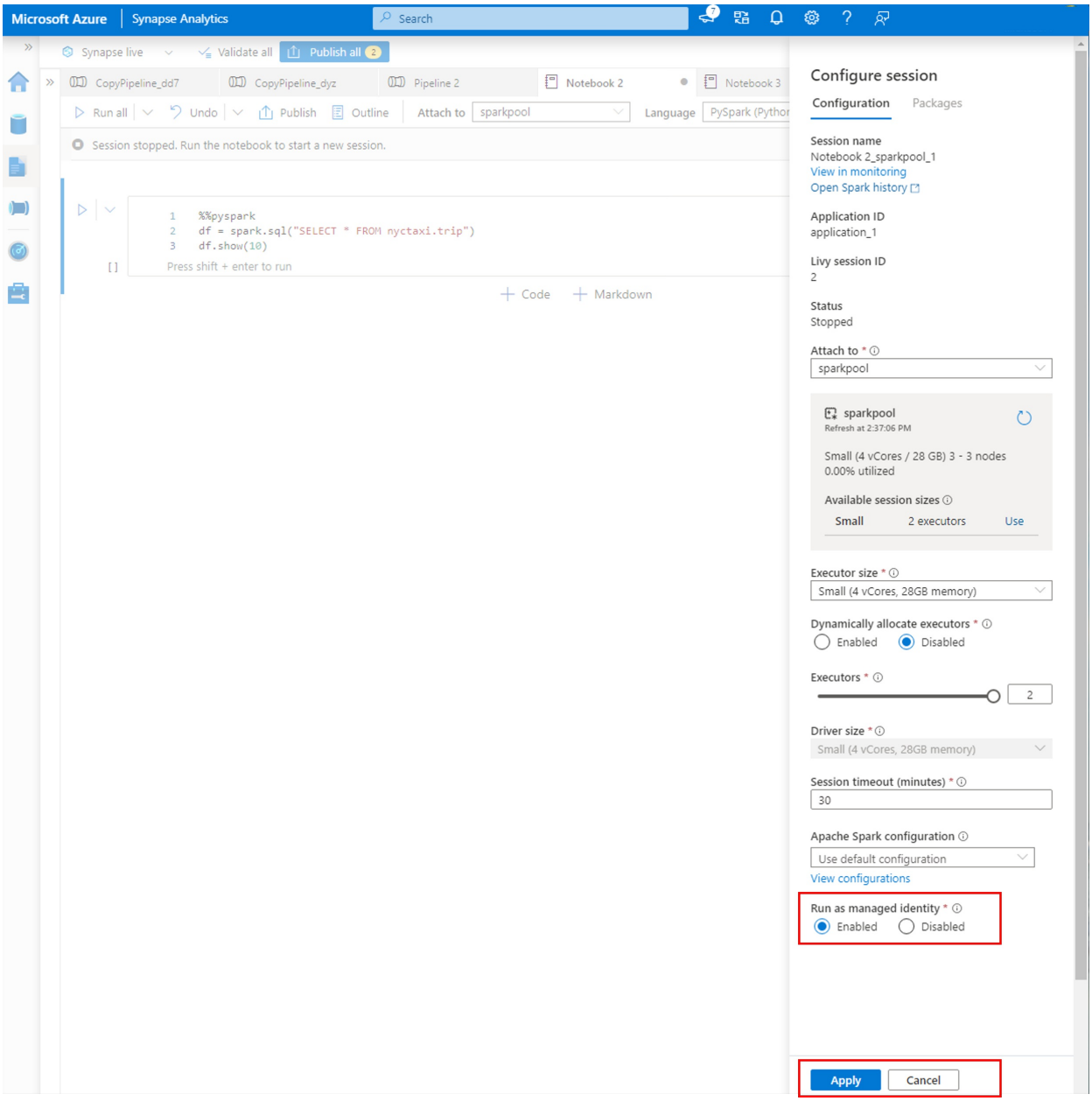

Execute Azure Synapse Spark Notebooks with system assigned managed identity

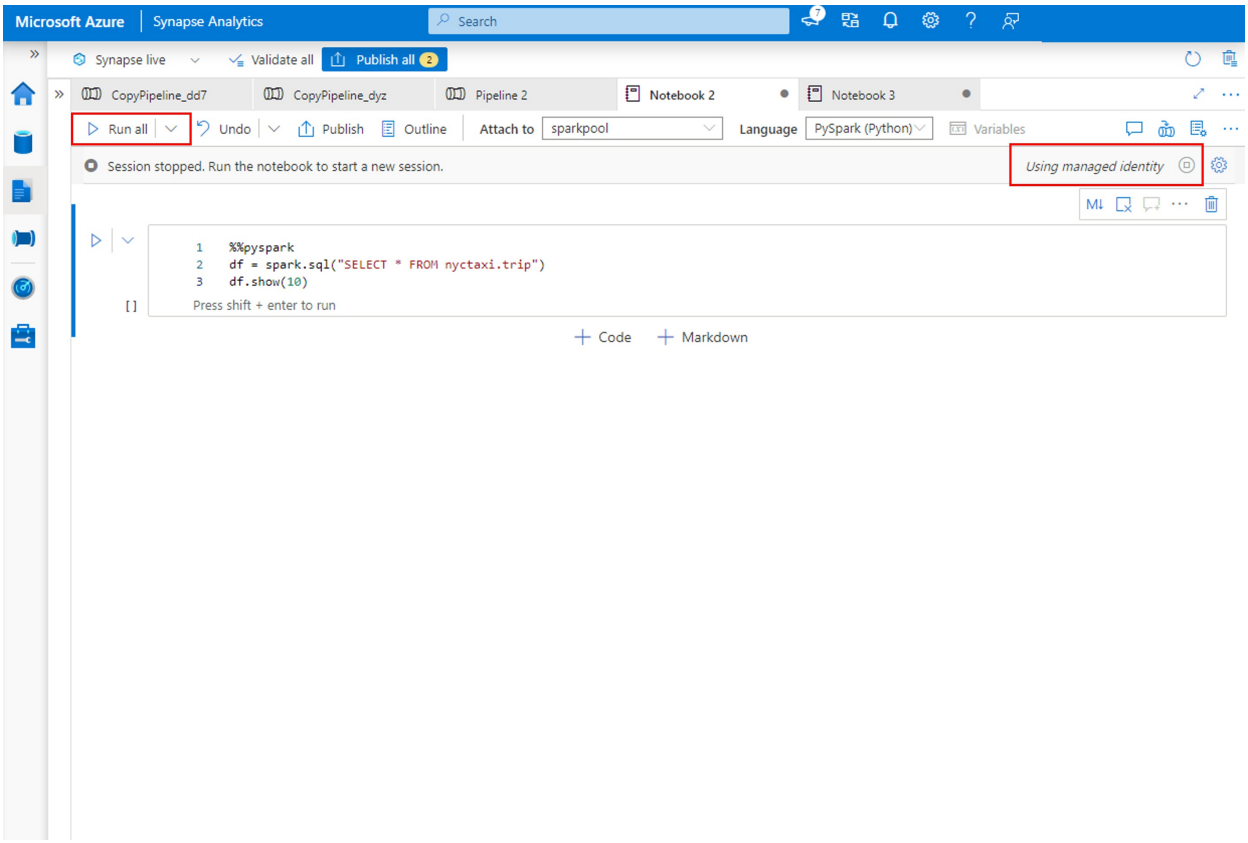
You can easily execute Synapse Spark Notebooks with the system assigned managed identity (or workspace

managed identity) by enabling *Run as managed identity* from the *Configure session* menu. To execute Spark Notebooks with workspace managed identity, users need to have following RBAC roles:

- Synapse Compute Operator on the workspace or selected Spark pool
- Synapse Credential User on the workspace managed identity




Microsoft Azure
Synapse Analytics
Search
Synapse live
Validate all
Publish all 2
CopyPipeline_dd7
CopyPipeline_dyz
Pipeline 2
Notebook 2
Notebook 3
Run all
Undo
Publish
Outline
Attach to
sparkpool
Language
PySpark (Python)
Session stopped. Run the notebook to start a new session.
1 %%pyspark
2 df = spark.sql("SELECT * FROM nyctaxi.trip")
3 df.show(10)
[ ]
Press shift + enter to run
Code
Markdown
Configure session
Configuration
Packages
Session name
Notebook 2_sparkpool_1
View in monitoring
Open Spark history
Application ID
application_1
Livy session ID
2
Status
Stopped
Attach to *
sparkpool
sparkpool
Refresh at 2:37:06 PM
Small (4 vCores / 28 GB) 3 - 3 nodes
0.00% utilized
Available session sizes
Small
2 executors
Use
Executor size *
Small (4 vCores, 28GB memory)
Dynamically allocate executors *
Enabled
Disabled
Executors *
2
Driver size *
Small (4 vCores, 28GB memory)
Session timeout (minutes) *
30
Apache Spark configuration
Use default configuration
View configurations
Run as managed identity *
Enabled
Disabled
Apply
Cancel



User-assigned managed identity

You can create, delete, manage user-assigned managed identities in Azure Active Directory. For more details refer to Create, list, delete, or assign a role to a user-assigned managed identity using the Azure portal.

In order to use a user-assigned managed identity, you must first create credentials in your service instance for the UAMI.

Next steps

- Create credentials.

See the following topics that introduce when and how to use managed identity:

- Store credential in Azure Key Vault.
- Copy data from/to Azure Data Lake Store using managed identities for Azure resources authentication.

See Managed Identities for Azure Resources Overview for more background on managed identities for Azure resources, on which managed identity in Azure Synapse is based.

Credentials in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Prerequisites

Users must have the Managed Identity Operator (Azure RBAC) role or a custom role with **Microsoft.ManagedIdentity/userAssignedIdentities/*/assign/action** RBAC action to configure a user assigned managed identity as a credential. Additional RBAC is required to create and use credentials in Synapse. Learn more.

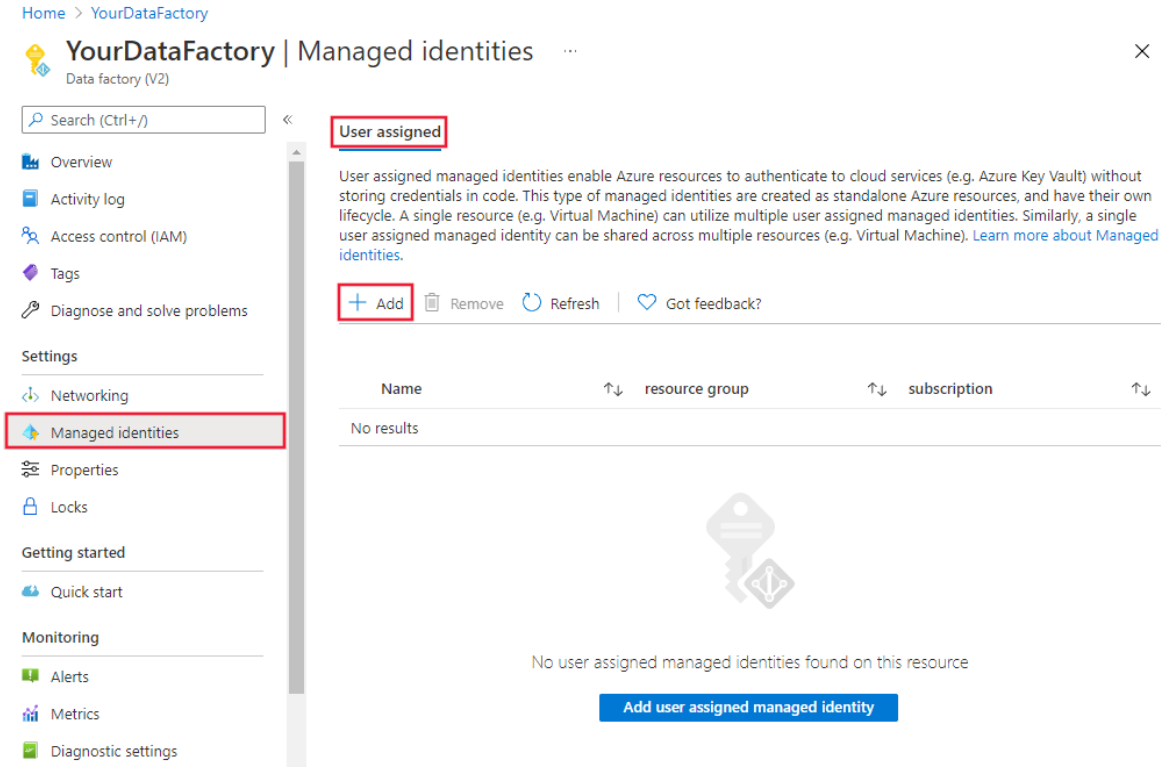
Using credentials

We are introducing Credentials which can contain user-assigned managed identities, service principals, and also lists the system-assigned managed identity that you can use in the linked services that support Azure Active Directory (AAD) authentication. It helps you consolidate and manage all your AAD-based credentials.

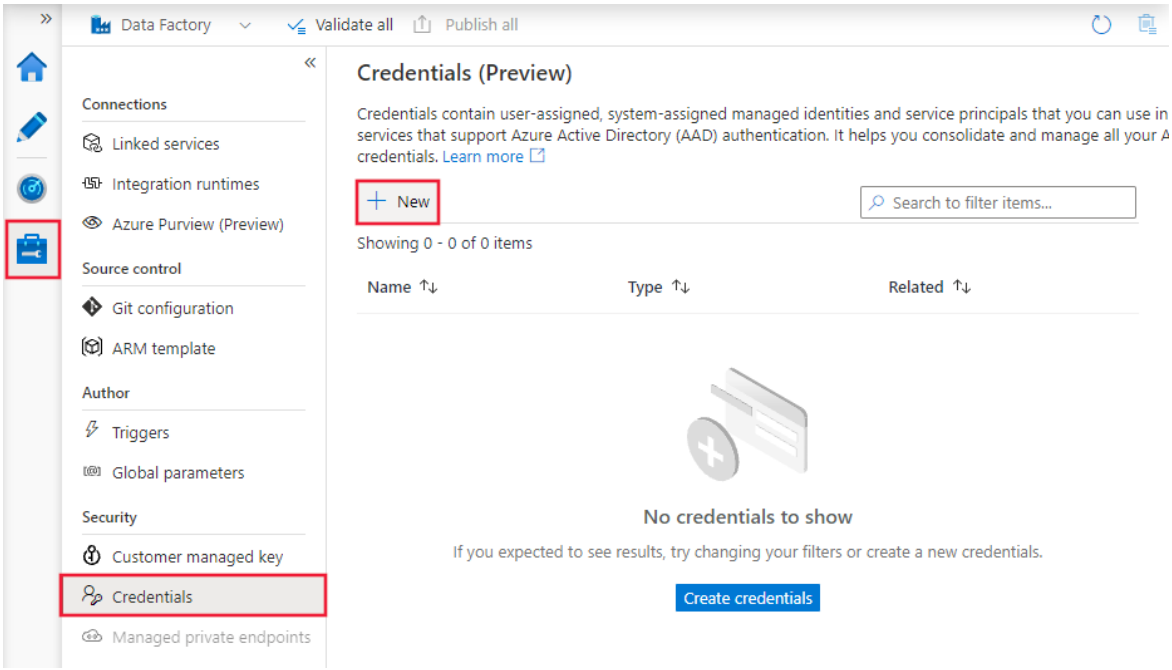
Below are the generic steps for using a **user-assigned managed identity** in the linked services for authentication.

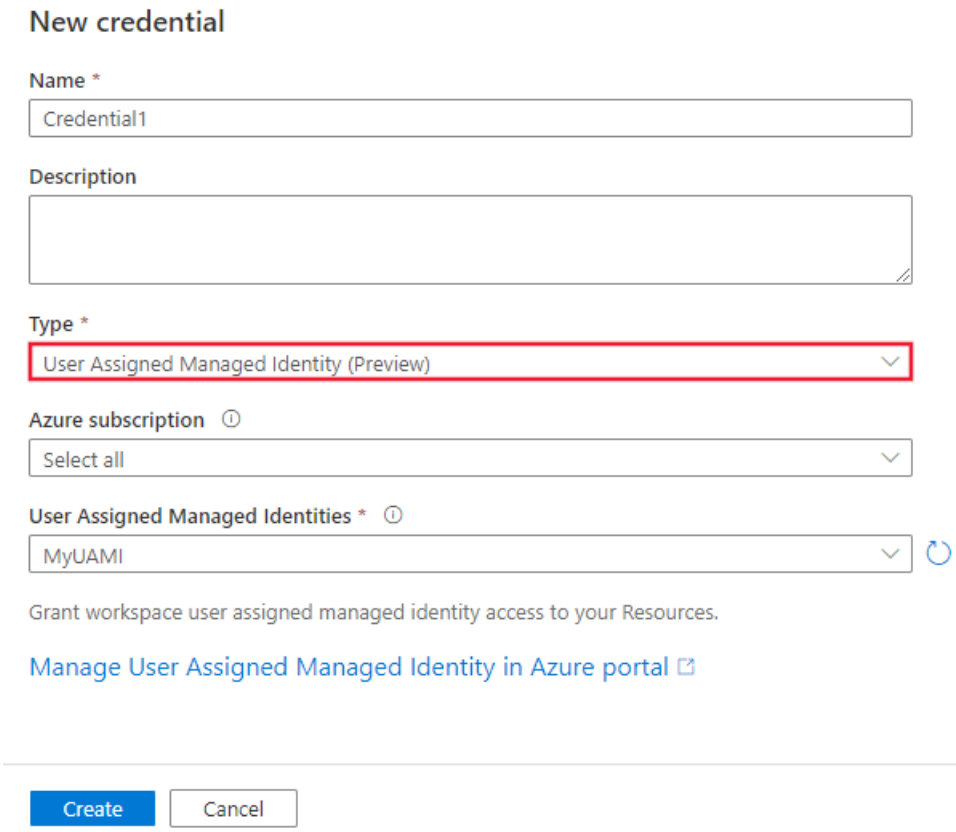
- Azure Data Factory
- Azure Synapse

1. If you do not have a user-assigned managed identity created in Azure, first create one in the Azure portal Managed Identities page.
2. Associate the user-assigned managed identity to the data factory instance using Azure portal, SDK, PowerShell, REST API. The screenshot below used Azure portal (data factory blade) to associate the user-assigned managed identity.

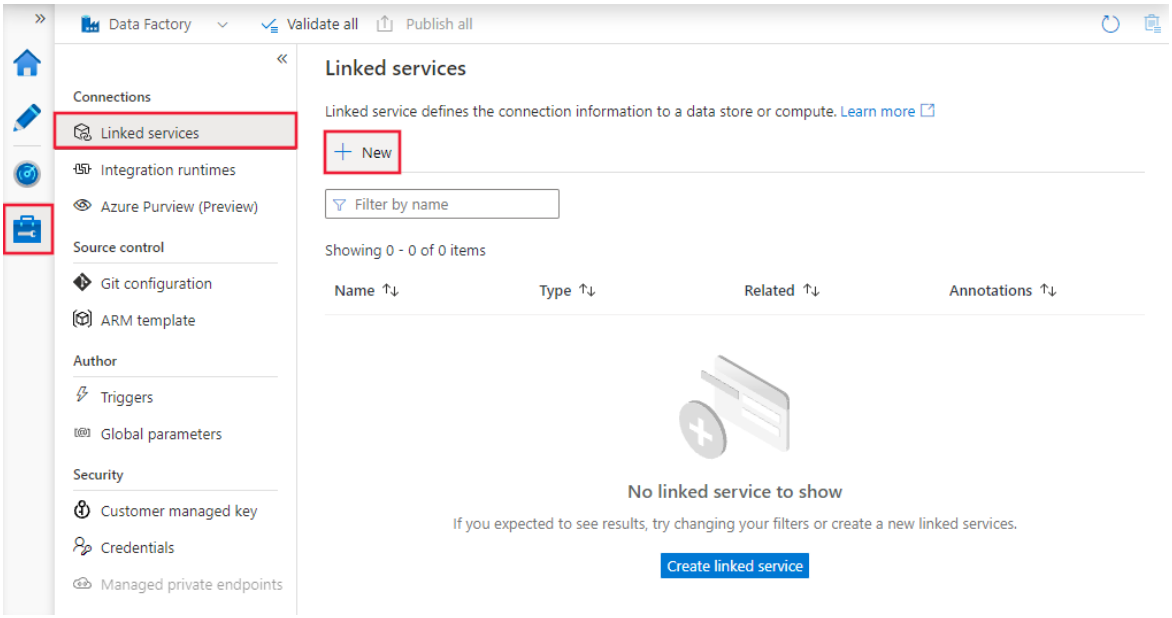


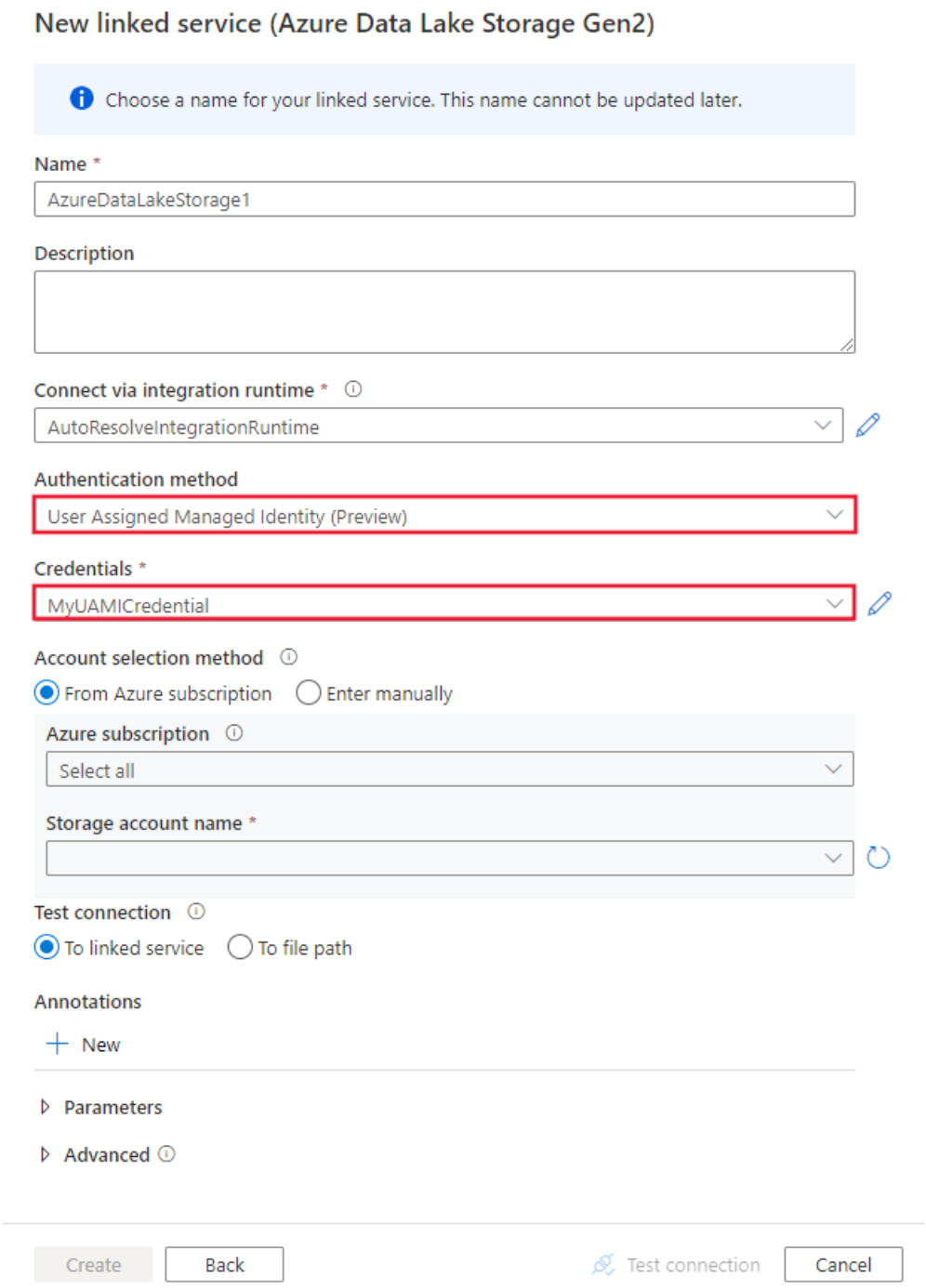
3. Create a **Credential** in data factory user interface interactively. You can select the user-assigned managed identity associated with the data factory in Step 1.





4. Create a new linked service and select **User-assigned managed identity** under authentication





NOTE

You can use SDK/ PowerShell/ REST APIs for the above actions. Linked services with user-assigned managed identity are currently not supported in Synapse Spark.

Next steps

- Managed identity

See the following topics that introduce when and how to use managed identity:

- Store credential in Azure Key Vault
- Copy data from/to Azure Data Lake Store using managed identities for Azure resources authentication

See Managed Identities for Azure Resources Overview for more background on managed identities for Azure resources, which data factory managed identity is based upon.

Use Azure Monitor with your Azure Synapse Analytics workspace

8/19/2022 • 9 minutes to read • Edit Online

Cloud applications are complex and have many moving parts. Monitors provide data to help ensure that your applications stay up and running in a healthy state. Monitors also help you avoid potential problems and troubleshoot past ones. You can use monitoring data to gain deep insights about your applications. This knowledge helps you improve application performance and maintainability. It also helps you automate actions that otherwise require manual intervention.

Azure Monitor provides base-level infrastructure metrics, alerts, and logs for most Azure services. Azure diagnostic logs are emitted by a resource and provide rich, frequent data about the operation of that resource. Azure Synapse Analytics can write diagnostic logs in Azure Monitor.

For more information, see Azure Monitor overview.

Metrics

With Monitor, you can gain visibility into the performance and health of your Azure workloads. The most important type of Monitor data is the metric, which is also called the performance counter. Metrics are emitted by most Azure resources. Monitor provides several ways to configure and consume these metrics for monitoring and troubleshooting.

To access these metrics, complete the instructions in Azure Monitor data platform.

Workspace-level metrics

Here are some of the metrics emitted by workspaces:

METRIC	METRIC CATEGORY, DISPLAY NAME	UNIT	AGGREGATION TYPES	DESCRIPTION
IntegrationActivityRunsEnded	Integration, Activity runs metric	Count	Sum (default), Count	The total number of activity runs that occurred/ended within a 1-minute window. Use the Result dimension of this metric to filter by Succeeded, Failed, or Cancelled final state.

METRIC	METRIC CATEGORY, DISPLAY NAME	UNIT	AGGREGATION TYPES	DESCRIPTION
IntegrationPipelineRunsEnded	Integration, Pipeline runs metric	Count	Sum (default), Count	The total number of pipeline runs that occurred/ended within a 1-minute window. Use the Result dimension of this metric to filter by Succeeded, Failed, or Cancelled final state.
IntegrationTriggerRunsEnded	Integration, Trigger runs metric	Count	Sum (default), Count	The total number of trigger runs that occurred/ended within a 1-minute window. Use the Result dimension of this metric to filter by Succeeded, Failed, or Cancelled final state.
BuiltinSqlPoolDataProcessedBytes	Built-in SQL pool, Data processed (bytes)	Byte	Sum (default)	Amount of data processed by the built-in serverless SQL pool.
BuiltinSqlPoolLoginAttempts	Built-in SQL pool, Login attempts	Count	Sum (default)	Number of login attempts for the built-in serverless SQL pool.
BuiltinSqlPoolDataRequestsEnded	Built-in SQL pool, Requests ended (bytes)	Count	Sum (default)	Number of ended SQL requests for the built-in serverless SQL pool. Use the Result dimension of this metric to filter by final state.

Dedicated SQL pool metrics

Here are some of the metrics emitted by dedicated SQL pools:

METRIC	DISPLAY NAME	UNIT	AGGREGATION TYPES	DESCRIPTION
DWULimit	DWU limit	Count	Max (default), Min, Avg	Configured size of the SQL pool

METRIC	DISPLAY NAME	UNIT	AGGREGATION TYPES	DESCRIPTION
DWUUsed	DWU used	Count	Max (default), Min, Avg	Represents a high-level representation of usage across the SQL pool. Measured by DWU limit * DWU percentage
DWUUsedPercent	DWU used percentage	Percent	Max (default), Min, Avg	Represents a high-level representation of usage across the SQL pool. Measured by taking the maximum between CPU percentage and Data IO percentage
ConnectionsBlockedByFirewall	Connections blocked by firewall	Count	Sum (default)	Count of connections blocked by firewall rules. Revisit access control policies for your SQL pool and monitor these connections if the count is high
AdaptiveCacheHitPercent	Adaptive cache hit percentage	Percent	Max (default), Min, Avg	Measures how well workloads are utilizing the adaptive cache. Use this metric with the cache hit percentage metric to determine whether to scale for additional capacity or rerun workloads to hydrate the cache
AdaptiveCacheUsedPercent	Adaptive cache used percentage	Percent	Max (default), Min, Avg	Measures how well workloads are utilizing the adaptive cache. Use this metric with the cache used percentage metric to determine whether to scale for additional capacity or rerun workloads to hydrate the cache
LocalTempDBUsedPercent	Local tempdb used percentage	Percent	Max (default), Min, Avg	Local tempdb utilization across all compute nodes - values are emitted every five minute
MemoryUsedPercent	Memory used percentage	Percent	Max (default), Min, Avg	Memory utilization across all nodes in the SQL pool

METRIC	DISPLAY NAME	UNIT	AGGREGATION TYPES	DESCRIPTION
CPUPercent	CPU used percentage	Percent	Max (default), Min, Avg	CPU utilization across all nodes in the SQL pool
Connections	Connections	Count	Sum (default)	Count of total logins to the SQL pool
ActiveQueries	Active queries	Count	Sum (default)	The active queries. Using this metric unfiltered and unsplit displays all active queries running on the system
QueuedQueries	Queued queries	Count	Sum (default)	Cumulative count of requests queued after the max concurrency limit was reached
WLGActiveQueries	Workload group active queries	Count	Sum (default)	The active queries within the workload group. Using this metric unfiltered and unsplit displays all active queries running on the system
WLGActiveQueriesTimeouts	Workload group query timeouts	Count	Sum (default)	Queries for the workload group that have timed out. Query timeouts reported by this metric are only once the query has started executing (it does not include wait time due to locking or resource waits)
WLGQueuedQueries	Workload group queued queries	Count	Sum (default)	Cumulative count of requests queued after the max concurrency limit was reached
WLGAllocationBySystemPercent	Workload group allocation by system percent	Percent	Max (default), Min, Avg, Sum	The percentage allocation of resources relative to the entire system

METRIC	DISPLAY NAME	UNIT	AGGREGATION TYPES	DESCRIPTION
WLGAllocationByEffectiveCapResourcePercent	Workload group allocation by max resource percent	Percent	Max (default), Min, Avg	Displays the percentage allocation of resources relative to the effective cap resource percent per workload group. This metric provides the effective utilization of the workload group
WLGEffectiveCapResourcePercent	Effective cap resource percent	Percent	Max (default), Min, Avg	The effective cap resource percent for the workload group. If there are other workload groups with min_percentage_resource > 0, the effective_cap_percentage_resource is lowered proportionally
WLGEffectiveMinResourcePercent	Effective min resource percent	Percent	Max (default), Min, Avg, Sum	The effective min resource percentage setting allowed considering the service level and the workload group settings. The effective min_percentage_resource can be adjusted higher on lower service levels

Apache Spark pool metrics

Here are some of the metrics emitted by Apache Spark pools:

METRIC	METRIC CATEGORY, DISPLAY NAME	UNIT	AGGREGATION TYPES	DESCRIPTION
BigDataPoolApplicationsEnded	Ended Apache Spark applications	Count	Sum (default)	Number of Apache Spark pool applications ended
BigDataPoolAllocatedCores	Number of vCores allocated to the Apache Spark pool	Count	Max (default), Min, Avg	Allocated vCores for an Apache Spark Pool
BigDataPoolAllocatedMemory	Amount of memory (GB) allocated to the Apache Spark pool	Count	Max (default), Min, Avg	Allocated Memory for Apache Spark Pool (GB)
BigDataPoolApplicationsActive	Active Apache Spark applications	Count	Max (default), Min, Avg	Number of active Apache Spark pool applications

Alerts

Sign in to the Azure portal and select **Monitor** > **Alerts** to create alerts.

Create Alerts

1. Select **+ New Alert rule** to create a new alert.
2. Define the **alert condition** to specify when the alert should fire.

> **NOTE**
> Make sure to select **All** in the **Filter by resource type** drop-down list.

3. Define the **alert details** to further specify how the alert should be configured.
4. Define the **action group** to determine who should receive the alerts (and how).

Logs

Workspace-level logs

Here are the logs emitted by Azure Synapse Analytics workspaces:

LOG ANALYTICS TABLE NAME	LOG CATEGORY NAME	DESCRIPTION
SynapseGatewayApiRequests	GatewayApiRequests	Azure Synapse gateway API requests.
SynapseRbacOperations	SynapseRbacOperations	Azure Synapse role-based access control (SRBAC) operations.
SynapseBuiltinSqlReqsEnded	BuiltinSqlReqsEnded	Azure Synapse built-in serverless SQL pool ended requests.
SynapseIntegrationPipelineRuns	IntegrationPipelineRuns	Azure Synapse integration pipeline runs.
SynapseIntegrationActivityRuns	IntegrationActivityRuns	Azure Synapse integration activity runs.
SynapseIntegrationTriggerRuns	IntegrationTriggerRuns	Azure Synapse integration trigger runs.

Dedicated SQL pool logs

Here are the logs emitted by dedicated SQL pools:

LOG ANALYTICS TABLE NAME	LOG CATEGORY NAME	DESCRIPTION
SynapseSqlPoolExecRequests	ExecRequests	Information about SQL requests/queries in an Azure Synapse dedicated SQL pool.
SynapseSqlPoolDmsWorkers	DmsWorkers	Information about workers completing DMS steps in an Azure Synapse dedicated SQL pool.

LOG ANALYTICS TABLE NAME	LOG CATEGORY NAME	DESCRIPTION
SynapseSqlPoolRequestSteps	RequestSteps	Information about request steps that compose a given SQL request/query in an Azure Synapse dedicated SQL pool.
SynapseSqlPoolSqlRequests	SqlRequests	Information about query distributions of the steps of SQL requests/queries in an Azure Synapse dedicated SQL pool.
SynapseSqlPoolWaits	Waits	Information about the wait states encountered during execution of a SQL request/query in an Azure Synapse dedicated SQL pool, including locks and waits on transmission queues.

For more information on these logs, see the following information:

- sys.dm_pdw_exec_requests
- sys.dm_pdw_request_steps
- sys.dm_pdw_dms_workers
- sys.dm_pdw_waits
- sys.dm_pdw_sql_requests

Apache Spark pool log

Here is the log emitted by Apache Spark pools:

LOG ANALYTICS TABLE NAME	LOG CATEGORY NAME	DESCRIPTION
SynapseBigDataPoolApplicationsEnded	BigDataPoolAppsEnded	Information about ended Apache Spark applications

Diagnostic settings

Use diagnostic settings to configure diagnostic logs for non-compute resources. The settings for a resource control have the following features:

- They specify where diagnostic logs are sent. Examples include an Azure storage account, an Azure event hub, or Monitor logs.
- They specify which log categories are sent.
- They specify how long each log category should be kept in a storage account.
- A retention of zero days means logs are kept forever. Otherwise, the value can be any number of days from 1 through 2,147,483,647.
- If retention policies are set but storing logs in a storage account is disabled, the retention policies have no effect. For example, this condition can happen when only Event Hubs or Monitor logs options are selected.
- Retention policies are applied per day. The boundary between days occurs at midnight Coordinated Universal Time (UTC). At the end of a day, logs from days that are beyond the retention policy are deleted. For example, if you have a retention policy of one day, at the beginning of today the logs from before yesterday are deleted.

With Azure Monitor diagnostic settings, you can route diagnostic logs for analysis to multiple different targets.

- **Storage account**: Save your diagnostic logs to a storage account for auditing or manual inspection. You can use the diagnostic settings to specify the retention time in days.
- **Event Hub**: Stream the logs to Azure Event Hubs. The logs become input to a partner service/custom

analytics solution like Power BI.

- **Log Analytics workspace**: Analyze the logs with Log Analytics. The Azure Synapse integration with Log Analytics is useful in the following scenarios:
  - You want to write complex queries on a rich set of metrics that are published by Azure Synapse to Log Analytics. You can create custom alerts on these queries via Azure Monitor.
  - You want to monitor across workspaces. You can route data from multiple workspaces to a single Log Analytics workspace.

You can also use a storage account or Event Hub namespace that isn't in the subscription of the resource that emits logs. The user who configures the setting must have appropriate Azure role-based access control (Azure RBAC) access to both subscriptions.

Configure diagnostic settings

Create or add diagnostic settings for your workspace, dedicated SQL pool, or Apache Spark pool.

1. In the portal, go to Monitor. Select **Settings** > **Diagnostic settings**.
2. Select the Synapse workspace, dedicated SQL pool, or Apache Spark pool for which you want to create a diagnostic setting.
3. If no diagnostic settings exist on the selected workspace, you're prompted to create a setting. Select **Turn on diagnostics**.

   If there are existing diagnostic settings on the workspace, you will see a list of settings already configured on the resource. Select **Add diagnostic setting**.
4. Give your setting a name, select **Send to Log Analytics**, and then select a workspace from **Log Analytics workspace**.

   > **NOTE**
   > Because an Azure log table can't have more than 500 columns, we **highly recommended** you select *Resource-Specific mode*. For more information, see AzureDiagnostics Logs reference.
5. Select **Save**.

After a few moments, the new setting appears in your list of settings for your workspace, dedicated SQL pool, or Apache Spark pool. Diagnostic logs are streamed to that workspace as soon as new event data are generated. Up to 15 minutes might elapse between when an event is emitted and when it appears in Log Analytics.

Next steps

For more information on monitoring pipeline runs, see the Monitor pipeline runs in Synapse Studio article.

For more information on monitoring Apache Spark applications, see the Monitor Apache Spark applications in Synapse Studio article.

For more information on monitoring SQL requests, see the Monitor SQL requests in Synapse Studio article.

How to set up access control for your Azure Synapse workspace

8/19/2022 • 10 minutes to read • Edit Online

This article teaches you how to control access to a Microsoft Azure Synapse workspace. We'll use a combination of Azure roles, Azure Synapse roles, SQL permissions, and Git permissions to achieve this.

In this guide, you'll set up a workspace and configure a basic access control system. You can use this information in many types of Synapse projects. You'll also find advanced options for finer-grained control should you need it.

Synapse access control can be simplified by aligning roles and personas in your organization with security groups. This enables you to manage access to security groups simply by adding and removing users.

Before you begin this walkthrough, read the Azure Synapse access control overview to familiarize yourself with access control mechanisms used by Synapse Analytics.

Access control mechanisms

> **NOTE**
> The approach in this guide is to create security groups. When you assign roles to these security groups, you only need to manage memberships within those groups to control access to workspaces.

To secure a Synapse workspace, you'll configure the following items:

- **Security Groups**, to group users with similar access requirements.
- **Azure roles**, to control who can create and manage SQL pools, Apache Spark pools and Integration runtimes, and access ADLS Gen2 storage.
- **Synapse roles**, to control access to published code artifacts, use of Apache Spark compute resources and integration runtimes.
- **SQL permissions**, to control administrative and data plane access to SQL pools.
- **Git permissions**, to control who can access code artifacts in source control if you configure Git-support for workspaces.

Steps to secure a Synapse workspace

This document uses standard names to simplify instructions. Replace them with names of your choice.

SETTING	STANDARD NAME	DESCRIPTION
Synapse workspace	`workspace1`	The name that the Azure Synapse workspace will have.
ADLSGEN2 account	`storage1`	The ADLS account to use with your workspace.
Container	`container1`	The container in STG1 that the workspace will use by default.

SETTING	STANDARD NAME	DESCRIPTION
Active directory tenant	`contoso`	the active directory tenant name.

STEP 1: Set up security groups

> **NOTE**
> During the preview, you were encouraged to create security groups and to map them to Azure Synapse **Synapse SQL Administrator** and **Synapse Apache Spark Administrator** roles. With the introduction of new finer-grained Synapse RBAC roles and scopes, you are now encouraged to use newer options to control access to your workspace. They give you greater configuration flexibility and they acknowledge that developers often use a mix of SQL and Spark to create analytics applications. So developers may need access to individual resources rather than an entire workspace. Learn more about Synapse RBAC.

Create the following security groups for your workspace:

- `workspace1_SynapseAdministrators`, for users who need complete control over a workspace. Add yourself to this security group, at least initially.
- `workspace1_SynapseContributors`, for developers who need to develop, debug, and publish code to a service.
- `workspace1_SynapseComputeOperators`, for users who need to manage and monitor Apache Spark pools and Integration runtimes.
- `workspace1_SynapseCredentialUsers`, for users who need to debug and run orchestration pipelines using workspace MSI (managed service identity) credentials and cancel pipeline runs.

You'll assign Synapse roles to these groups at the workspace scope shortly.

Also create this security group:

- `workspace1_SQLAdmins`, group for users who need SQL Active Directory Admin authority, within SQL pools in the workspace.

The `workspace1_SQLAdmins` group to configure SQL permissions when you create SQL pools.

These five groups are sufficient for a basic setup. Later, you can add security groups to handle users who need more specialized access or restrict access to individual resources only.

> **NOTE**
> - Learn how to create a security group in Create a basic group and add members using Azure Active Directory.
> - Learn how to add a security group from another security group in Add or remove a group from another group using Azure Active Directory.

> **TIP**
> Individual Synapse users can use Azure Active Directory in the Azure portal to view their group memberships. This allows them to determine which roles they've been granted.

STEP 2: Prepare your ADLS Gen2 storage account

Synapse workspaces use default storage containers for:

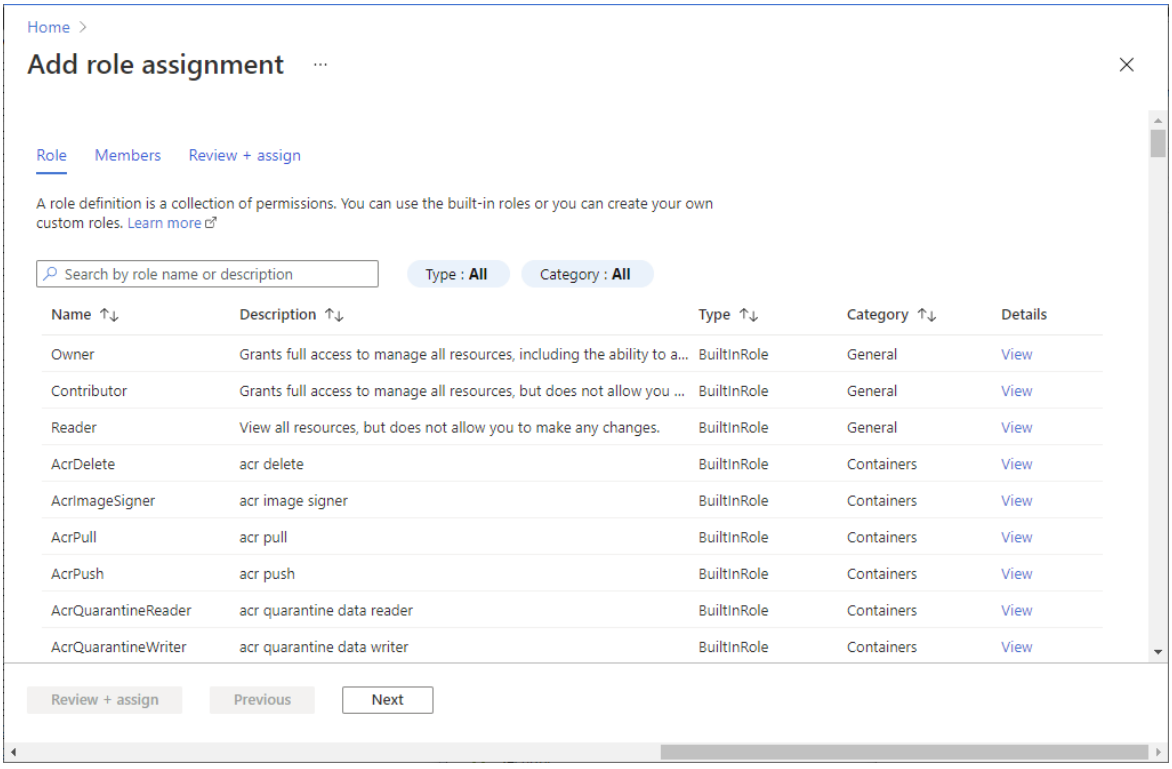
- Storage of backing data files for Spark tables

- Execution logs for Spark jobs
- Management of libraries that you choose to install

Identify the following information about your storage:

- The ADLS Gen2 account to use for your workspace. This document calls it `storage1`. `storage1` is considered the "primary" storage account for your workspace.
- The container inside `workspace1` that your Synapse workspace will use by default. This document calls it `container1`.
- Select **Access control (IAM)**.
- Select **Add** > **Add role assignment** to open the Add role assignment page.
- Assign the following role. For detailed steps, see Assign Azure roles using the Azure portal.

SETTING	VALUE
Role	Storage Blob Data Contributor
Assign access to	SERVICEPRINCIPAL
Members	workspace1_SynapseAdministrators, workspace1_SynapseContributors, and workspace1_SynapseComputeOperators



STEP 3: Create and configure your Synapse workspace

In Azure portal, create a Synapse workspace:

- Select your subscription
- Select or create a resource group for which you have an Azure **Owner** role.
- Name the workspace `workspace1`

- Choose `storage1` for the Storage account
- Choose `container1` for the container that is being used as the "filesystem".
- Open WS1 in Synapse Studio
- In Synapse Studio, navigate to **Manage** > **Access Control**. In **workspace scope**, assign Synapse roles to security groups as follows:
  - Assign the **Synapse Administrator** role to `workspace1_SynapseAdministrators`
  - Assign the **Synapse Contributor** role to `workspace1_SynapseContributors`
  - Assign the **Synapse Compute Operator** role to `workspace1_SynapseComputeOperators`

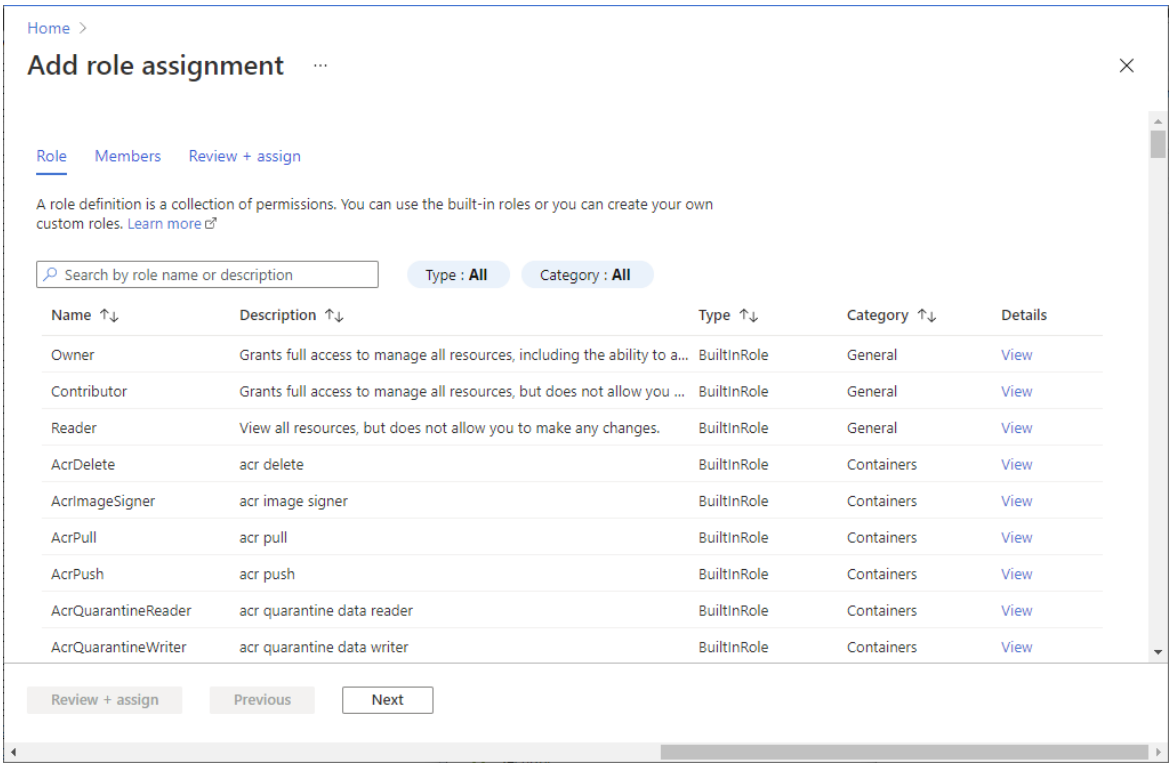
STEP 4: Grant the workspace MSI access to the default storage container

To run pipelines and perform system tasks, Azure Synapse requires managed service identity (MSI) to have access to `container1` in the default ADLS Gen2 account, for the workspace. For more information, see Azure Synapse workspace managed identity.

- Open Azure portal
- Locate the storage account, `storage1`, and then `container1`.
- Select **Access control (IAM)**.
- To open the **Add role assignment** page, select **Add** > **Add role assignment** .
- Assign the following role. For detailed steps, see Assign Azure roles using the Azure portal.

SETTING	VALUE
Role	Storage Blob Data Contributor
Assign access to	MANAGEDIDENTITY
Members	managed identity name

> **NOTE**
> The managed identity name is also the workspace name.

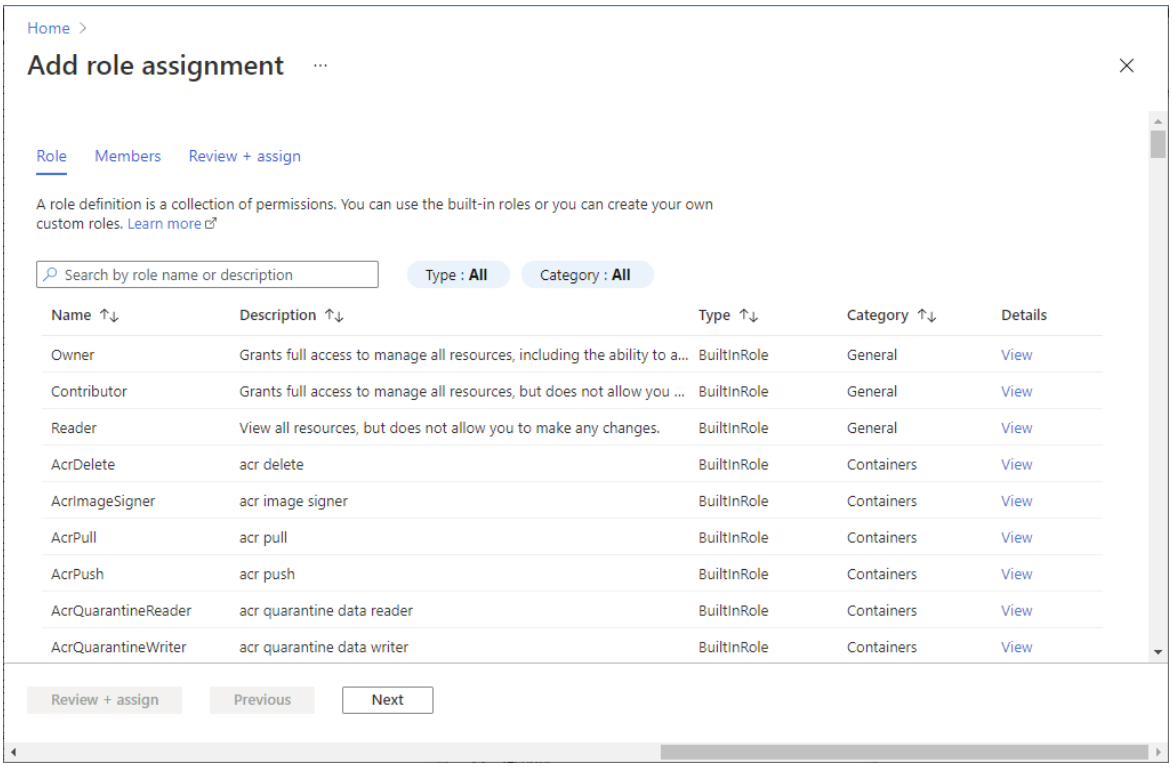


STEP 5: Grant Synapse administrators an Azure Contributor role for the workspace

To create SQL pools, Apache Spark pools and Integration runtimes, users need an Azure Contributor role for the workspace, at minimum. A Contributor role also allows users to manage resources, including pausing and scaling. To use Azure portal or Synapse Studio to create SQL pools, Apache Spark pools and Integration runtimes, you need a Contributor role at the resource group level.

- Open Azure portal
- Locate the workspace, `workspace1`
- Select **Access control (IAM)**.
- To open the **Add role assignment** page, select **Add** > **Add role assignment**.
- Assign the following role. For detailed steps, see Assign Azure roles using the Azure portal.

SETTING	VALUE
Role	Contributor
Assign access to	SERVICEPRINCIPAL
Members	workspace1_SynapseAdministrators



STEP 6: Assign an SQL Active Directory Admin role

The *workspace creator* is automatically assigned as *SQL Active Directory Admin* for the workspace. Only a single user or a group can be granted this role. In this step, you assign the SQL Active Directory Admin for the workspace to the `workspace1_SQLAdmins` security group. This gives the group highly privileged admin access to all SQL pools and databases in the workspace.

- Open Azure portal
- Navigate to `workspace1`
- Under **Settings**, select **Azure Active Directory**
- Select **Set admin** and choose `workspace1_SQLAdmins`

> **NOTE**
> Step 6 is optional. You might choose to grant the `workspace1_SQLAdmins` group a less privileged role. To assign `db_owner` or other SQL roles, you must run scripts on each SQL database.

STEP 7: Grant access to SQL pools

The Synapse Administrator is by default given the SQL `db_owner` role for serverless SQL pools in the workspace as well.

Access to SQL pools for other users is controlled by SQL permissions. Assigning SQL permissions requires SQL scripts to be run on each SQL database post-creation. The following are examples that require you to run these scripts:

1. To grant users access to the serverless SQL pool, 'Built-in', and its databases.
2. To grant users access to dedicated SQL pool databases. Example SQL scripts are included later in this article.
3. To grant access to a dedicated SQL pool database, scripts can be run by the workspace creator or any member of the `workspace1_SynapseAdministrators` group.

4. To grant access to the serverless SQL pool, 'Built-in', scripts can be run by any member of the `workspace1_SQLAdmins` group or the `workspace1_SynapseAdministrators` group.

> **TIP**
> You can grant access to all SQL databases by taking the following steps for **each** SQL pool. Section Configure-Workspace-scoped permissions is an exception to the rule and it allows you to assign a user a sysadmin role at the workspace level.

STEP 7.1: Serverless SQL pool, Built-in

You can use the script examples in this section to give users permission to access an individual database or all databases in the serverless SQL pool, `Built-in`.

> **NOTE**
> In the script examples, replace *alias* with the alias of the user or group being granted access. Replace *domain* with the company domain you are using.

Configure Database-scoped permissions

You can grant users access to a **single** serverless SQL database with the steps outlined in this example:

1. Create a login. Change to the `master` database context.

```
--In the master database
CREATE LOGIN [alias@domain.com] FROM EXTERNAL PROVIDER;
```

2. Create user in your database. Change context to your database.

```
-- In your database
CREATE USER alias FROM LOGIN [alias@domain.com];
```

3. Add user as a member of the specified role in your database (in this case, the **db_owner** role).

```
ALTER ROLE db_owner ADD member alias; -- Type USER name from step 2
```

Configure Workspace-scoped permissions

You can grant full access to **all** serverless SQL pools in the workspace. Run the script in this example in the `master` database:

```
CREATE LOGIN [alias@domain.com] FROM EXTERNAL PROVIDER;
ALTER SERVER ROLE sysadmin ADD MEMBER [alias@domain.com];
```

STEP 7.2: configure Dedicated SQL pools

You can grant access to a **single**, dedicated, SQL pool database. Use these steps in the Azure Synapse SQL script editor:

1. Create a user in the database by running the following commands. Select the target database in the *Connect to* dropdown:

```
--Create user in the database
CREATE USER [<alias@domain.com>] FROM EXTERNAL PROVIDER;
```

2. Grant the user a role to access the database:

```
--Grant role to the user in the database
EXEC sp_addrolemember 'db_owner', '<alias@domain.com>';
```

> **IMPORTANT**
> **db_datareader** and **db_datawriter** database roles can provide read/write permission when you do not want to give **db_owner** permissions. However, **db_owner** permission is necessary for Spark users to read and write directly from Spark into or from an SQL pool.

You can run queries to confirm that serverless SQL pools can query storage accounts, after you have created your users.

STEP 8: Add users to security groups

The initial configuration for your access control system is now complete.

You can now add and remove users to the security groups you've set up, to manage access to them. You can manually assign users to Azure Synapse roles, but this sets permissions inconsistently. Instead, only add or remove users to your security groups.

STEP 9: Network security

As a final step to secure your workspace, you should secure network access, using the workspace firewall.

- With and without a managed virtual network, you can connect to your workspace from public networks. For more information, see Connectivity Settings.
- Access from public networks can be controlled by enabling the public network access feature or the workspace firewall.
- Alternatively, you can connect to your workspace using a managed private endpoint and private Link. Azure Synapse workspaces without the Azure Synapse Analytics Managed Virtual Network do not have the ability to connect via managed private endpoints.

STEP 10: Completion

Your workspace is now fully configured and secured.

Supporting more advanced scenarios

This guide has focused on setting up a basic access control system. You can support more advanced scenarios by creating additional security groups and assigning these groups more granular roles at more specific scopes. Consider the following cases:

Enable Git-support for the workspace for more advanced development scenarios including CI/CD. While in Git mode, Git permissions and Synapse RBAC will determine whether a user can commit changes to their working branch. Publishing to the service only takes place from the collaboration branch. Consider creating a security group for developers who need to develop and debug updates in a working branch but don't need to publish changes to the live service.

Restrict developer access to specific resources. Create additional finer-grained security groups for developers who need access only to specific resources. Assign these groups appropriate Azure Synapse roles that are scoped to specific Spark pools, Integration runtimes, or credentials.

Restrict operators from accessing code artifacts. Create security groups for operators who need to monitor operational status of Synapse compute resources and view logs but who don't need access to code or to publish updates to the service. Assign these groups the Compute Operator role scoped to specific Spark pools and Integration runtimes.

Disable local authentication. By allowing only Azure Active Directory authentication, you can centrally manage access to Azure Synapse resources, such as SQL pools. Local authentication for all resources within the workspace can be disabled during or after workspace creation. For more information on Azure AD-only authentication, see Disabling local authentication in Azure Synapse Analytics.

Next steps

- Learn how to manage Azure Synapse RBAC role assignments
- Create a Synapse Workspace

How to review Synapse RBAC role assignments

8/19/2022 • 2 minutes to read • Edit Online

Synapse RBAC roles are used to assign permissions to users, groups, and other security principals to enable access and use of Synapse resources. For more information, see What is Synapse role-based access control (RBAC)?

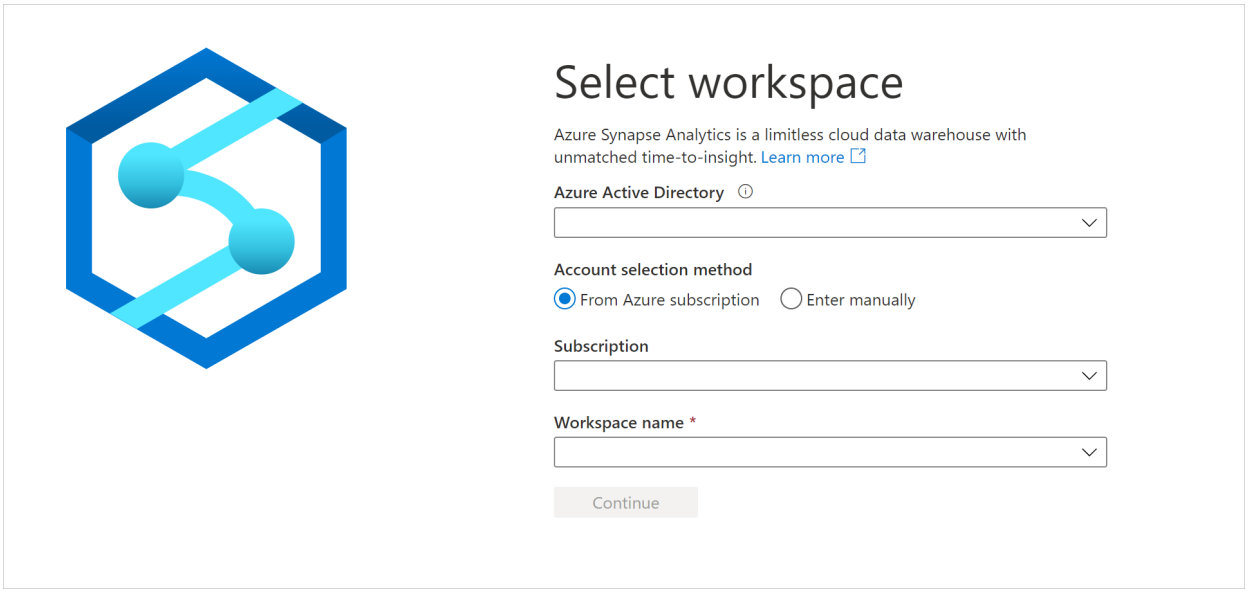
This article explains how to review the current role assignments for a workspace.

With any Synapse RBAC role, you can list Synapse RBAC role assignments for all scopes, including assignments for objects you don't have access to. Only a Synapse Administrator can grant Synapse RBAC access.

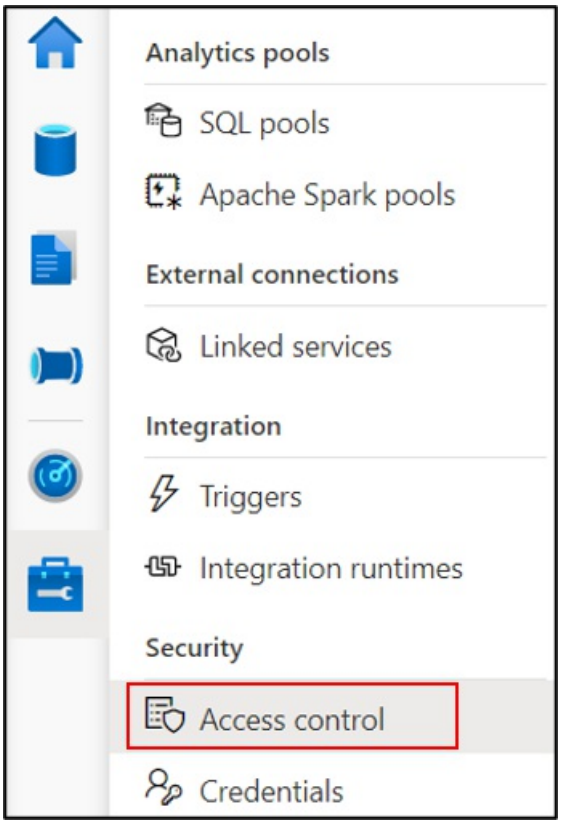
> **NOTE**
> A guest users (users from a different AD tenant) can also see and manage role assignment after being assigned the Synapse Administrator role.

Open Synapse Studio

To review role assignments, first open the Synapse Studio and select your workspace. To sign into your workspace, there are two **Account selection methods**. One is from **Azure subscription**, the other is from **Enter manually**. If you have the Synapse Azure role or higher level Azure roles, you can use both methods to log into the workspace. If you don't have the related Azure roles, and you were granted as the Synapse RBAC role, **Enter manually** is the only way to log into the workspace.

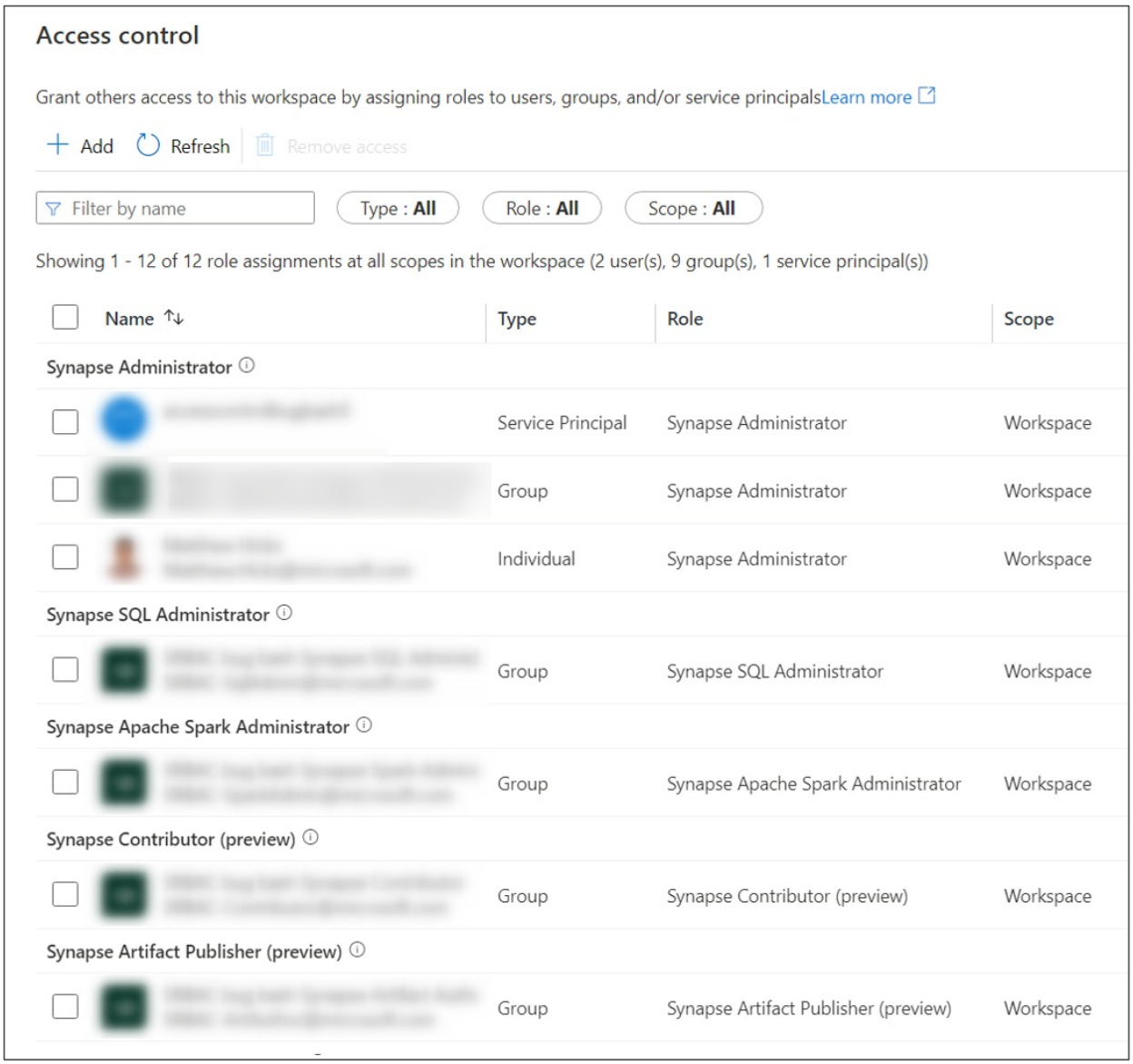


Once you've opened your workspace, select the **Manage** hub on the left, then expand the **Security** section and select **Access control**.



Review workspace role assignments

The Access control screen lists all current role assignments for the workspace, grouped by role. Each assignment includes the principal name, principal type, role, and its scope.



If a principal is assigned the same role at different scopes, you'll see multiple assignments for the principal, one for each scope.

If a role is assigned to a security group, you'll see the roles explicitly assigned to the group but not roles inherited from parent groups.

You can filter the list by principal name or email, and selectively filter the object types, roles, and scopes. enter your name or email alias in the Name filter to see roles assigned to you. Only a Synapse Administrator can change your roles.

> **IMPORTANT**
> If you are directly or indirectly a member of a group that is assigned roles, you may have permissions that are not shown.

> **TIP**
> You can find your group memberships using Azure Active Directory in the Azure portal.

If you create a new workspace, you and the workspace MSI service principal are automatically given the Synapse Administrator role at workspace scope.

Next steps

Learn how to manage Synapse RBAC role assignments.

Learn which role you need to do specific tasks

How to manage Synapse RBAC role assignments in Synapse Studio

8/19/2022 • 2 minutes to read • Edit Online

Synapse RBAC uses roles to assign permissions to users, groups, and other security principals to enable access and use of Synapse resources and code artifacts. For more information, see What is Synapse role-based access control (RBAC)?

This article shows how to add and delete Synapse RBAC role assignments.

NOTE

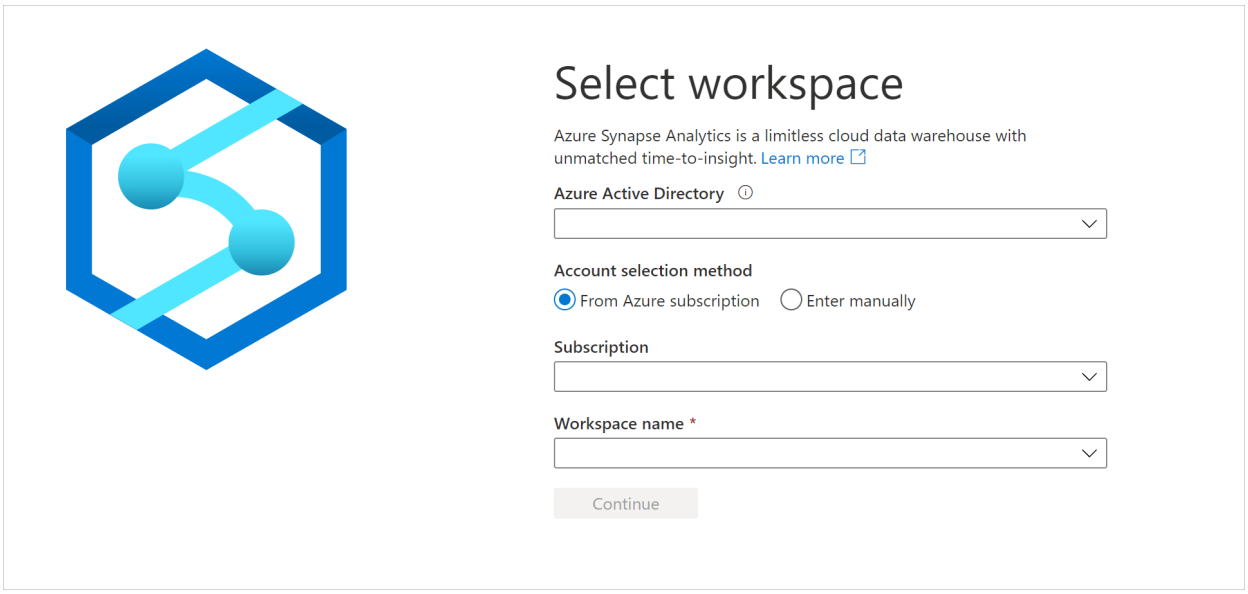
- To manage Synapse RBAC role assignments you need to have the Synapse Administrator role on the workspace or at a lower-level scope that includes the objects you want to manage. If you are a Synapse Administrator on the workspace you can grant access to all objects in the workspace.
- **Guest users** from a different AD tenant can also see and manage role assignments after being assigned the Synapse Administrator role.
- To help you regain access to a workspace in the event that no Synapse Administrators are assigned or available to you, users with permissions to manage **Azure RBAC** role assignments on the workspace can also manage **Synapse RBAC** role assignments, allowing the addition of Synapse Administrator or other Synapse role assignments.
- Access to SQL pools is managed using SQL permissions. With the exception of the Synapse Administrator and Synapse SQL Administrator roles, Synapse RBAC roles do not grant access to SQL pools.

IMPORTANT

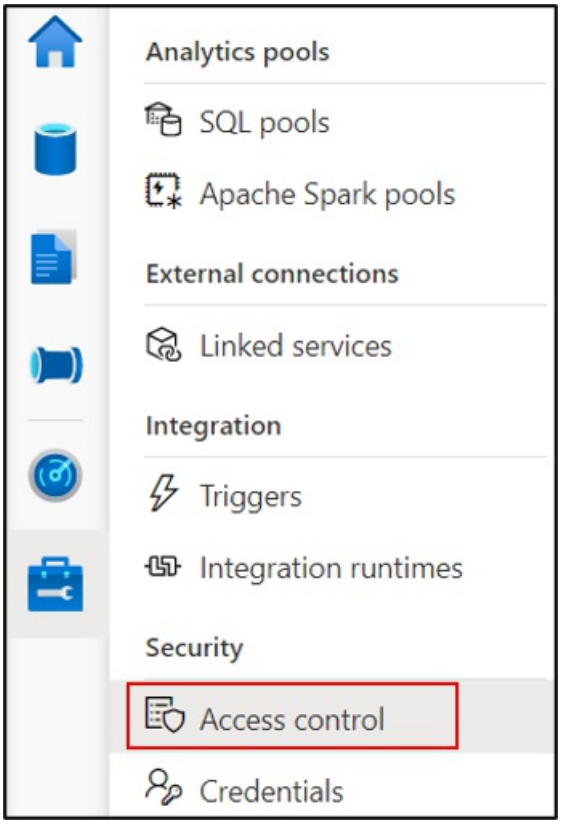
- Changes made to Synapse RBAC role assignments may take 2-5 minutes to take effect.
- If you are managing Synapse RBAC permissions by modifying membership of security groups, then changes to membership are managed using Azure Active Directory. Changes to group memberships may take 10-15 minutes or longer to take effect.

Open Synapse Studio

To assign a role to a user, group, service principal, or managed identity, first open the Synapse Studio and log into your workspace.



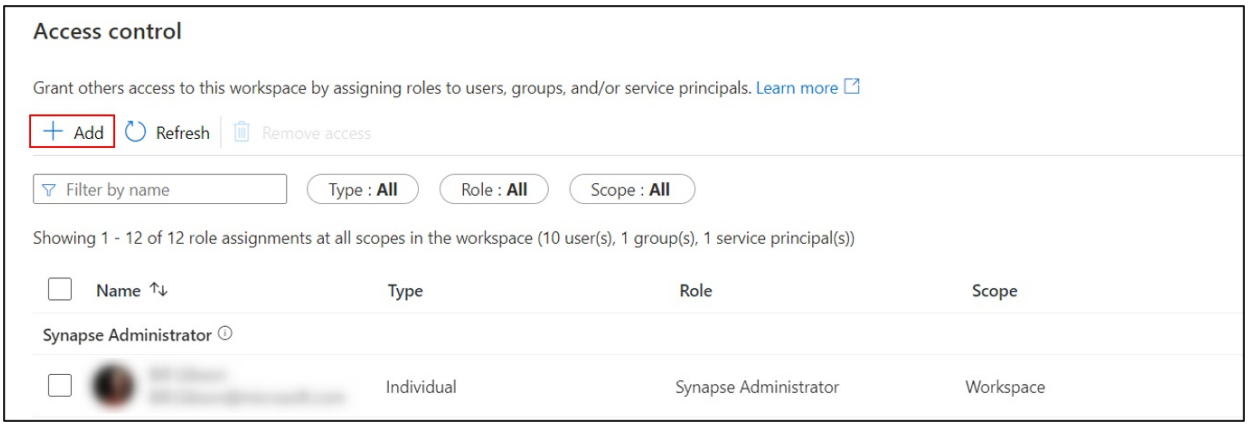
Once you've opened your workspace, expand the **Security** section on the left and select **Access control**.



The Access control screen lists current role assignments. You can filter the list by principal name or email, and selectively filter the object types, roles, or scopes included. From this screen, you can add or remove role assignments.

Add a Synapse role assignment

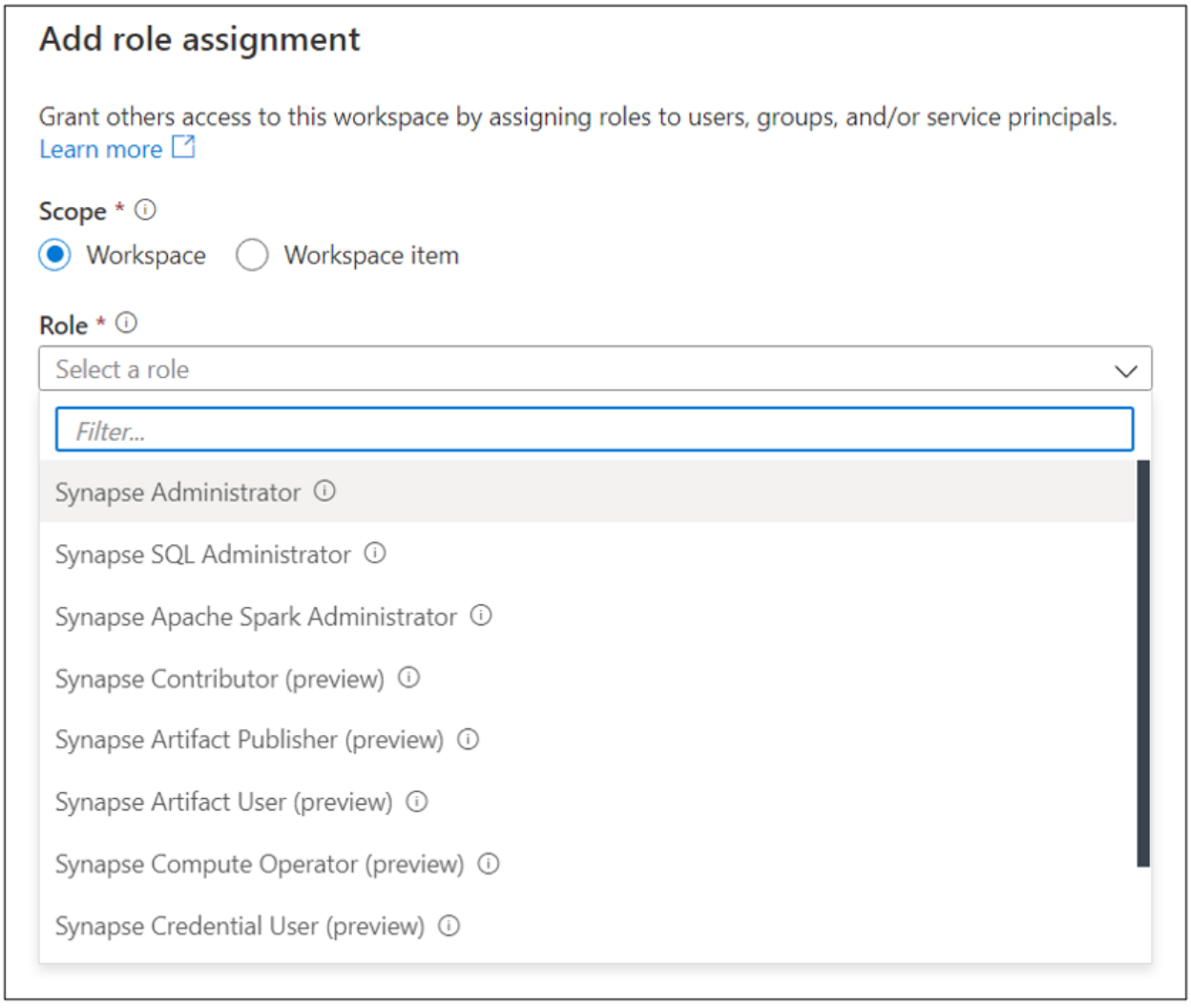
On the access control screen, select **+Add** to create a new role assignment



On the Add role assignment tab, you can create role assignments at workspace scope or workspace item scope.

Add workspace-scoped role assignment

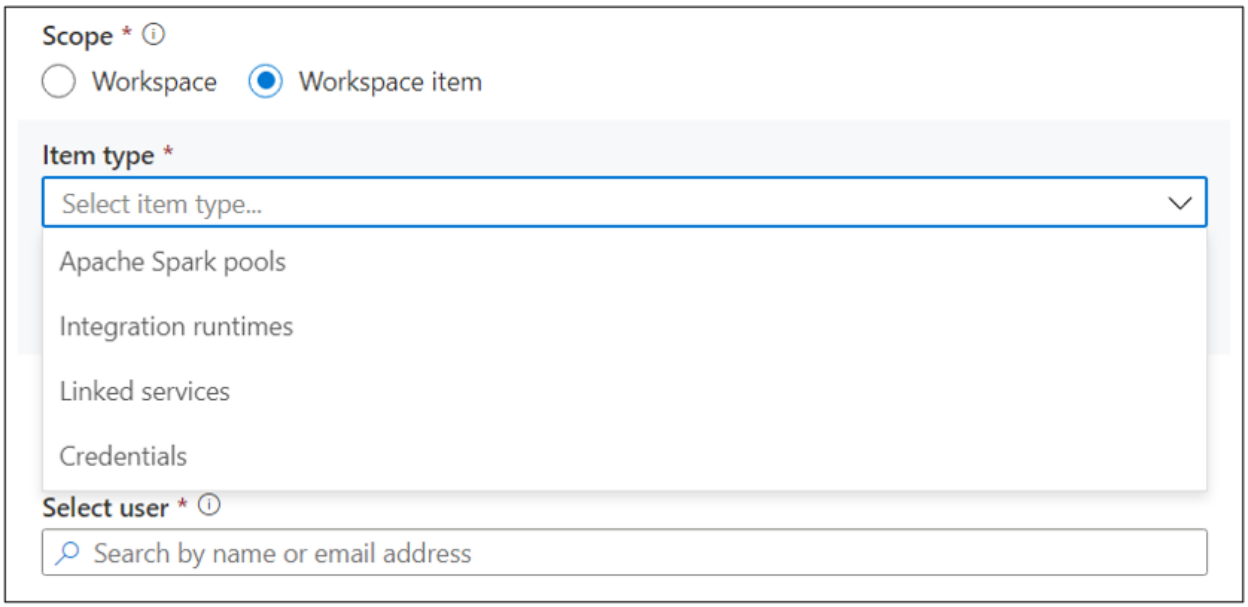
First, select **Workspace** as the scope, then select the **Synapse RBAC role**. Select the **principal(s)** to be assigned the role, then create the role assignment(s).



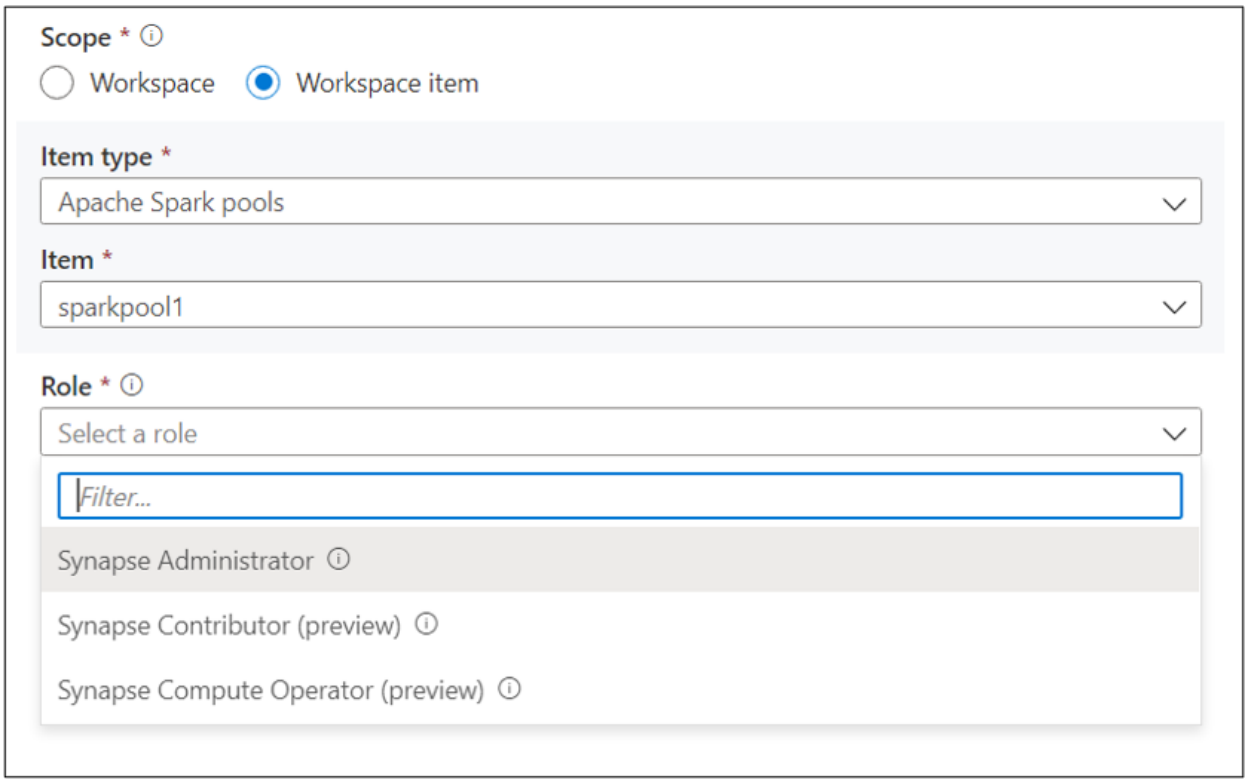
The assigned role will apply for all applicable objects in the workspace.

Add workspace item-scoped role assignment

To assign a role at a finer-grained scope, select **Workspace item** as the scope, then select the scoping **Item type**.



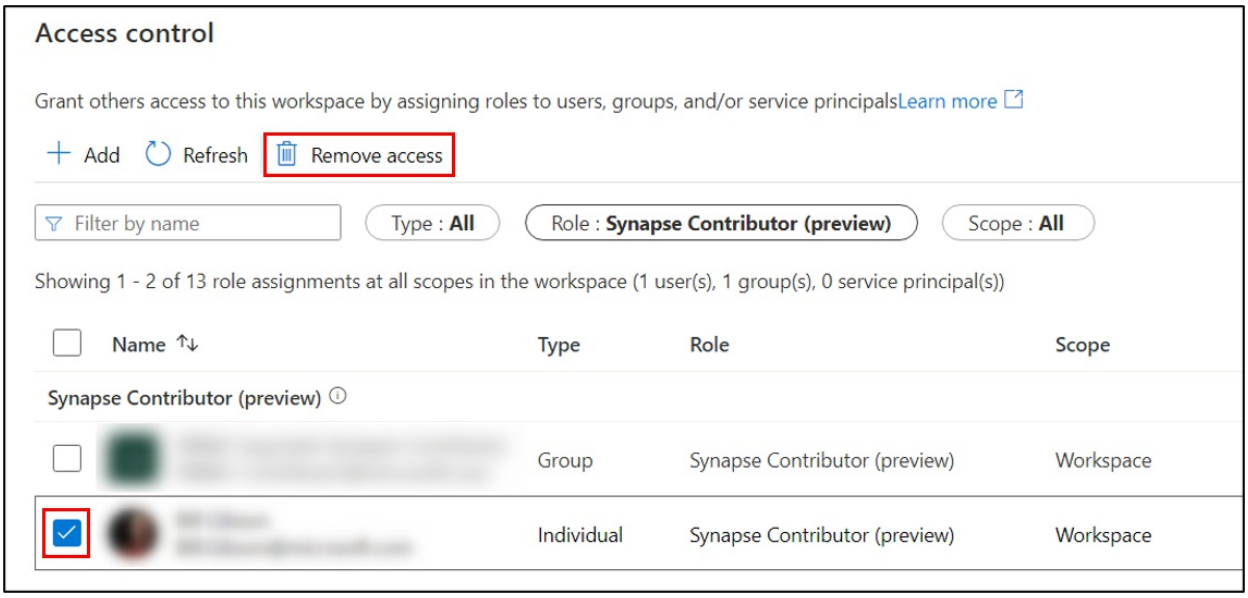
Select the specific **item** to be used as the scope, then select the **role** to be assigned from the drop-down. The drop-down lists only those roles that are valid for the selected item type. Learn more.



Then **select the principal(s)** to which the role is to be assigned. You can iteratively select multiple principals. Select **Apply** to create the role assignment(s).

Remove a Synapse RBAC role assignment

To revoke Synapse RBAC access, you remove the appropriate role assignments. On the Access control screen, use the filters to locate the role assignment(s) to be removed. Check the role assignments and then select **Remove access**.



Remember that changes to role assignments will take 2-5 minutes to take effect.

Next steps

Understand the Synapse RBAC roles required to perform common tasks

Grant permissions to workspace managed identity

8/19/2022 • 2 minutes to read • Edit Online

This article teaches you how to grant permissions to the managed identity in Azure synapse workspace. Permissions, in turn, allow access to dedicated SQL pools in the workspace and ADLS Gen2 storage account through the Azure portal.

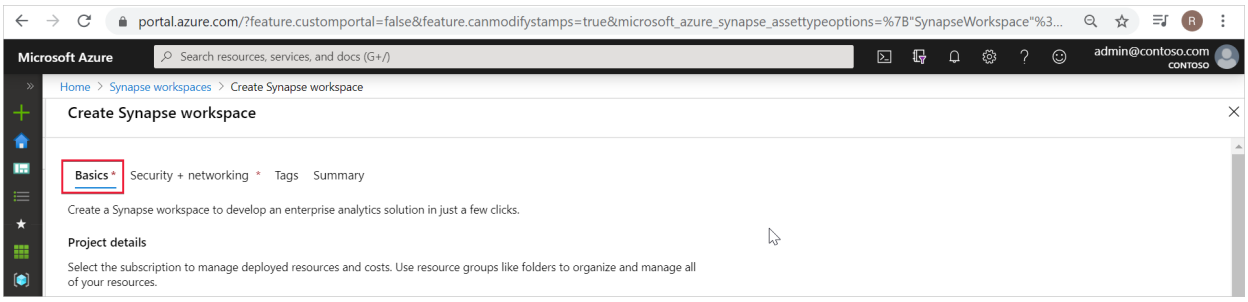
> **NOTE**
>
> This workspace managed identity will be referred to as managed identity through the rest of this document.

Grant the managed identity permissions to ADLS Gen2 storage account

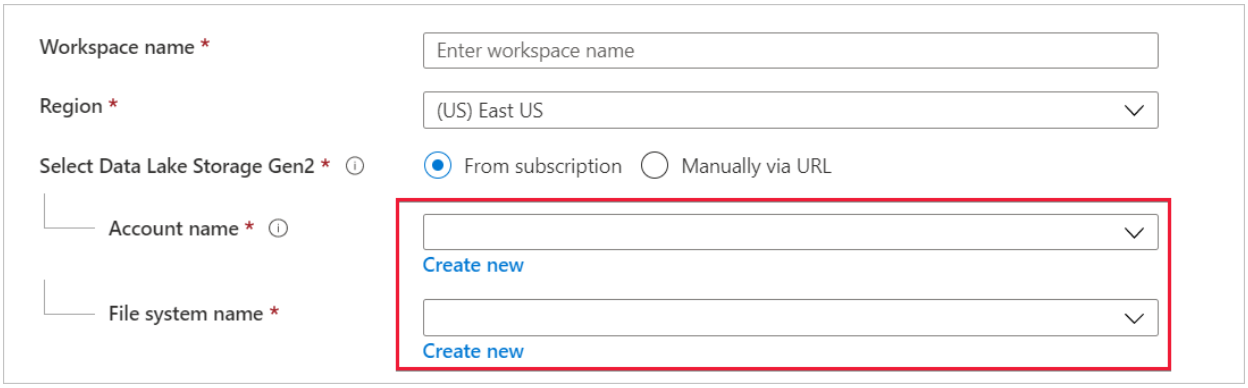
An ADLS Gen2 storage account is required to create an Azure Synapse workspace. To successfully launch Spark pools in Azure Synapse workspace, the Azure Synapse managed identity needs the *Storage Blob Data Contributor* role on this storage account . Pipeline orchestration in Azure Synapse also benefits from this role.

Grant permissions to managed identity during workspace creation

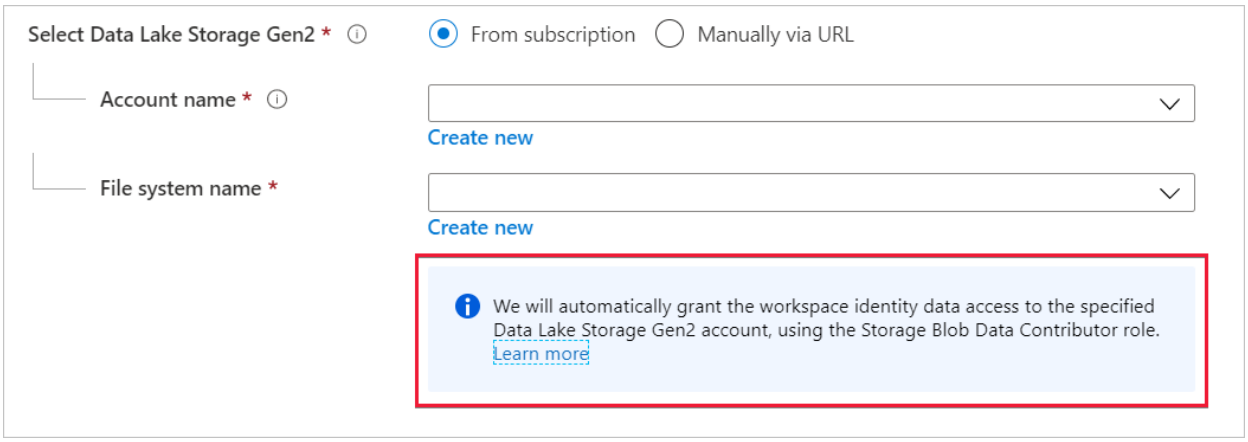
Azure Synapse will attempt to grant the Storage Blob Data Contributor role to the managed identity after you create the Azure Synapse workspace using Azure portal. You provide the ADLS Gen2 storage account details in the **Basics** tab.



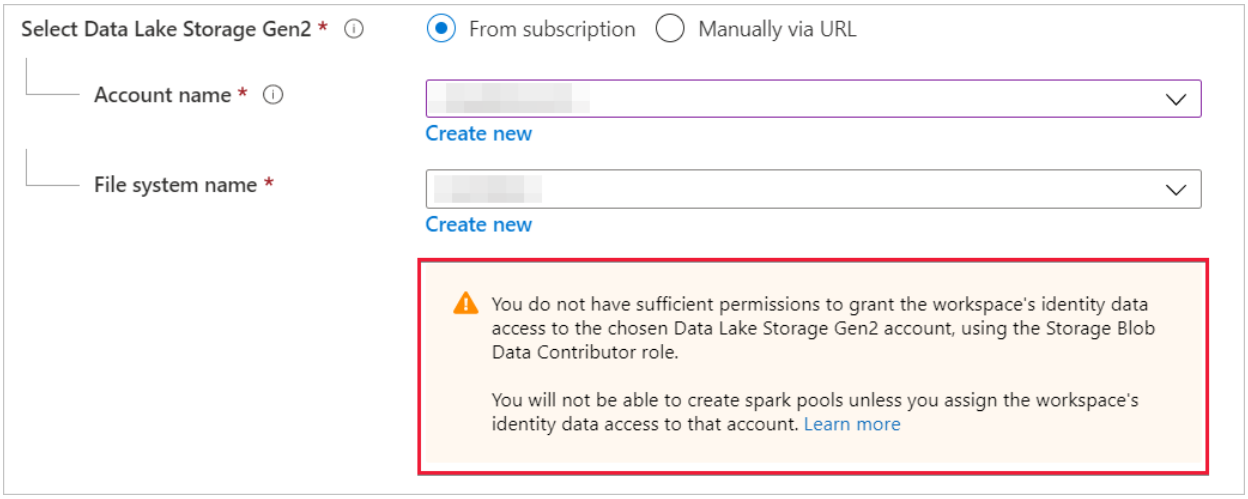
Choose the ADLS Gen2 storage account and filesystem in **Account name** and **File system name**.



If the workspace creator is also **Owner** of the ADLS Gen2 storage account, then Azure Synapse will assign the *Storage Blob Data Contributor* role to the managed identity. You'll see the following message below the storage account details that you entered.



If the workspace creator isn't the owner of the ADLS Gen2 storage account, then Azure Synapse doesn't assign the *Storage Blob Data Contributor* role to the managed identity. The message appearing below the storage account details notifies the workspace creator that they don't have sufficient permissions to grant the *Storage Blob Data Contributor* role to the managed identity.



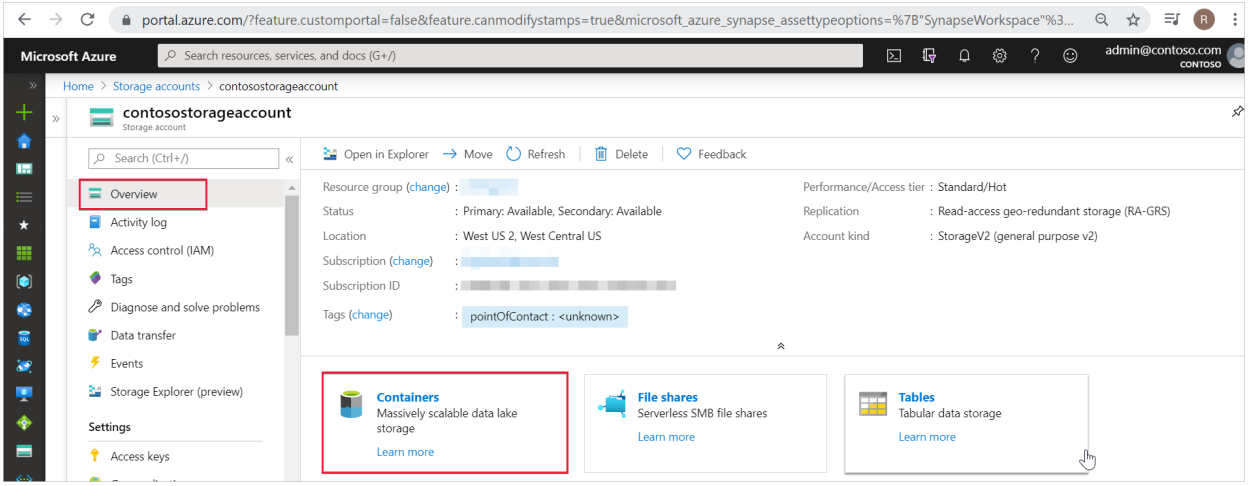
As the message states, you can't create Spark pools unless the *Storage Blob Data Contributor* is assigned to the managed identity.

Grant permissions to managed identity after workspace creation

During workspace creation, if you don't assign the *Storage Blob Data contributor* to the managed identity, then the **Owner** of the ADLS Gen2 storage account manually assigns that role to the identity. The following steps will help you to accomplish manual assignment.

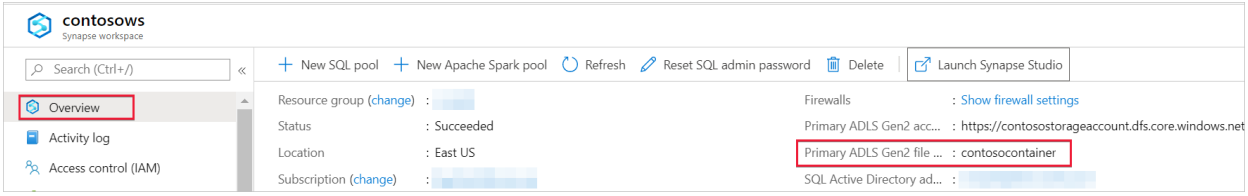
Step 1: Navigate to the ADLS Gen2 storage account in Azure portal

In Azure portal, open the ADLS Gen2 storage account and select **Overview** from the left navigation. You'll only need to assign The *Storage Blob Data Contributor* role at the container or filesystem level. Select **Containers**.

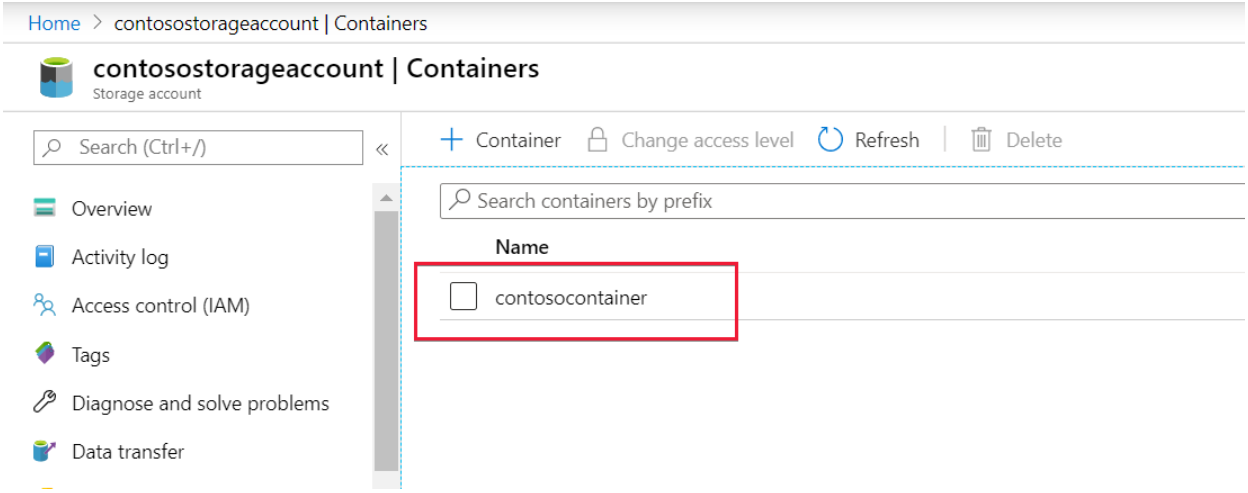


Step 2: Select the container

The managed identity should have data access to the container (file system) that was provided when the workspace was created. You can find this container or file system in Azure portal. Open the Azure Synapse workspace in Azure portal and select the **Overview** tab from the left navigation.



Select that same container or file system to grant the *Storage Blob Data Contributor* role to the managed identity.

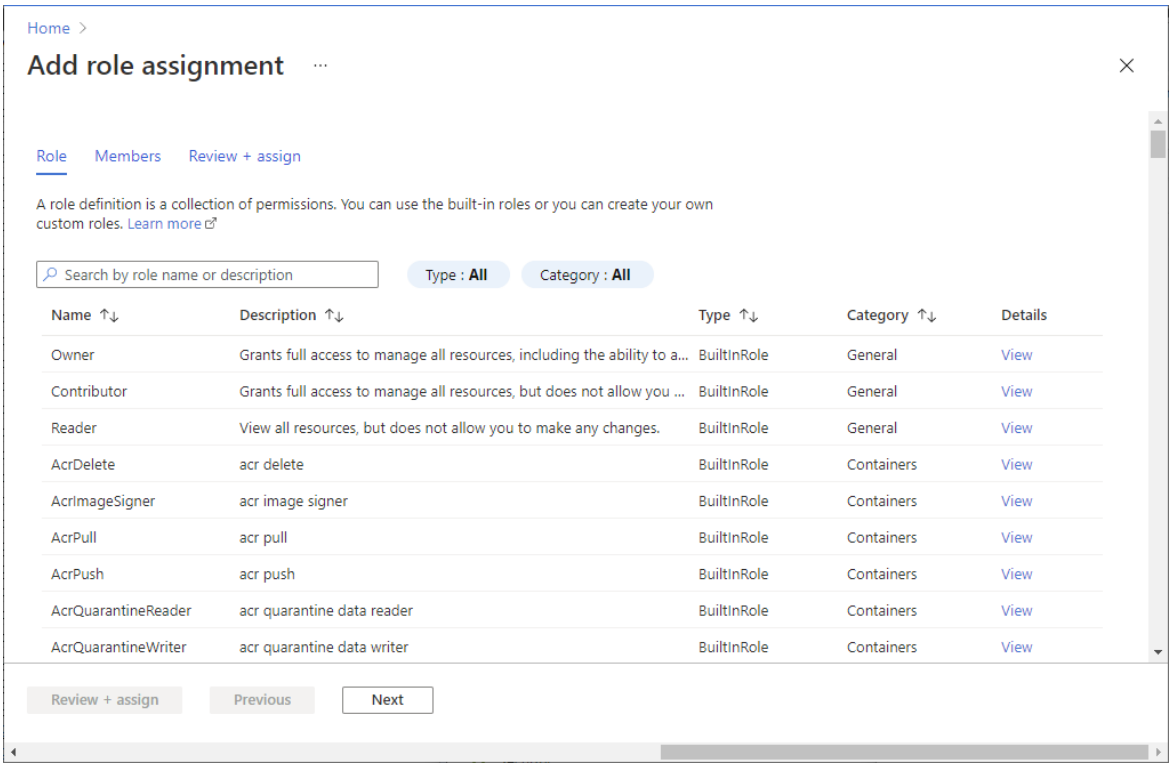


Step 3: Open Access control and add role assignment

1. Select **Access control (IAM)**.
2. Select **Add** > **Add role assignment** to open the Add role assignment page.
3. Assign the following role. For detailed steps, see Assign Azure roles using the Azure portal.

SETTING	VALUE
Role	Storage Blob Contributor
Assign access to	MANAGEDIDENTITY
Members	managed identity name

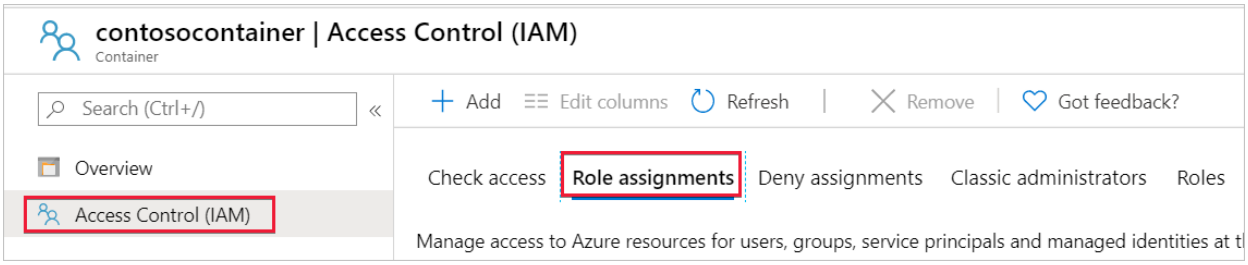
> **NOTE**
> The managed identity name is also the workspace name.



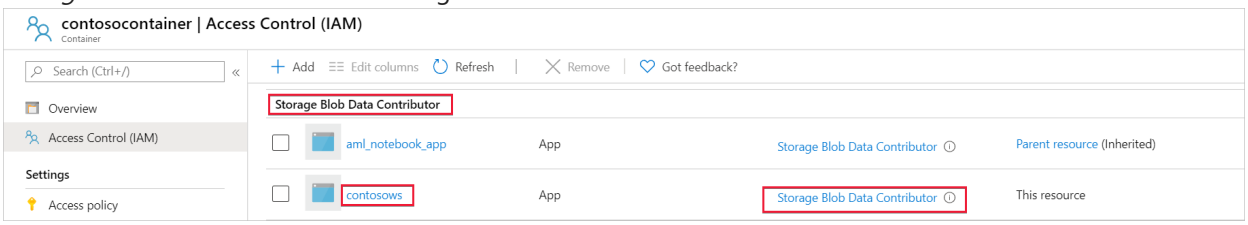
4. Select **Save** to add the role assignment.

Step 4: Verify that the Storage Blob Data Contributor role is assigned to the managed identity

Select **Access Control(IAM)** and then select **Role assignments**.



You should see your managed identity listed under the **Storage Blob Data Contributor** section with the *Storage Blob Data Contributor* role assigned to it.



Next steps

Learn more about Workspace managed identity

Create a workspace with data exfiltration protection enabled

8/19/2022 • 2 minutes to read • Edit Online

This article describes how to create a workspace with data exfiltration protection enabled and how to manage the approved Azure AD tenants for this workspace.

> **NOTE**
> You cannot change the workspace configuration for managed virtual network and data exfiltration protection after the workspace is created.

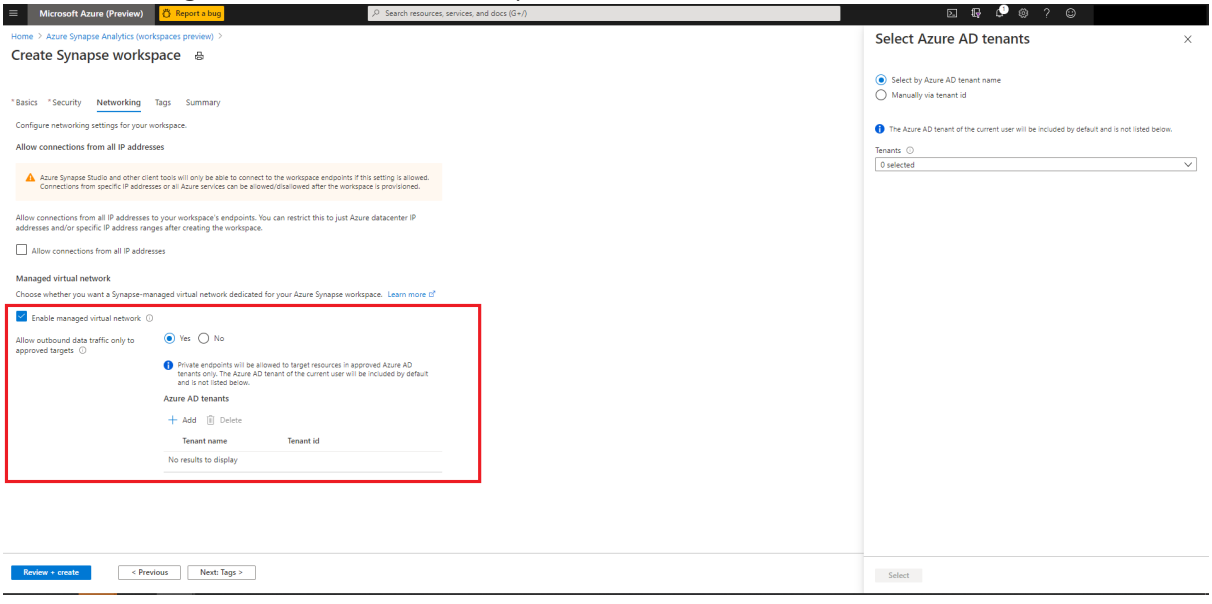
Prerequisites

- Permissions to create a workspace resource in Azure.
- Synapse workspace permissions to create managed private endpoints.
- Subscriptions registered for the Networking resource provider. Learn more.

Follow the steps listed in Quickstart: Create a Synapse workspace to get started with creating your workspace. Before creating your workspace, use the information below to add data exfiltration protection to your workspace.

Add data exfiltration protection when creating your workspace

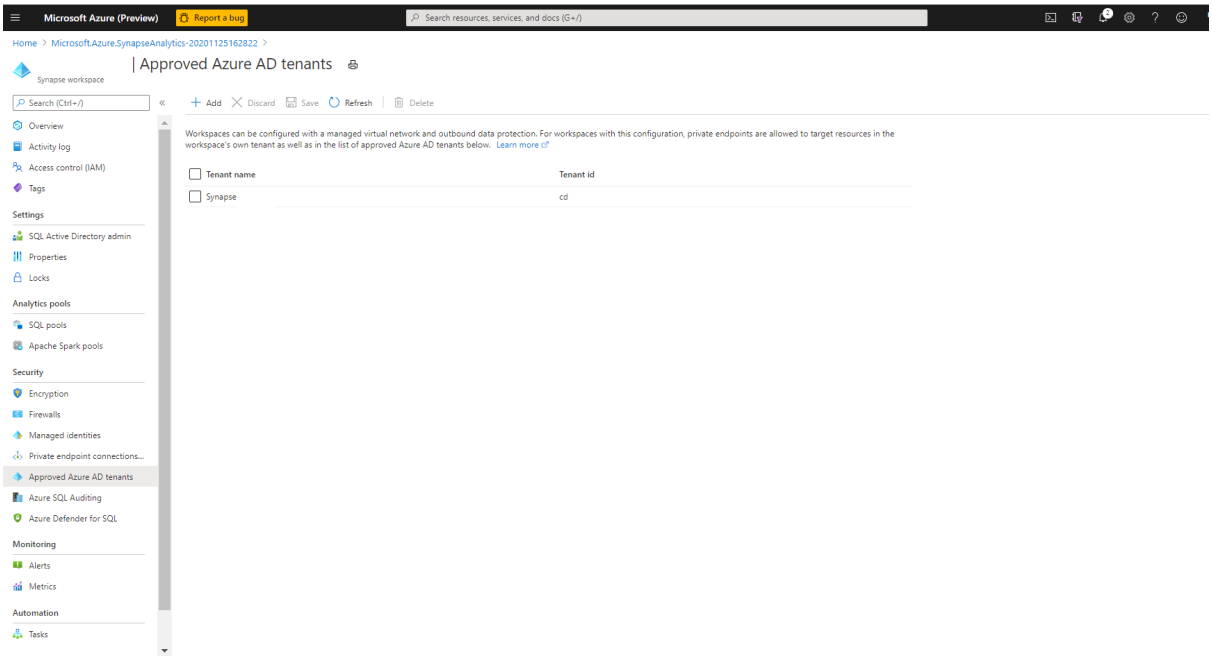
1. On the Networking tab, select the "Enable managed virtual network" checkbox.
2. Select "Yes" for the "Allow outbound data traffic only to approved targets" option.
3. Choose the approved Azure AD tenants for this workspace.
4. Review the configuration and create the workspace.



Manage approved Azure Active Directory tenants for the workspace

1. From the workspace's Azure portal, navigate to "Approved Azure AD tenants". The list of approved Azure AD tenants for the workspace will be listed here. The workspace's tenant is included by default and is not listed.

2. Use "+Add" to include new tenants to the approved list.
3. To remove an Azure AD tenant from the approved list, select the tenant and select on "Delete" and then "Save".



Connecting to Azure resources in approved Azure AD tenants

You can create managed private endpoints to connect to Azure resources that reside in Azure AD tenants, which are approved for a workspace. Follow the steps listed in the guide for creating managed private endpoints.

> **IMPORTANT**
> Resources in tenants other than the workspace's tenant must not have blocking firewall rules in place for the SQL pools to connect to them. Resources within the workspace's managed virtual network, such as Spark clusters, can connect over managed private links to firewall-protected resources.

Known limitations

Users can provide an environment configuration file to install Python packages from public repositories like PyPI. In data exfiltration protected workspaces, connections to outbound repositories are blocked. As a result, Python library installed from public repositories like PyPI are not supported.

As an alternative, users can upload workspace packages or create a private channel within their primary Azure Data Lake Storage account. For more information, visit Package management in Azure Synapse Analytics

Next steps

Learn more about data exfiltration protection in Synapse workspaces

Learn more about Managed workspace Virtual Network

Learn more about Managed private endpoints

Create Managed private endpoints to your data sources

Create a Managed private endpoint to your data source

8/19/2022 • 2 minutes to read • Edit Online

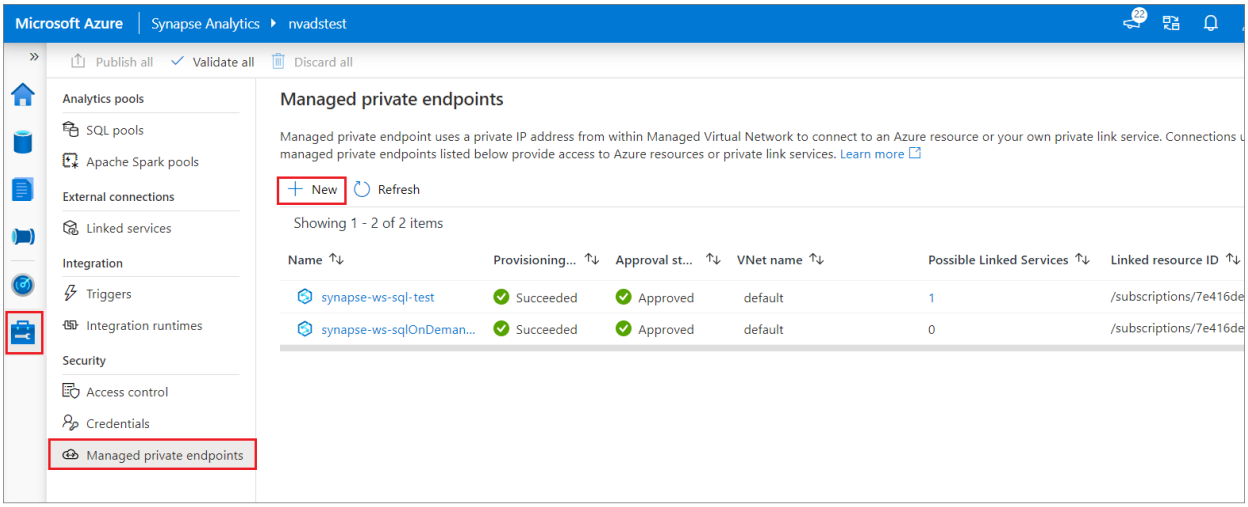
This article will teach you how to create a Managed private endpoint to your data source in Azure. See Managed private endpoints to learn more.

Step 1: Open your Azure Synapse workspace in Azure portal

You can create a Managed private endpoint to your data source from Azure Synapse Studio. Select the **Overview** tab in Azure portal and select **Open** on the Open Synapse Studio card in the Getting started section.

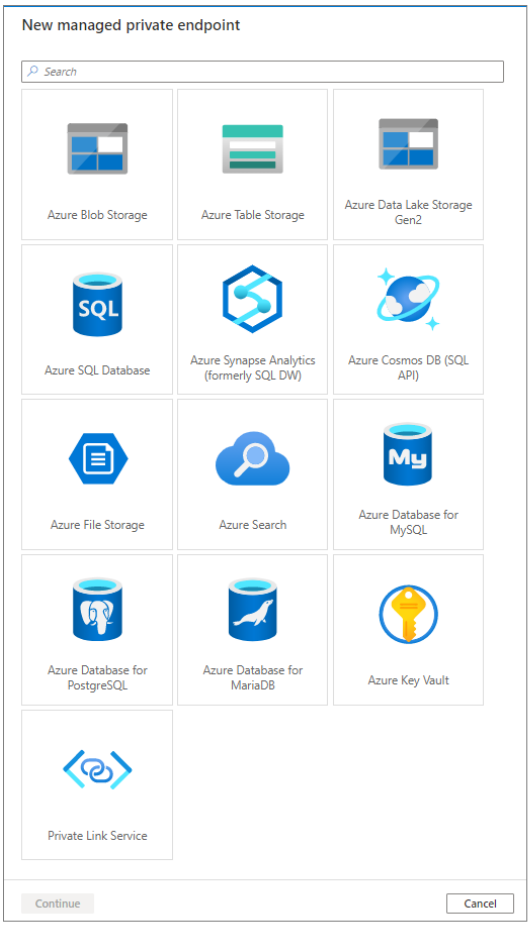
Step 2: Navigate to the Managed Virtual Networks tab in Synapse Studio

In Azure Synapse Studio, select the **Manage** tab from the left navigation. Select **Managed private endpoints** and then select **+ New**.



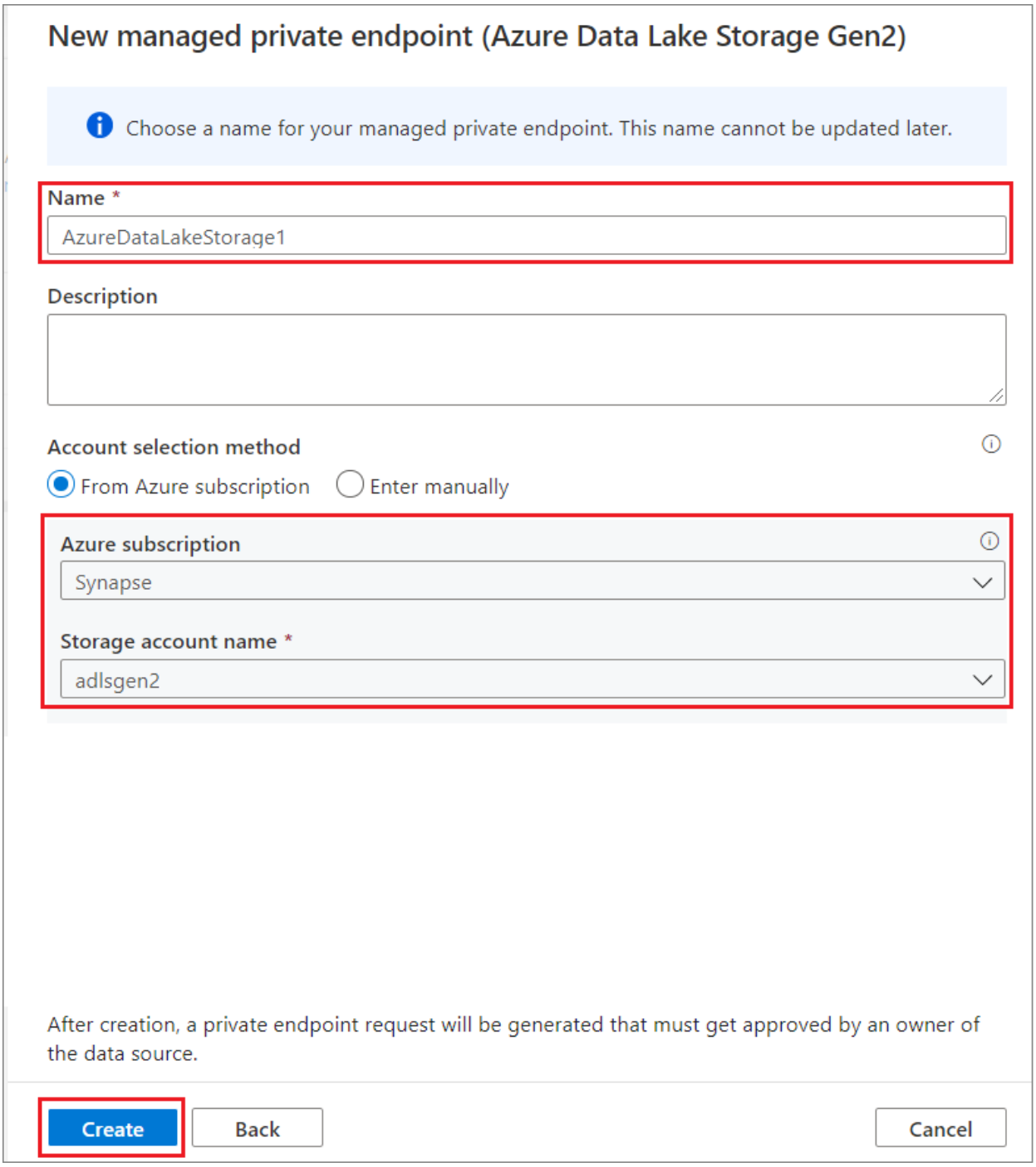
Step 3: Select the data source type

Select the data source type. In this case, the target data source is an ADLS Gen2 account. Select **Continue**.



Step 4: Enter information about the data source

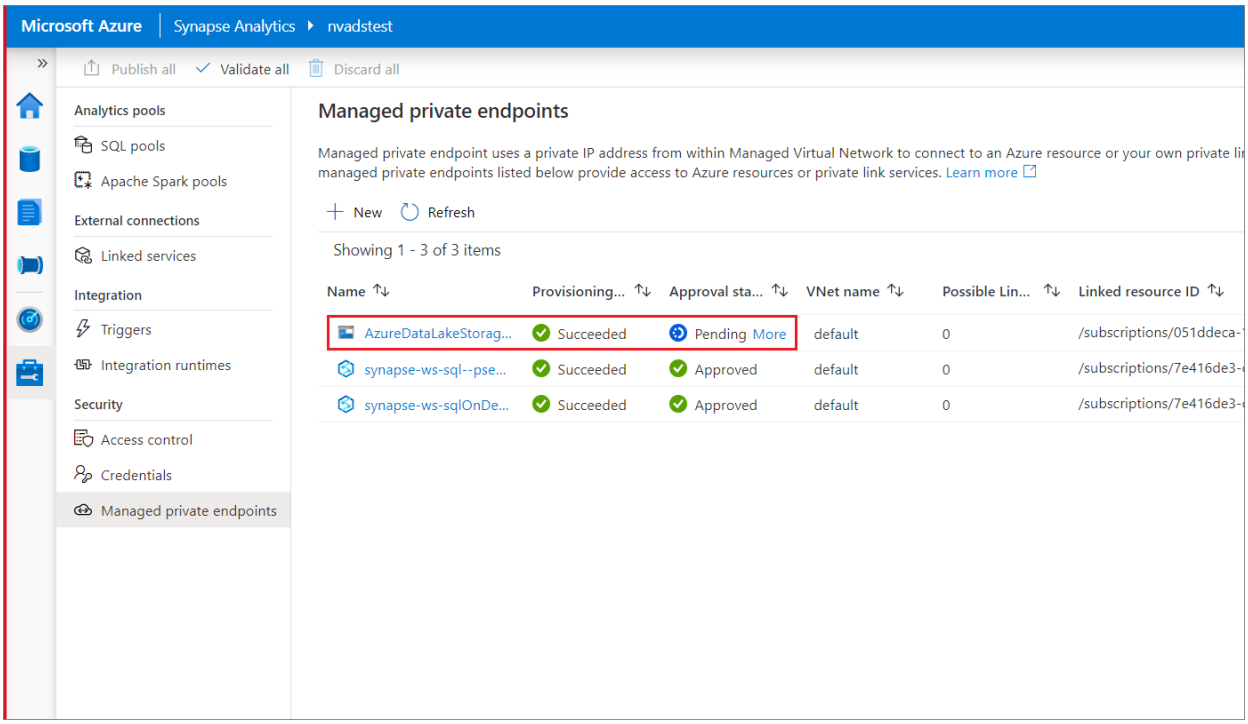
In the next window, enter information about the data source. In this example, we're creating a Managed private endpoint to an ADLS Gen2 account. Enter a **Name** for the Managed private endpoint. Provide an **Azure subscription** and a **Storage account name**. Select **Create**.



Step 5: Verify that your Managed private endpoint was successfully created

After submitting the request, you'll see its status. To verify the successful creation of your Managed private endpoint was created, check its *Provisioning State*. You may need to wait 1 minute and select **Refresh** to update the provisioning state. You can see that the Managed private endpoint to the ADLS Gen2 account was successfully created.

You can also see that the *Approval State* is *Pending*. The owner of the target resource can approve or deny the private endpoint connection request. If the owner approves the private endpoint connection request, then a private link is established. If denied, then a private link isn't established.



Next steps

Learn more about Managed private endpoints

Connect to your Azure Synapse workspace using private links

8/19/2022 • 2 minutes to read • Edit Online

This article will teach you how to create a private endpoint to your Azure Synapse workspace. See private links and private endpoints to learn more.

Step 1: Register Network resource provider

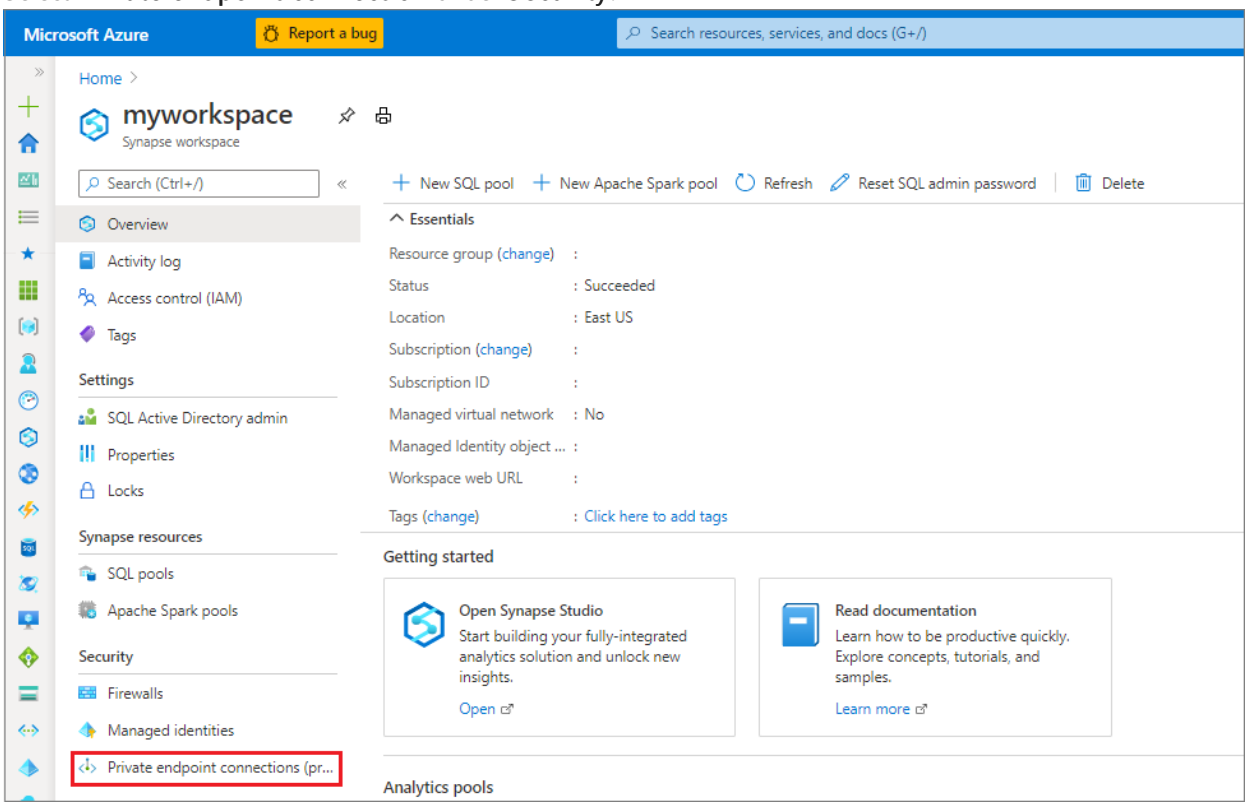
If you haven't already done so, register the Network resource provider in the subscription hosting the Azure Synapse Workspace. Registering a resource provider configures your subscription to work with the resource provider. Choose *Microsoft.Network* from the list of resource providers when you register.

If you are creating a private endpoint in a different subscription than the subscription hosting the Azure Synapse Workspace, register *Microsoft.Synapse* in the subscription hosting the private endpoint. This is required when trying to approve or delete the private endpoint connection.

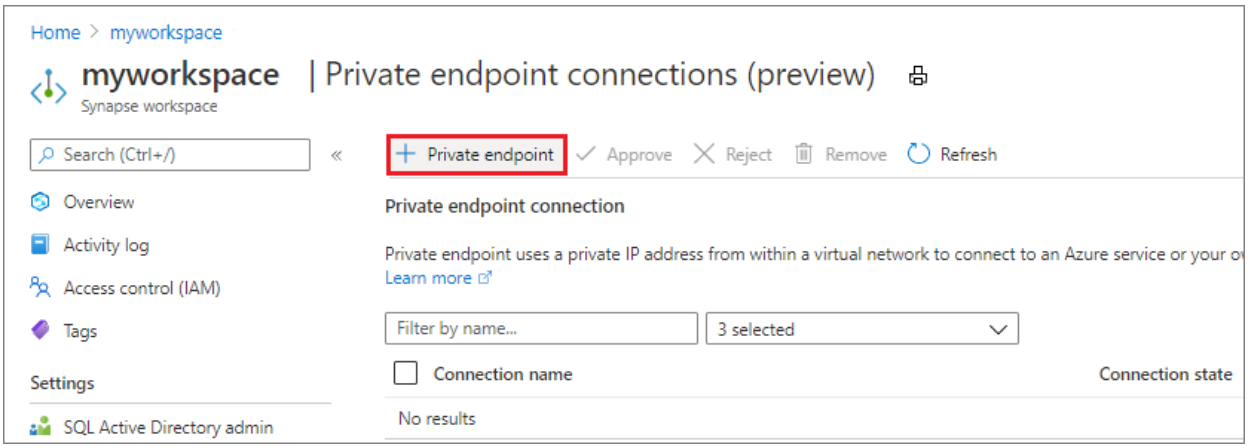
If the required resource provider is already registered, then proceed to Step 2.

Step 2: Open your Azure Synapse workspace in Azure portal

Select **Private endpoint connection** under **Security**.



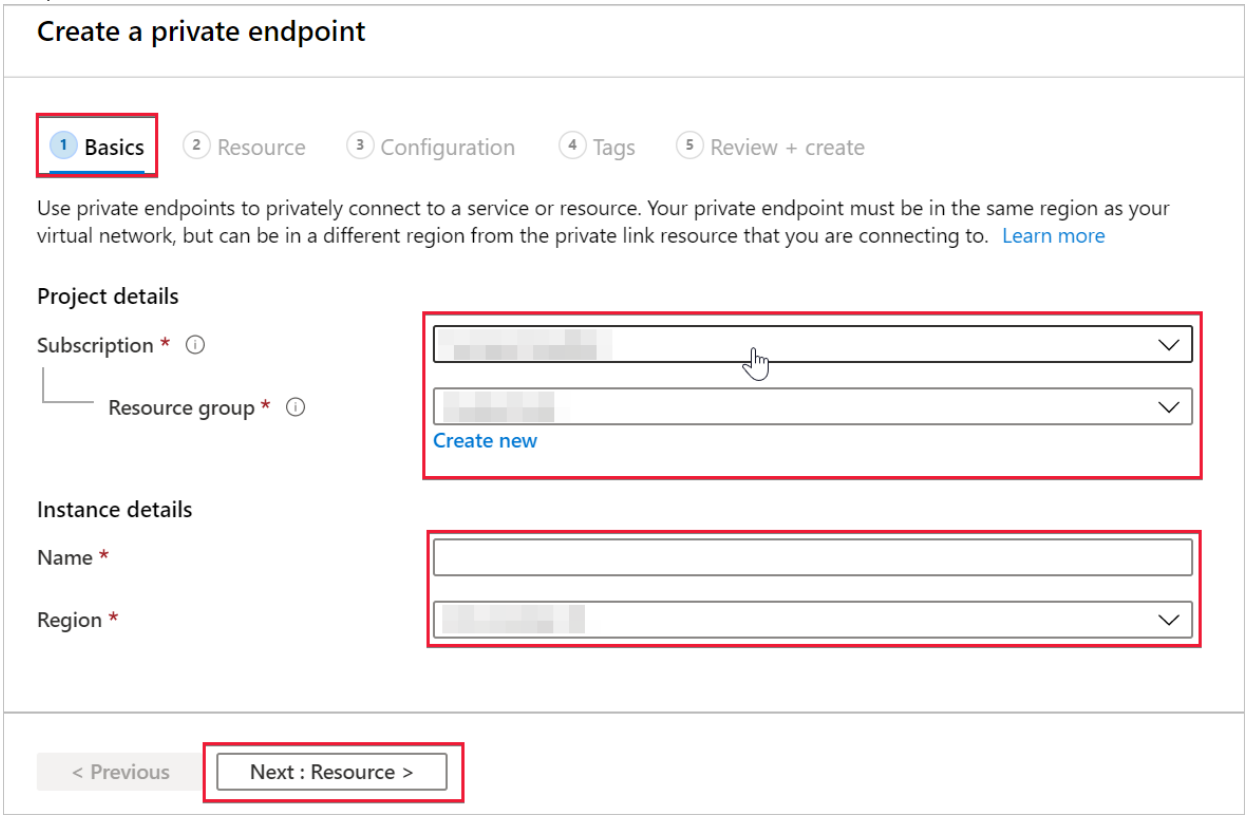
On the next screen select **+ Private endpoint**.



Step 3: Select your subscription and region details

Under the **Basics** tab in the **Create a private endpoint** window, choose your **Subscription** and **Resource Group**. Give a **Name** to the private endpoint that you want to create. Select the **Region** where you want the private endpoint created.

Private endpoints are created in a subnet. The subscription, resource group, and region selected filter the private endpoint subnets. Select **Next: Resource >** when done.



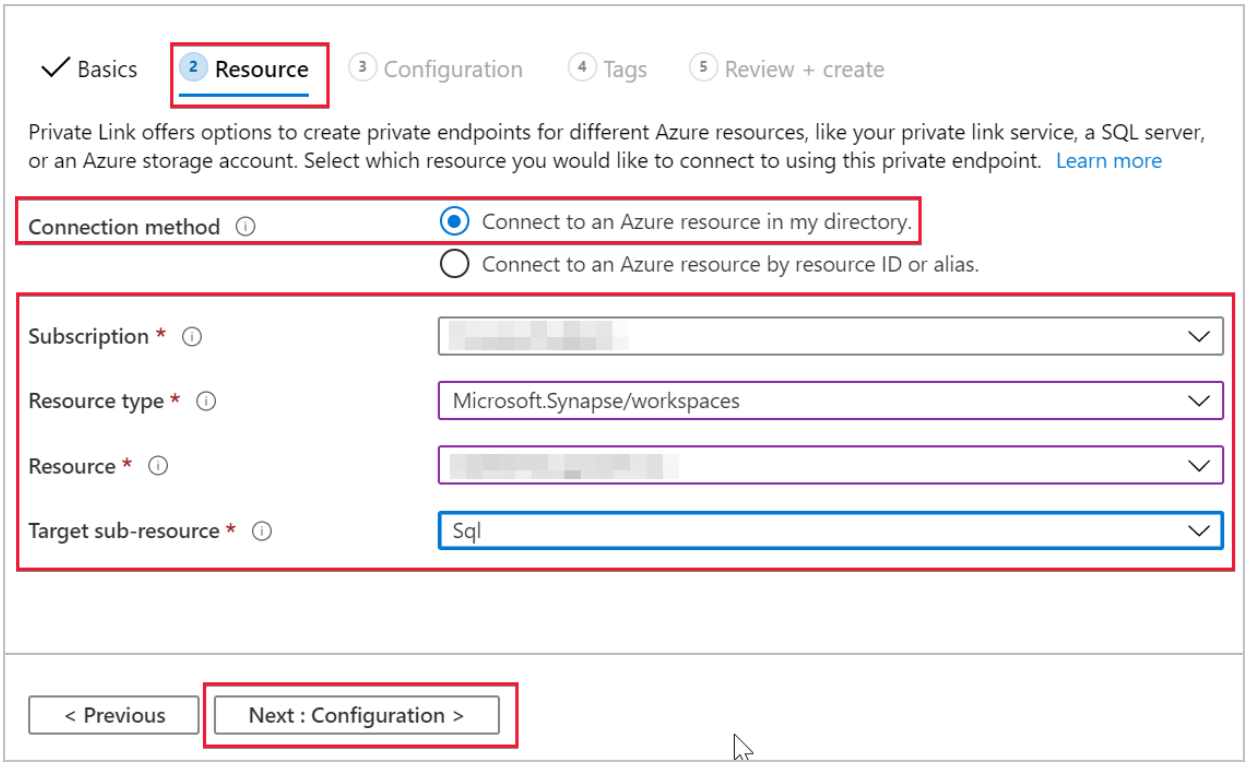
Step 4: Select your Azure Synapse workspace details

Select **Connect to an Azure resource in my directory** in the **Resource** tab. Select the **Subscription** that contains your Azure Synapse workspace. The **Resource type** for creating private endpoints to an Azure Synapse workspace is *Microsoft.Synapse/workspaces*.

Select your Azure Synapse workspace as the **Resource**. Every Azure Synapse workspace has three **Target sub-resource** that you can create a private endpoint to: Sql, SqlOnDemand, and Dev.

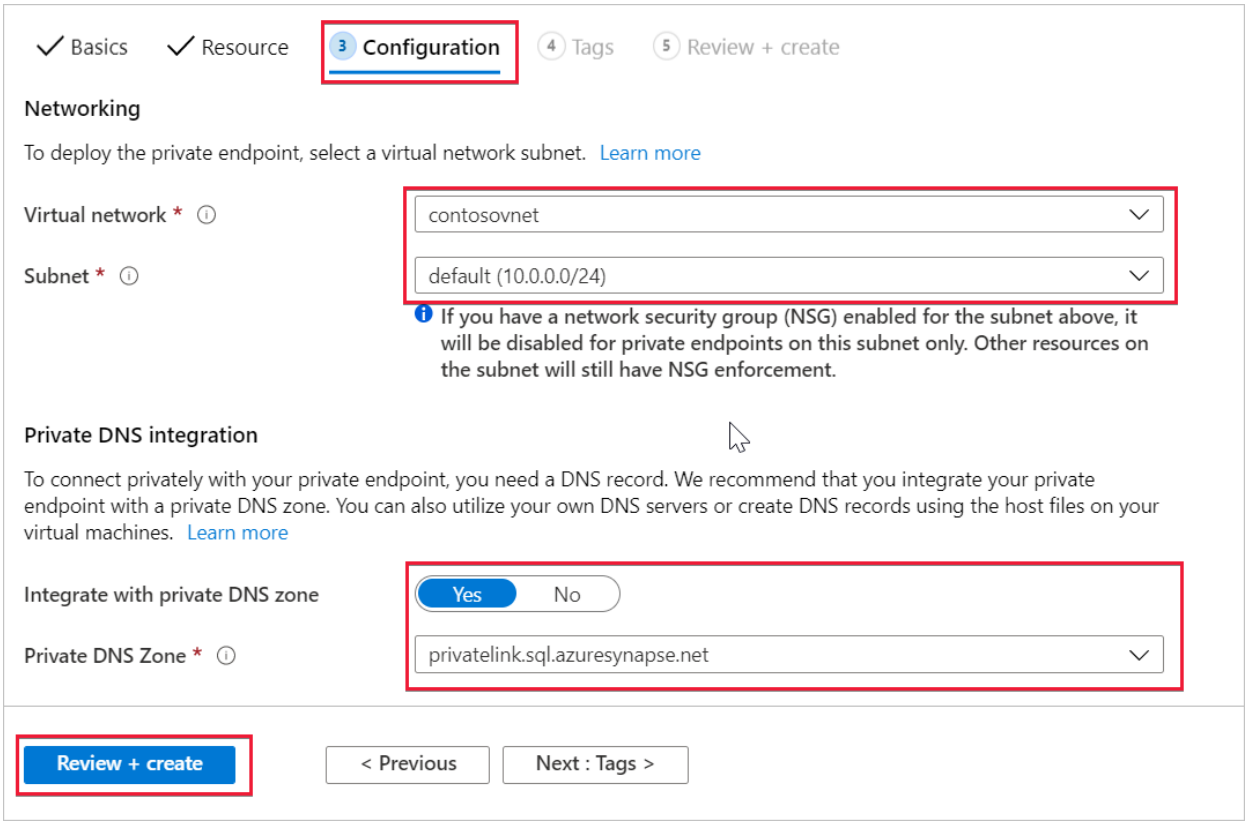
- Sql is for SQL query execution in SQL pool.
- SqlOnDemand is for SQL built-in query execution.
- Dev is for accessing everything else inside Azure Synapse Analytics Studio workspaces.

Select **Next: Configuration>** to advance to the next part of the setup.

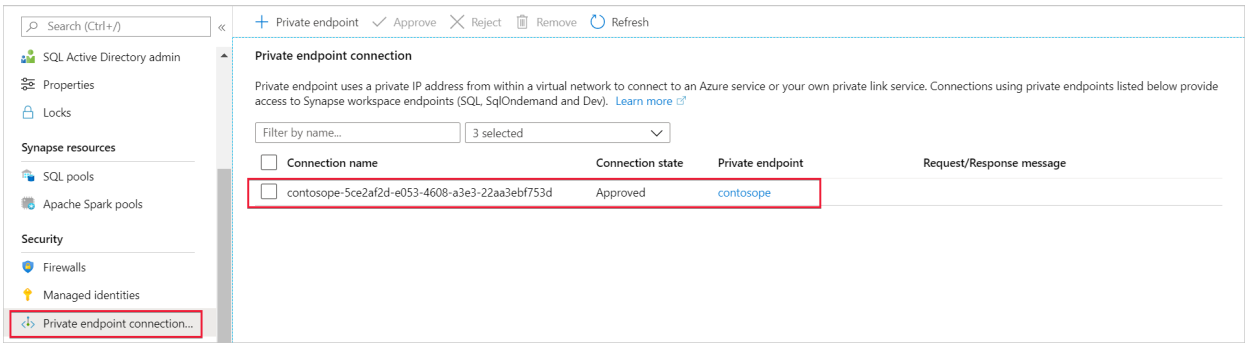


In the **Configuration** tab, select the **Virtual network** and the **Subnet** in which the private endpoint should be created. You also need to create a DNS record that maps to the private endpoint.

Select **Yes** for **Integrate with private DNS zone** to integrate your private endpoint with a private DNS zone. If you don't have a private DNS zone associated with your Microsoft Azure Virtual Network, then a new private DNS zone is created. Select **Review + create** when done.



When the deployment is complete, open your Azure Synapse workspace in Azure portal and select **Private endpoint connections**. The new private endpoint and private endpoint connection name associated to the private endpoint are shown.



Next steps

Learn more about Managed workspace Virtual Network

Learn more about Managed private endpoints

Create Managed private endpoints to your data sources

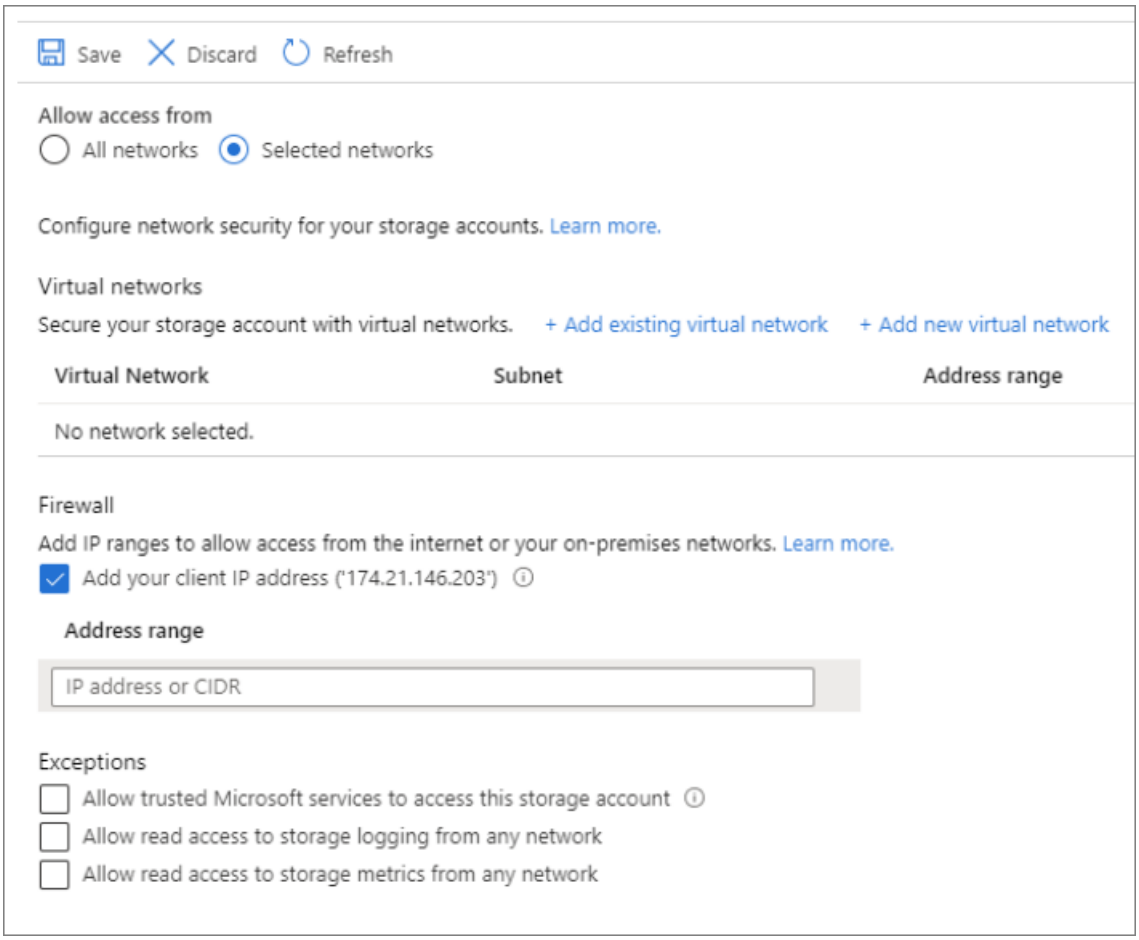
Secure a linked service with Private Links

8/19/2022 • 2 minutes to read • Edit Online

In this article, you'll learn how to secure a linked service in Synapse with a private endpoint.

Prerequisites

- **Azure subscription**: If you don't have an Azure subscription, create a free Azure account before you begin.
- **Azure Storage account**: You use Azure Data Lake Gen 2 as a *source* data store. If you don't have a storage account, see Create an Azure Storage account for steps to create one. Make sure that the Storage Account has the Synapse Studio IP filtering for accessing it and that you only allow **Selected networks** to access the Storage account. The setting under the blade **Firewalls and virtual networks** should look like the picture below.



Create a linked service with Private Links

In Azure Synapse Analytics, a linked service is where you define your connection information to other services. In this section, you'll add Azure Synapse Analytics and Azure Data Lake Gen 2 as linked services.

1. Open the Synapse Studio and go to the **Manage** tab.
2. Under **External connections**, select **Linked services**.
3. To add a linked service, select **New**.
4. Select the Azure Data Lake Storage Gen2 tile from the list and select **Continue**.
5. Make sure you enable **Interactive Authoring**. It might take around 1 minute to be enabled.

6. Enter your authentication credentials. Account key, service principal, and managed identity are currently supported authentication types. Select test connection to verify your credentials are correct.
7. Select **Test connection**, it should fail because the Storage Account doesn't enable access into it without the creation and approval of a Private Endpoint. In the error message, you should see a link to create a **private endpoint** that you can follow to go to the next part. If you follow that link, skip the next part.
8. Select **Create** when finished.

Create a managed private endpoint

If you didn't select into the hyperlink when testing the connection above, follow the following path. Create a managed private endpoint that you'll connect to the linked service created above.

1. Go to the **Manage** tab.
2. Go to the **Managed Virtual Networks** section.
3. Select **+ New** under Managed private endpoint.
4. Select the Azure Data Lake Storage Gen2 tile from the list and select **Continue**.
5. Enter the name of the Storage Account you created above.
6. Select **Create**
7. You should see after waiting some seconds that the private link created needs an approval.

Private link approval

1. Select the Private Endpoint that you created above. You can see a hyperlink that will let you approve the Private Endpoint at the Storage Account level. *An alternative is to go directly to the Azure portal Storage Account and go into the **Private endpoint connections** blade.*
2. Tick the Private endpoint you created in the Studio and select **Approve**.
3. Add a description and select **yes**
4. Go back to Synapse Studio in under the **Managed Virtual Networks** section of the **Manage** tab.
5. It should take around 1 minute to get the approval reflected for your private endpoint.

Check the connection works

1. Go to the **Manage** tab and select the linked service you created.
2. Make sure that **Interactive authoring** is active.
3. Select **Test connection**. You should see the connection be successful.

You've now established a secure and private connection between Synapse and your linked service.

Next steps

To develop further understanding of Managed private endpoint in Azure Synapse Analytics, see Managed private endpoints.

For more information on data integration for Azure Synapse Analytics, see the Ingesting data into a Data Lake article.

Connect to workspace resources from a restricted network

8/19/2022 • 5 minutes to read • Edit Online

Suppose you are an IT administrator who is managing your organization's restricted network. You want to enable the network connection between Azure Synapse Analytics Studio and a workstation within this restricted network. This article shows you how.

Prerequisites

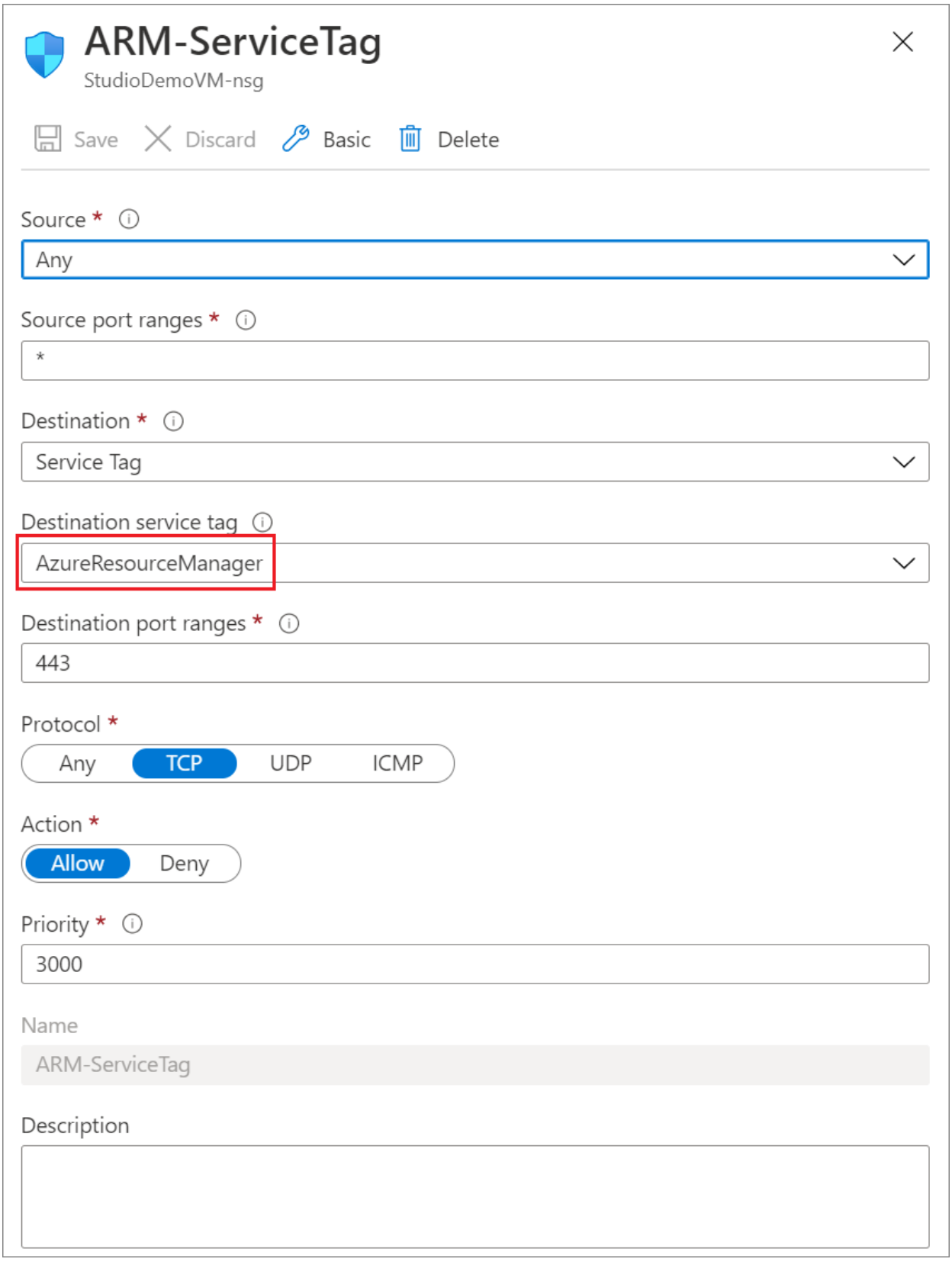
- **Azure subscription**: If you don't have an Azure subscription, create a free Azure account before you begin.
- **Azure Synapse Analytics workspace**: You can create one from Azure Synapse Analytics. You need the workspace name in step 4.
- **A restricted network**: The IT admin maintains the restricted network for the organization, and has permission to configure the network policy. You need the virtual network name and its subnet in step 3.

Step 1: Add network outbound security rules to the restricted network

You'll need to add four network outbound security rules with four service tags.

- AzureResourceManager
- AzureFrontDoor.Frontend
- AzureActiveDirectory
- AzureMonitor (This type of rule is optional. Add it only when you want to share the data with Microsoft.)

The following screenshot shows details for the Azure Resource Manager outbound rule.

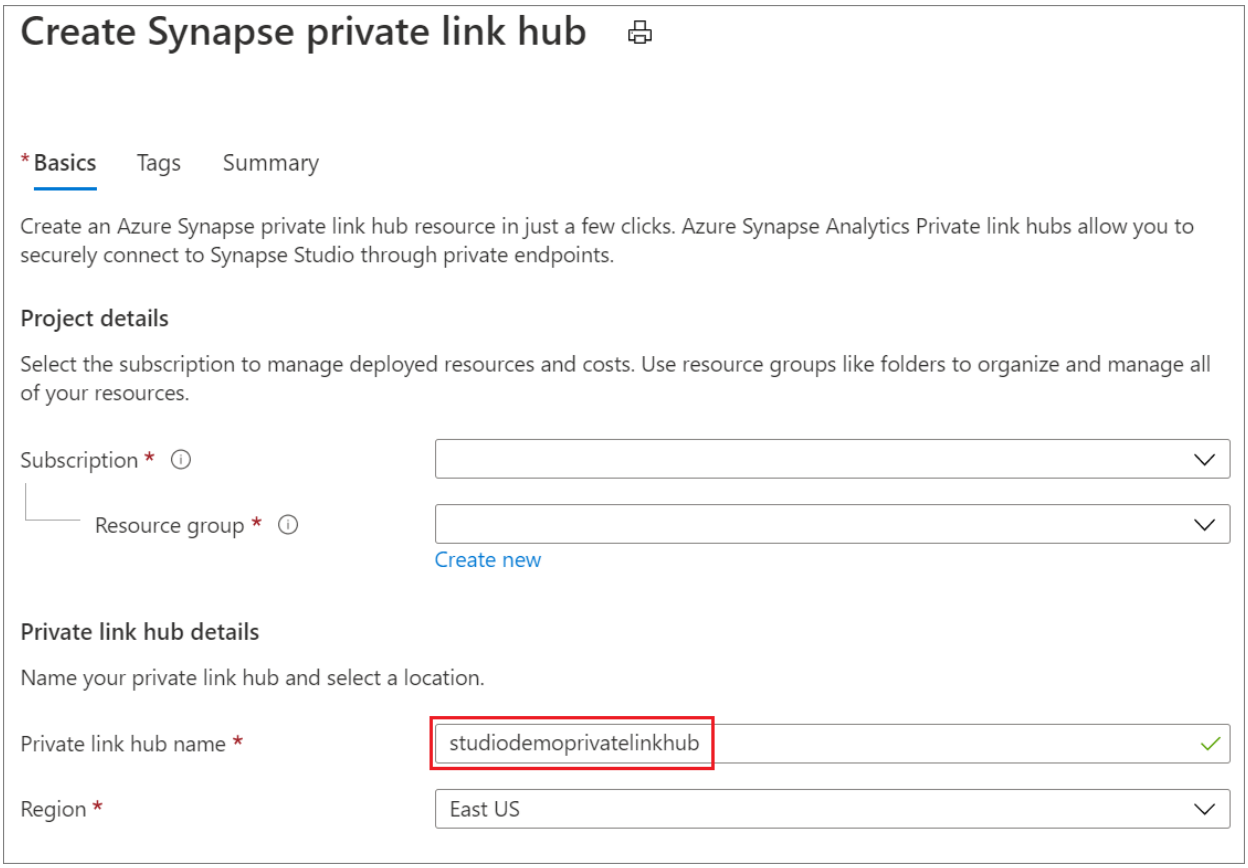


When you're creating the other three rules, replace the value of **Destination service tag** with **AzureFrontDoor.Frontend**, **AzureActiveDirectory**, or **AzureMonitor** from the list.

For more information, see Service tags overview.

Step 2: Create private link hubs

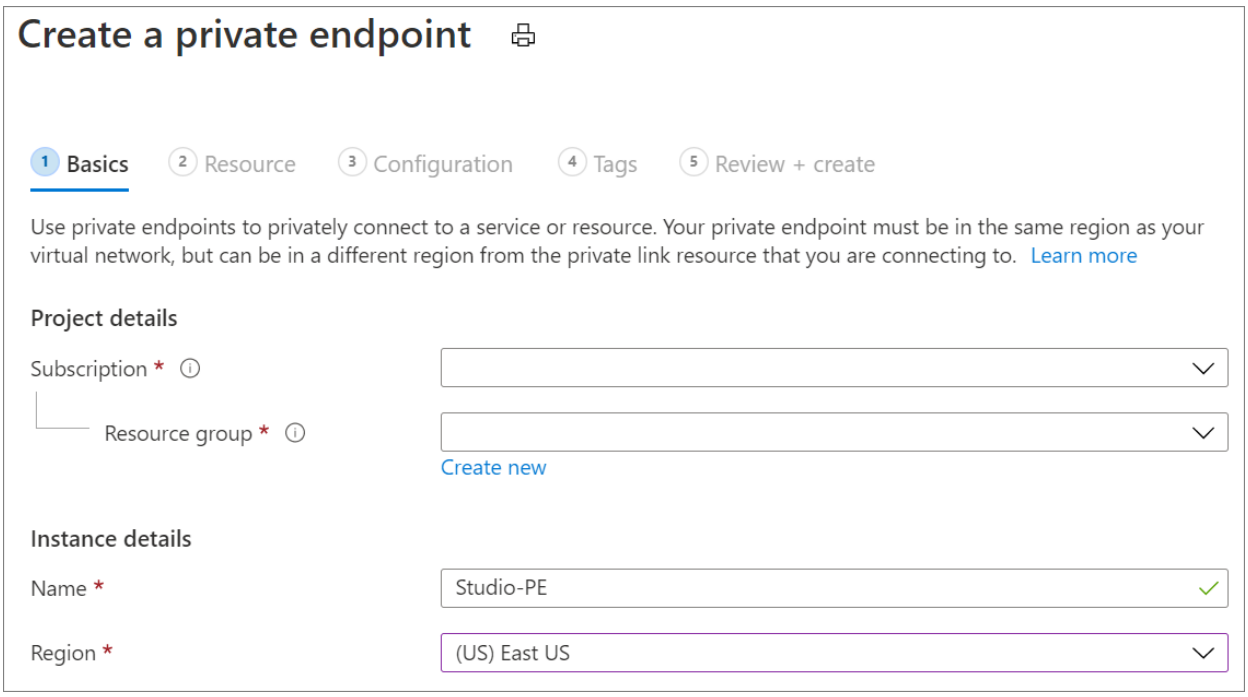
Next, create private link hubs from the Azure portal. To find this in the portal, search for *Azure Synapse Analytics (private link hubs)*, and then fill in the required information to create it.



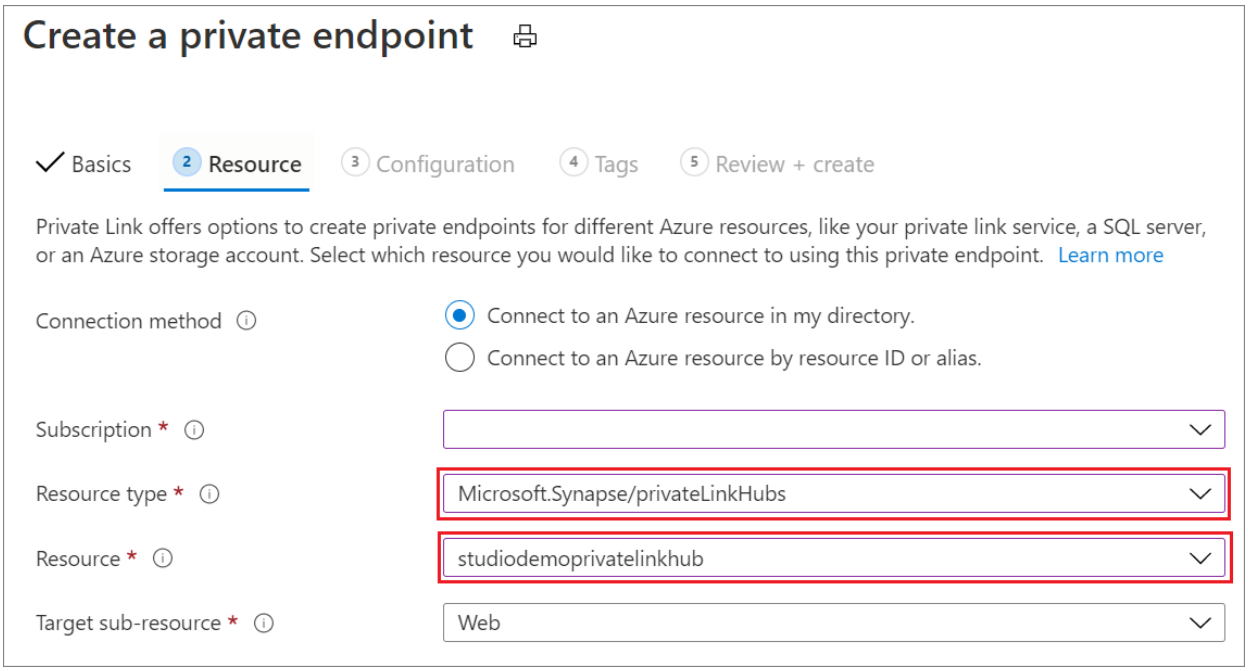
Step 3: Create a private endpoint for your Synapse Studio

To access the Azure Synapse Analytics Studio, you must create a private endpoint from the Azure portal. To find this in the portal, search for *Private Link*. In the **Private Link Center**, select **Create private endpoint**, and then fill in the required information to create it.

> **NOTE**
> Ensure that the **Region** value is the same as the one where your Azure Synapse Analytics workspace is.

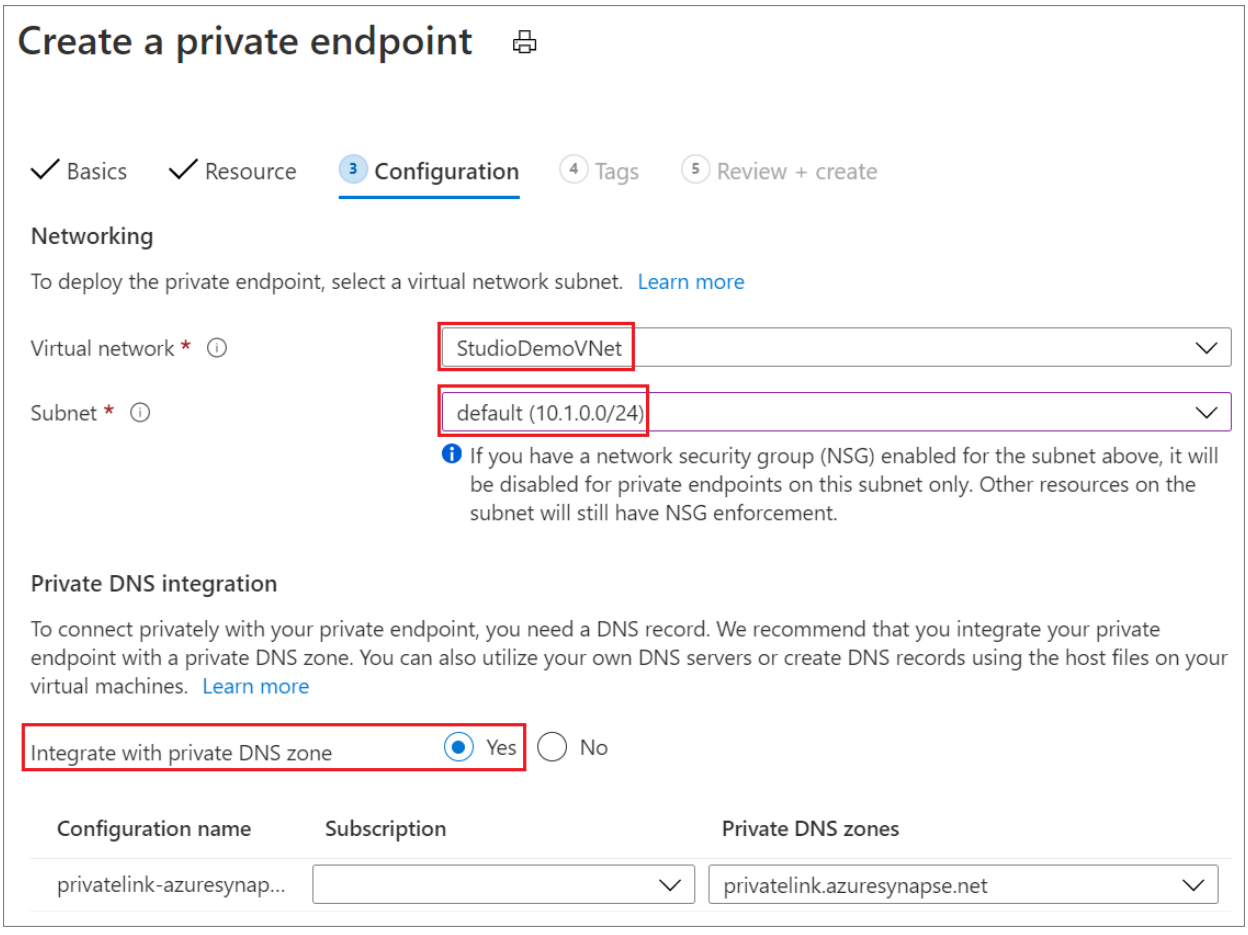


On the **Resource** tab, choose the private link hub, which you created in step 2.



On the **Configuration** tab:

- For **Virtual network**, select the restricted virtual network name.
- For **Subnet**, select the subnet of the restricted virtual network.
- For **Integrate with private DNS zone**, select **Yes**.



After the private link endpoint is created, you can access the sign-in page of the web tool for Azure Synapse Analytics Studio. However, you aren't able to access the resources inside your workspace yet. For that, you need to complete the next step.

Step 4: Create private endpoints for your workspace resource

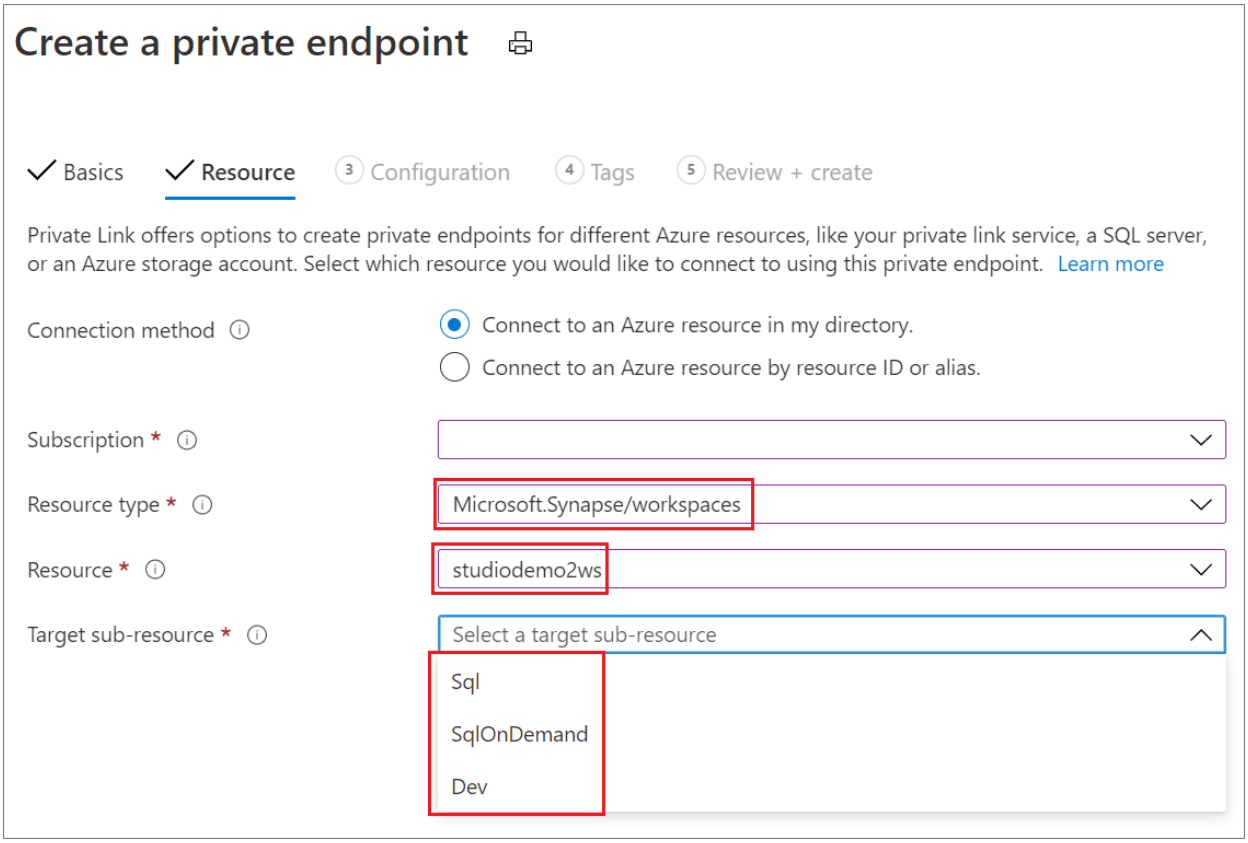
To access the resources inside your Azure Synapse Analytics Studio workspace resource, you need to create the following:

- At least one private link endpoint with a **Target sub-resource** type of **Dev**.
- Two other optional private link endpoints with types of **Sql** or **SqlOnDemand**, depending on what resources in the workspace you want to access.

Creating these is similar to how you create the endpoint in the previous step.

On the **Resource** tab:

- For **Resource type**, select **Microsoft.Synapse/workspaces**.
- For **Resource**, select the workspace name that you created previously.
- For **Target sub-resource**, select the endpoint type:
  - **Sql** is for SQL query execution in SQL pool.
  - **SqlOnDemand** is for SQL built-in query execution.
  - **Dev** is for accessing everything else inside Azure Synapse Analytics Studio workspaces. You need to create at least one private link endpoint of this type.

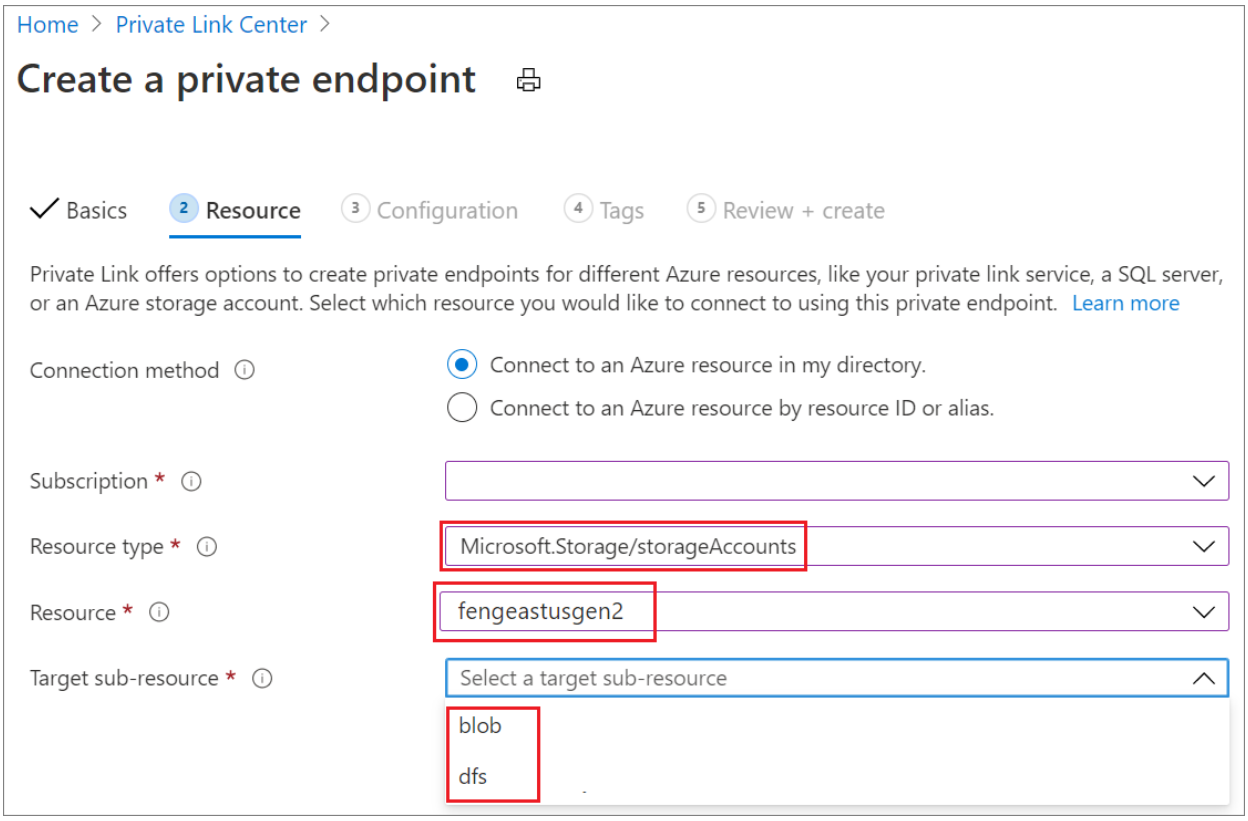


Step 5: Create private endpoints for workspace linked storage

To access the linked storage with the storage explorer in Azure Synapse Analytics Studio workspace, you must create one private endpoint. The steps for this are similar to those of step 3.

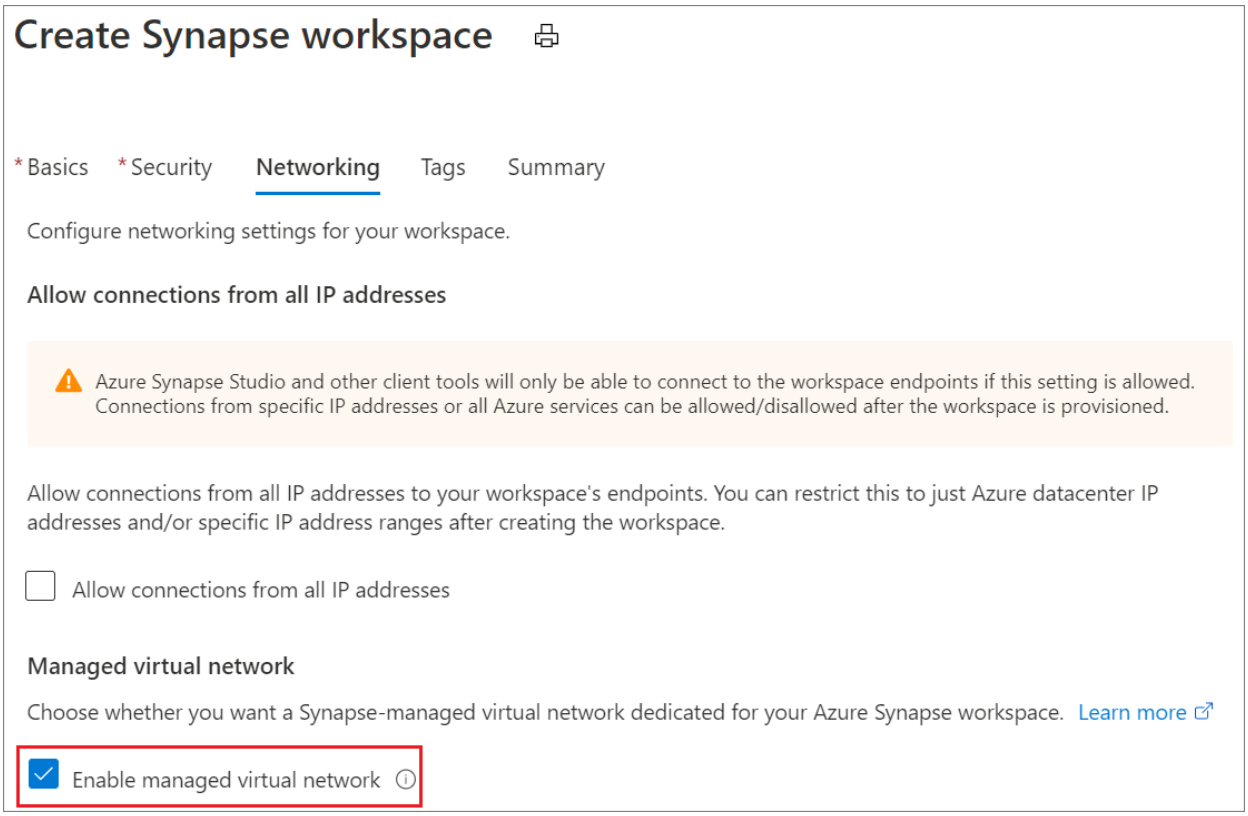
On the **Resource** tab:

- For **Resource type**, select **Microsoft.Storage/storageAccounts**.
- For **Resource**, select the storage account name that you created previously.
- For **Target sub-resource**, select the endpoint type:
  - **blob** is for Azure Blob Storage.
  - **dfs** is for Azure Data Lake Storage Gen2.



Now, you can access the linked storage resource. Within your virtual network, in your Azure Synapse Analytics Studio workspace, you can use the storage explorer to access the linked storage resource.

You can enable a managed virtual network for your workspace, as shown in this screenshot:



If you want your notebook to access the linked storage resources under a certain storage account, add managed private endpoints under your Azure Synapse Analytics Studio. The storage account name should be the one your notebook needs to access. For more information, see Create a managed private endpoint to your data source.

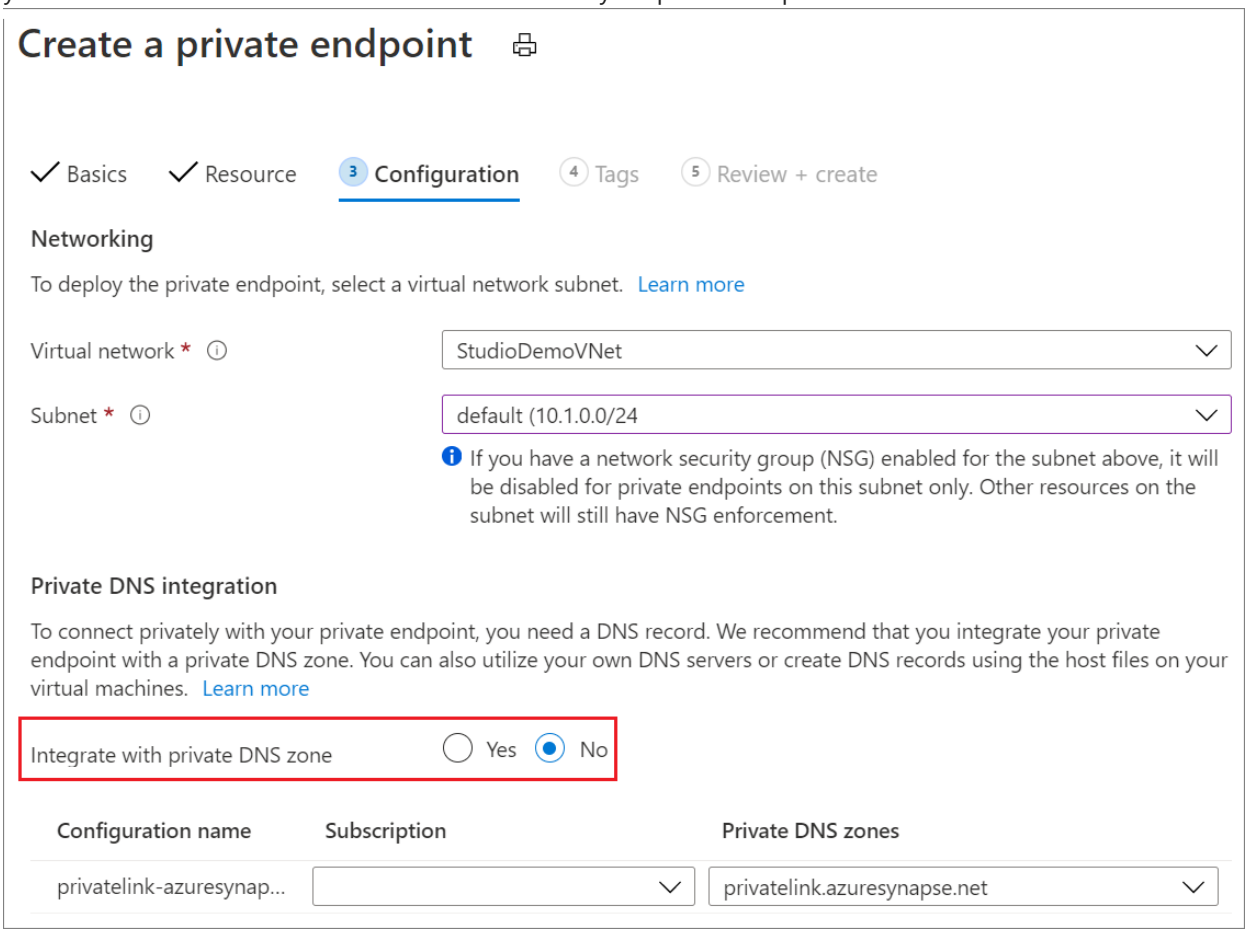
After you create this endpoint, the approval state shows a status of **Pending**. Request approval from the owner of this storage account, in the **Private endpoint connections** tab of this storage account in the Azure portal.

After it's approved, your notebook can access the linked storage resources under this storage account.

Now, all set. You can access your Azure Synapse Analytics Studio workspace resource.

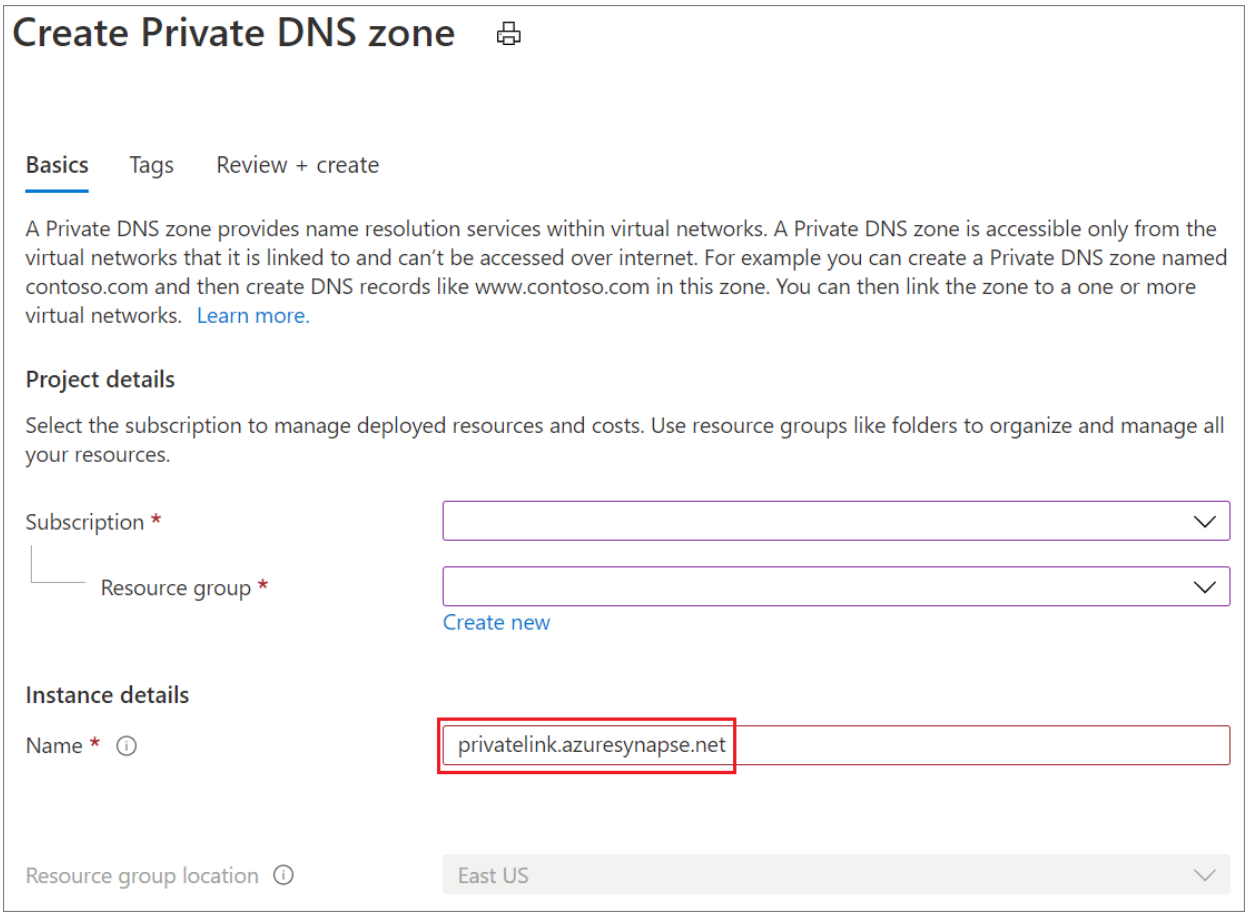
Appendix: DNS registration for private endpoint

If the "Integrate with private DNS zone" is not enabled during the private endpoint creation as screenshot below, you must create the "**Private DNS zone**" for each of your private endpoints.

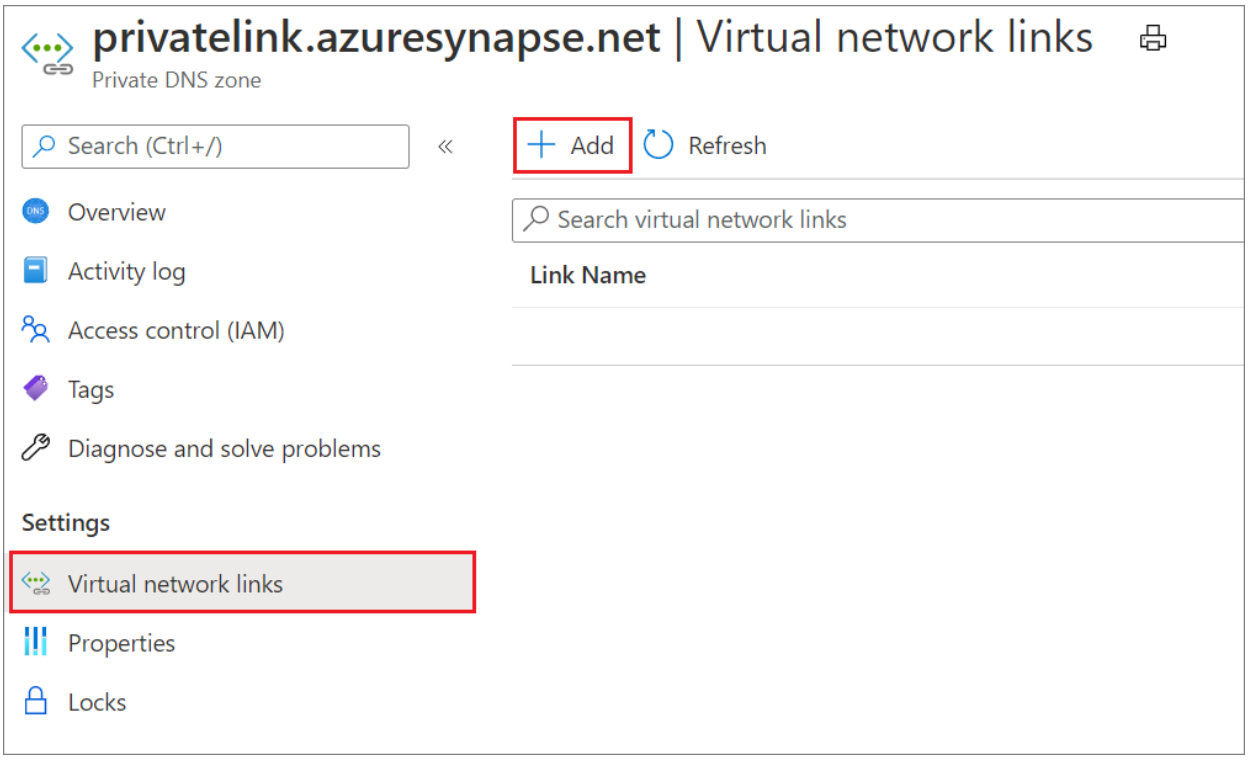


To find the **Private DNS zone** in the portal, search for *Private DNS zone*. In the **Private DNS zone**, fill in the required information below to create it.

- For **Name**, input the private DNS zone dedicated name for specific private endpoint as below:
  - `privatelink.azuresynapse.net` is for the private endpoint of accessing Azure Synapse Analytics Studio gateway. See this type of private endpoint creation in step 3.
  - `privatelink.sql.azuresynapse.net` is for this type of private endpoint of sql query execution in SQL pool and built-in pool. See the endpoint creation in step 4.
  - `privatelink.dev.azuresynapse.net` is for this type of private endpoint of accessing everything else inside Azure Synapse Analytics Studio workspaces. See this type of private endpoint creation in step 4.
  - `privatelink.dfs.core.windows.net` is for the private endpoint of accessing workspace linked Azure Data Lake Storage Gen2. See this type of private endpoint creation in step 5.
  - `privatelink.blob.core.windows.net` is for the private endpoint of accessing workspace linked Azure Blob Storage. See this type of private endpoint creation in step 5.

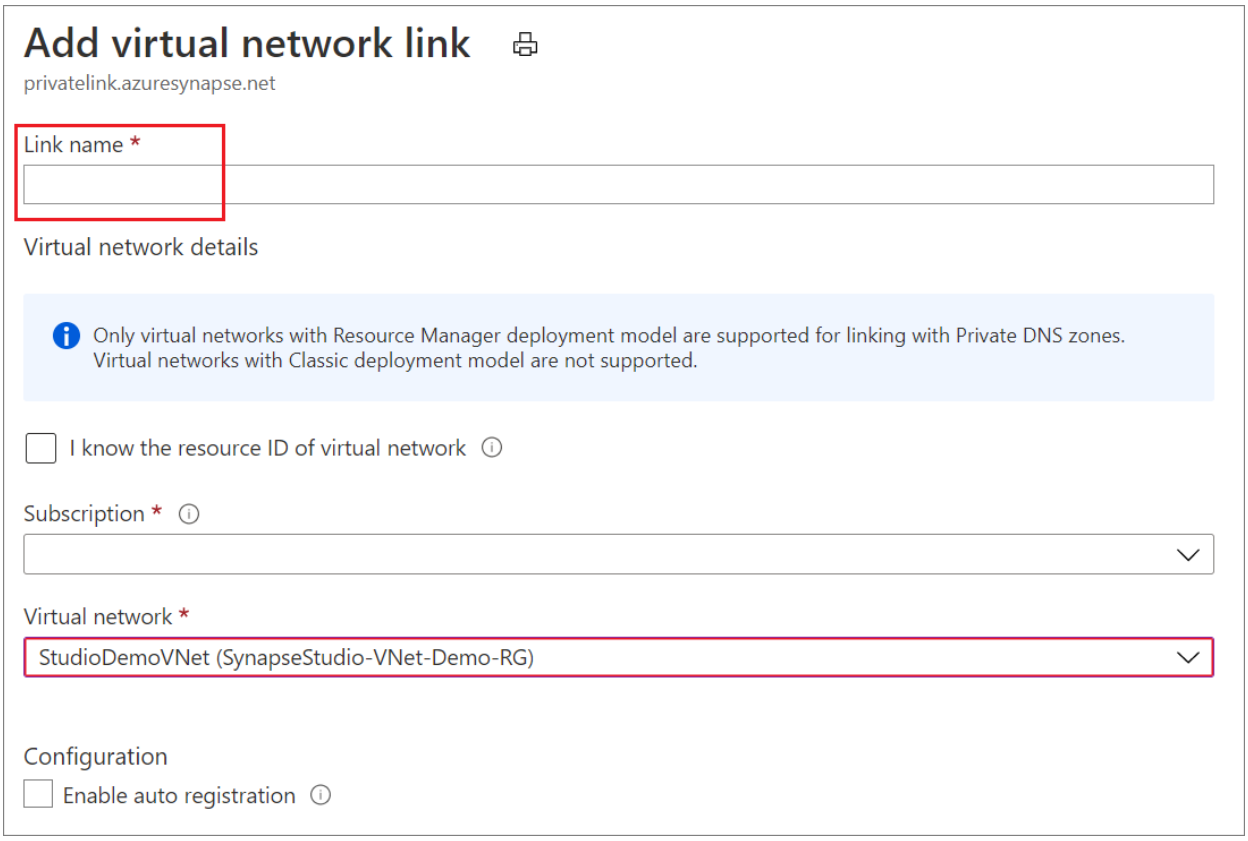


After the **Private DNS zone** created, enter the created private DNS zone and select the **Virtual network links** to add the link to your virtual network.



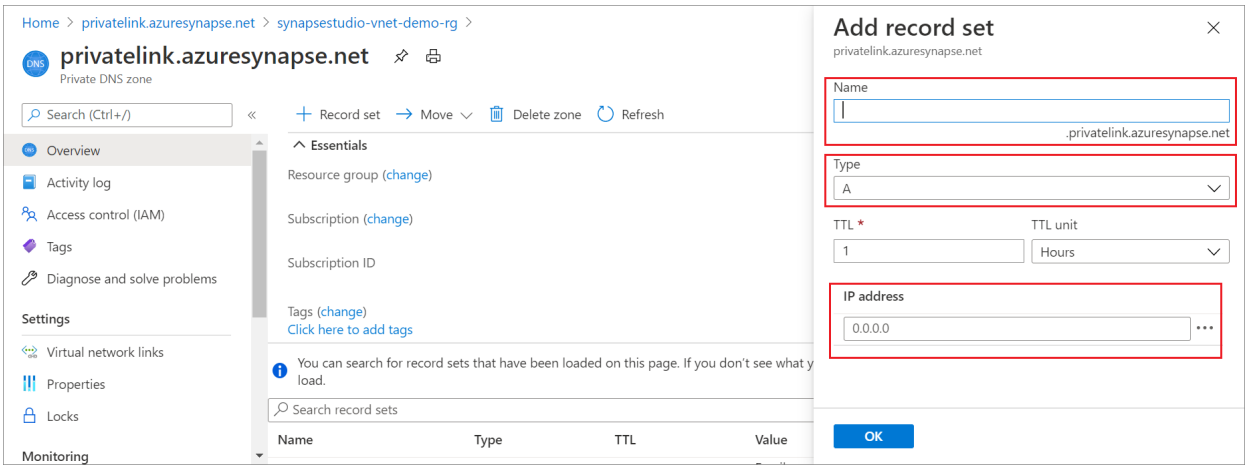
Fill in the mandatory fields as below:

- For **Link name**, input the link name.
- For **Virtual network**, select your virtual network.



After the virtual network link is added, you need to add the DNS record set in the **Private DNS zone** you created before.

- For **Name**, input the dedicated name strings for different private endpoint:
  - **web** is for the private endpoint of accessing Azure Synapse Analytics Studio.
  - "***YourWorkSpaceName***" is for the private endpoint of sql query execution in SQL pool and also for the private endpoint of accessing everything else inside Azure Synapse Analytics Studio workspaces.
  - "***YourWorkSpaceName*-ondemand**" is for the private endpoint of sql query execution in built-in pool.
- For **Type**, select DNS record type **A** only.
- For **IP address**, input the corresponding IP address of each private endpoint. You can get the IP address in **Network interface** from your private endpoint overview.



Next steps

Learn more about Managed workspace virtual network.

Learn more about Managed private endpoints.

Synapse Studio troubleshooting

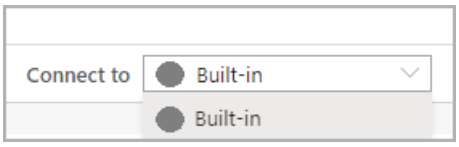
8/19/2022 • 5 minutes to read • Edit Online

This trouble-shooting guide provides instruction on what information to provide when opening a support ticket on network connectivity issues. With the proper information, we can possibly resolve the issue more quickly.

Serverless SQL pool service connectivity issue

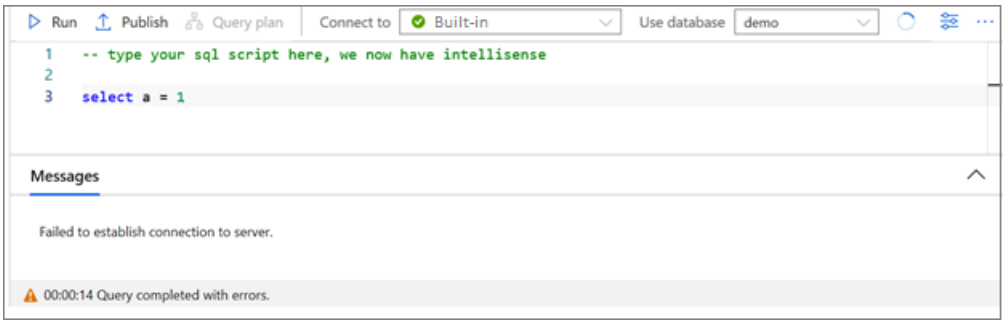
Symptom 1

"Serverless SQL pool" option is grayed out in the "Connect to" dropdown.



Symptom 2

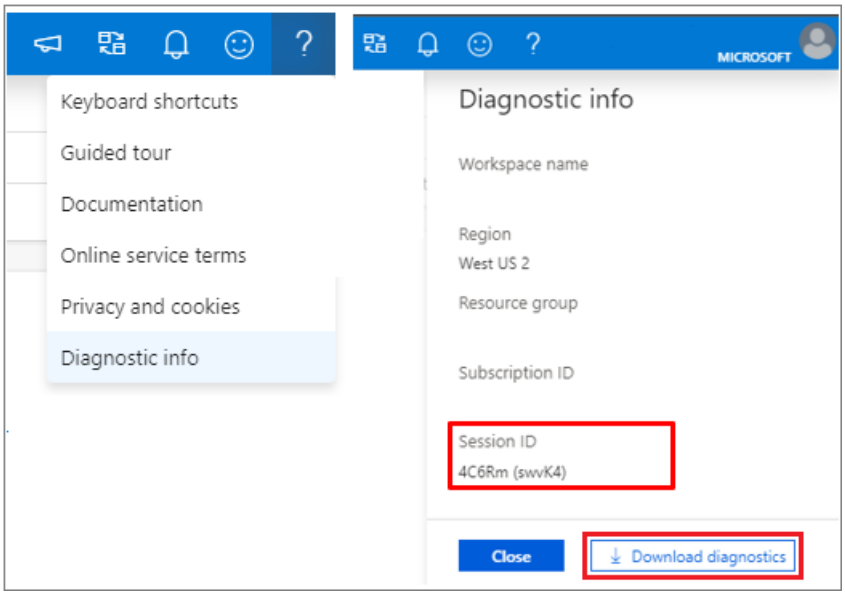
Running the query with "serverless SQL pool" gives you the "Failed to establish connection to server" error message.



Troubleshooting steps

> **NOTE**
> The following troubleshooting steps are for Chromium Edge and Chrome. You may use other browsers (such as FireFox) with the same troubleshooting steps, but the "Developer Tool" window may have different layout from the screenshots in this TSG. If possible, DO NOT use classical Edge for troubleshooting, as it may show inaccurate information in certain situation.

Open the "Diagnostic Info" panel, select the "Download Diagnostic" button. Keep the downloaded information for error reporting. You can instead, copy the "Session ID" and attach it when opening the support ticket.



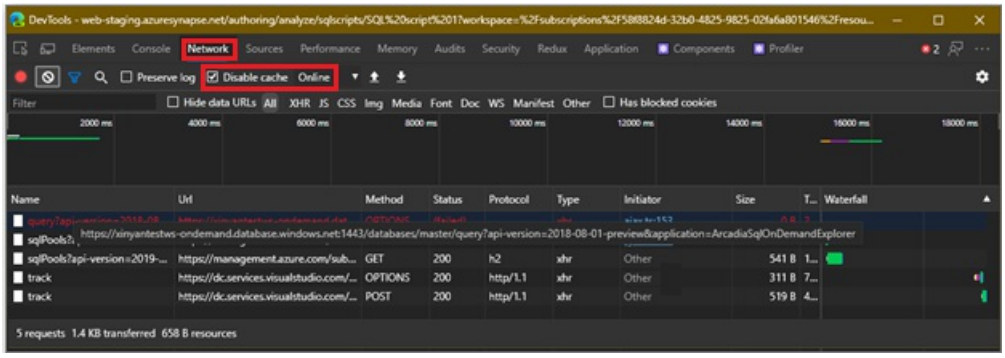
To begin troubleshooting, retry the operation you performed in Synapse Studio.

- For symptom 1, select the "Refresh" button to the right of the "Use database" dropdown in the "SQL script" tab and check whether you can see "serverless SQL pool".
- For symptom 2, try running the query again to see if it executes successfully.

If the problem still exists, press F12 in your browser to open "Developer Tools" (DevTools).

In the "Developer Tools" window, switch to the "Network" panel. Select the "Clear" button on the toolbar in "Network" panel if necessary. Make sure "Disable cache" in "Network" panel is checked.

Retry the operation you performed in Azure Synapse Studio. You may see new items shown in the "Network" list in "Developer Tools". Note your current system time to provide in the support ticket.



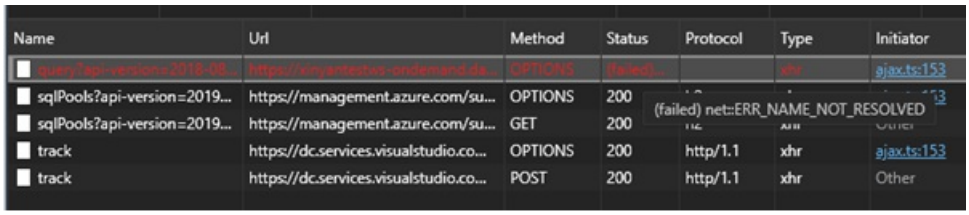
Find the item whose Url column matches the following pattern:

```
https://[*A*]-ondemand.database.windows.net:1443/databases/[*B*]/query?api-version=2018-08-01-preview&application=ArcadiaSqlOnDemandExplorer
```

Where [*A*] is your workspace name, and "-ondemand" could be "-sqlod" and where [*B*] should be a database name, such as "master". There should be at most two items with the same URL value but different method values; OPTIONS and POST. Check whether these two items have "200" or "20x" under the status column, where "x" could be any single digit.

If either one of them has something other than "20x" and:

- Status starts with "(failed)", either widen the "Status" column or hover your pointer over the status text to see the complete text. Include the text and/or screenshot when opening the support ticket.

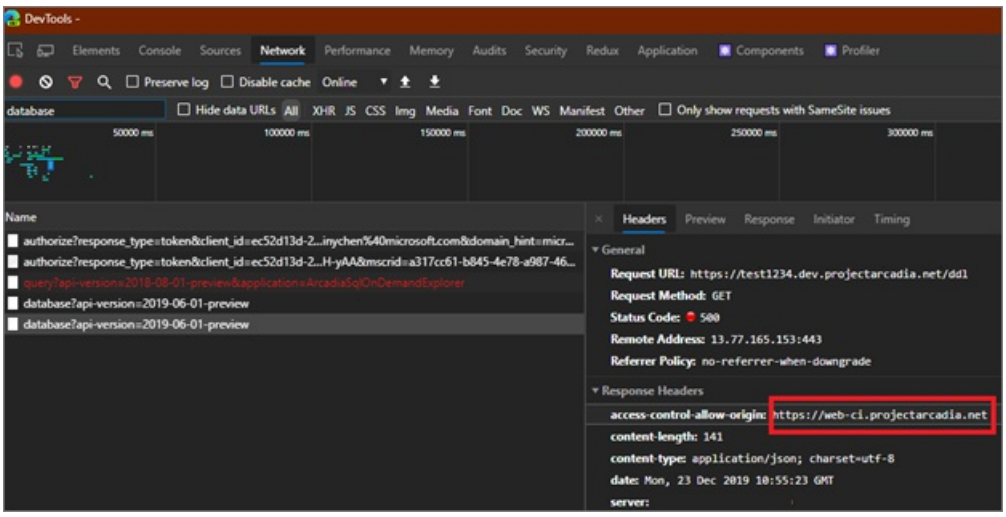


- If you see ERR_NAME_NOT_RESOLVED and you created your workspace within 10 minutes, wait for 10 minutes and retry to see whether the problem still exists.
- If you see ERR_INTERNET_DISCONNECTED or ERR_NETWORK_CHANGED, it may indicate your PC network connection is having issues. Check your network connection and retry the operation.
- If you see ERR_CONNECTION_RESET, ERR_SSL_PROTOCOL_ERROR or other error codes containing "SSL", it may indicate your local SSL configuration is having issues, or your network administrator has blocked access to the serverless SQL pool server. Open a support ticket and attach the error code in the description.
- If you see ERR_NETWORK_ACCESS_DENIED, you may need to check with the administrator on whether your local firewall policy has blocked access to either *.database.windows.net domain, or remote port 1443.
- Optionally, try the same operation immediately on a different machine and/or network environment to rule out a network configuration issue on your PC.

- Status is "40x", "50x", or other numbers, select on the item(s) to see the details. You should see the item details to the right. Find the "Response Header" section; then check whether an item named "access-control-allow-origin" exists. If so, check whether it has one of the following values:
  - `*` (single asterisk)
  - https://web.azuresynapse.net/ (or other value that the text in your browser address bar starts with)

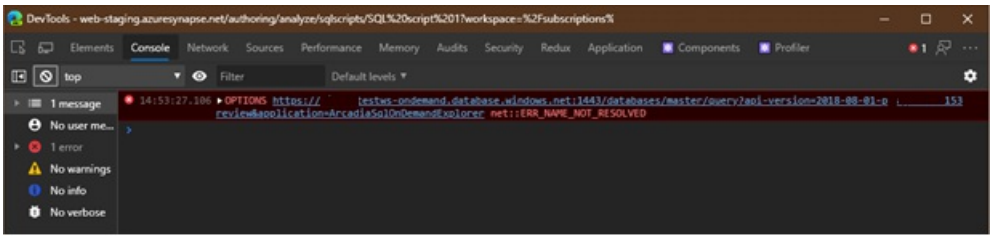
If the response header contains one of the above values, it means we should have already collected the failure information. You can open a support ticket if needed, and optionally attach the screenshot of the item details.

If you can't see the header, or the header does not have one of the values listed above, attach a screenshot of the item details when you open the ticket.



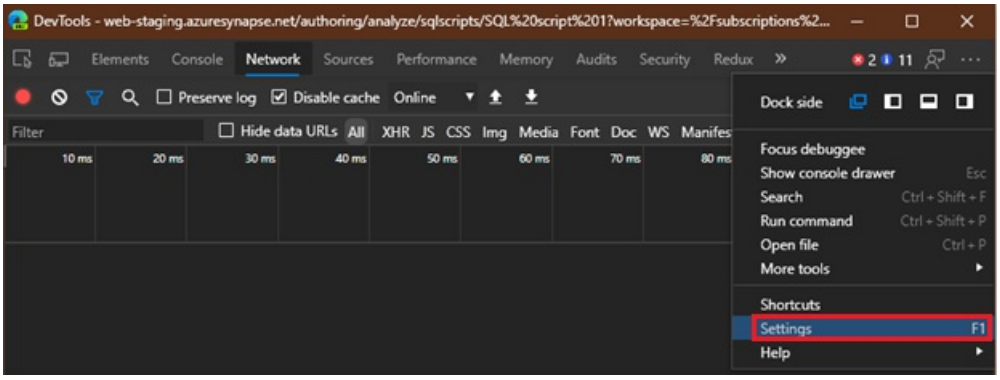
If the steps above do not solve your issue, you may need to open a support ticket. When submitting your support ticket, include the "Session ID" or "Diagnostic Info" downloaded at the beginning of this guide.

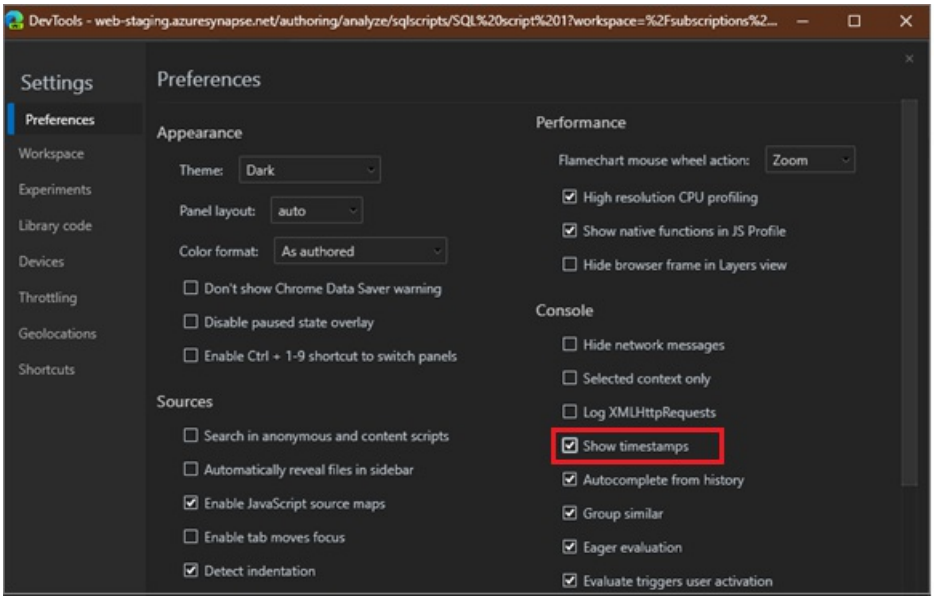
When reporting the issue, you may optionally take a screenshot of your "Console" tab in the "Developer Tools" and attach it to the support ticket. Scroll the content and take more than one screenshot if necessary to capture the entire message.



If you're attaching screenshots, provide the time (or an estimated time range) of when you took the screenshots. It will help us when looking into the problem.

Certain browsers support showing timestamps in the "Console" tab. For Chromium Edge/Chrome, open the "Settings" dialog in "Developer Tools", and check "Show timestamps" in "Preferences" tab.

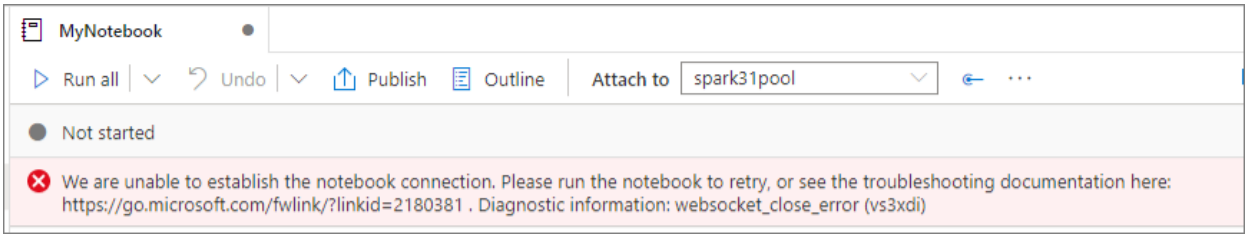




Notebook websocket connection issue

Symptom

Error message shows: your notebook connection has closed unexpectedly. To re-establish the connection, run the notebook again. Diagnostic information: websocket_close_error (correlation id)



Root cause:

Notebook execution depends on establishing a WebSocket connection to the following URL

```
wss://{workspace}.dev.azuresynapse.net/jupyterApi/versions/1/sparkPools/{spark-pool}/api/kernels/{kernel-id}/channels
```

- **{workspace}** is the name of Synapse workspace,
- **{spark-pool}** is the name of Spark pool you are currently working on,
- **{kernel-id}** is a GUID used for distinguishing notebook sessions.

When setting up WebSocket connection, Synapse Studio will include an access token (JWT bearer token) in the Sec-WebSocket-Protocol header of the WebSocket request.

Sometimes, WebSocket request might be blocked, or JWT token in the request header might be redacted in your network environment. This will cause Synapse Notebook unable to establish the connection to our server and run your notebook.

Action:

If possible, please try to switch your network environment, such as inside/outside corpnet, or access Synapse Notebook on another workstation.

- If you can run notebook on the same workstation but in a different network environment, please work with your network administrator to find out whether the WebSocket connection has been blocked.
- If you can run notebook on a different workstation but in the same network environment, please ensure you didn't install any browser plugin that may block the WebSocket request.

Otherwise, please contact your network administrator and ensure the outbound WebSocket requests with the following URL pattern is allowed and their request header is not redacted:

```
wss://{workspace}.dev.azuresynapse.net/{path}
```

- **{workspace}** is the Synapse workspace name;
- **{path}** indicates any sub-path (i.e., slash character is included) in URI.

This URL pattern is looser than the one shown in "Root Cause" section because it allows for us adding new WebSocket-dependent features to Synapse without any potential connectivity issue in the future.

Next steps

If the previous steps don't help to resolve your issue Create a support ticket

Troubleshoot Synapse Studio connectivity with PowerShell

8/19/2022 • 2 minutes to read • Edit Online

Azure Synapse Studio depends on a set of Web API endpoints to work properly. This guide will help you identify causes of connectivity issues when you're:

- configuring your local network (such as network behind a corporate firewall) for accessing Azure Synapse Studio.
- experiencing connectivity issues using Azure Synapse Studio.

Prerequisite

- PowerShell 5.0 or higher version on Windows, or
- PowerShell Core 6.0 or higher version on Windows or Linux.

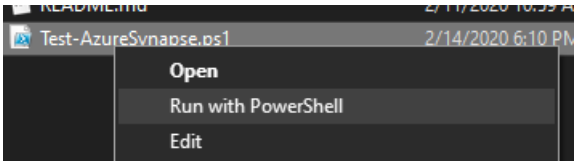
Troubleshooting steps

Right-click on the following link, and select "Save target as":

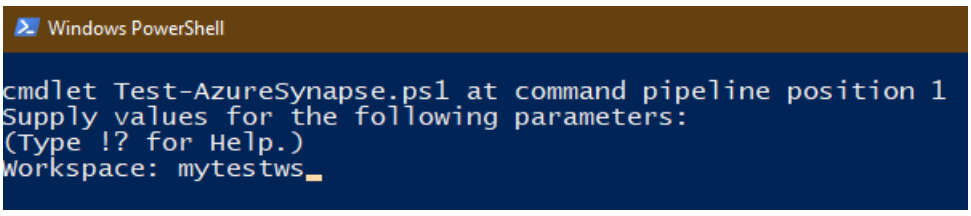
- Test-AzureSynapse.ps1

Alternatively, you may open the link directly, and save the opened script file. Don't save the address of the link above, as it may change in the future.

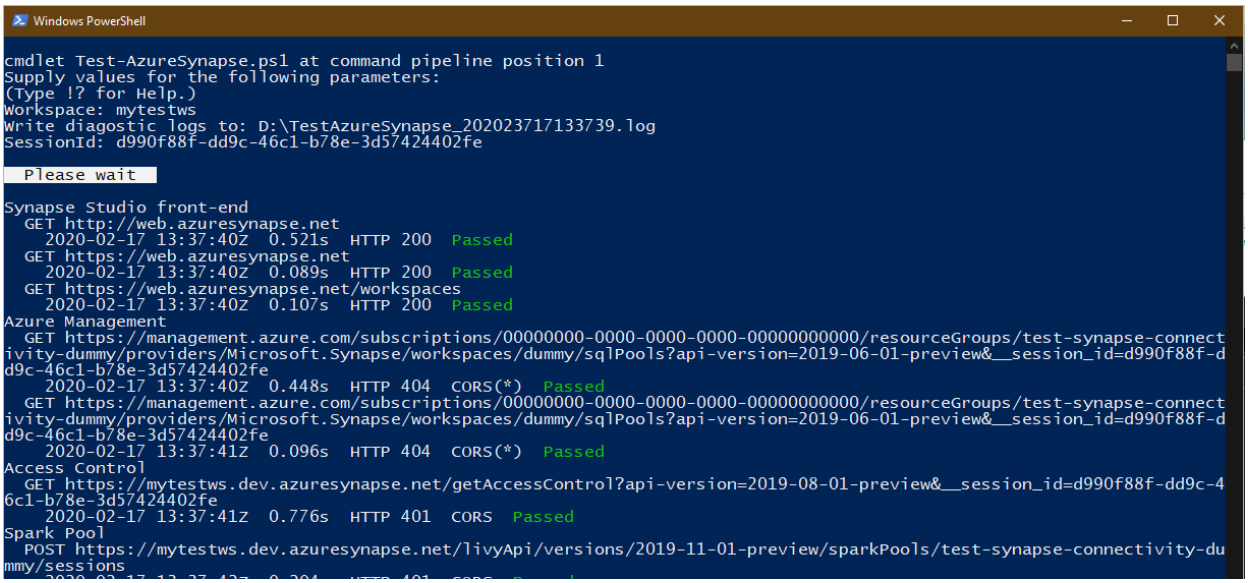
In file explorer, right-click on the downloaded script file, and select "Run with PowerShell".



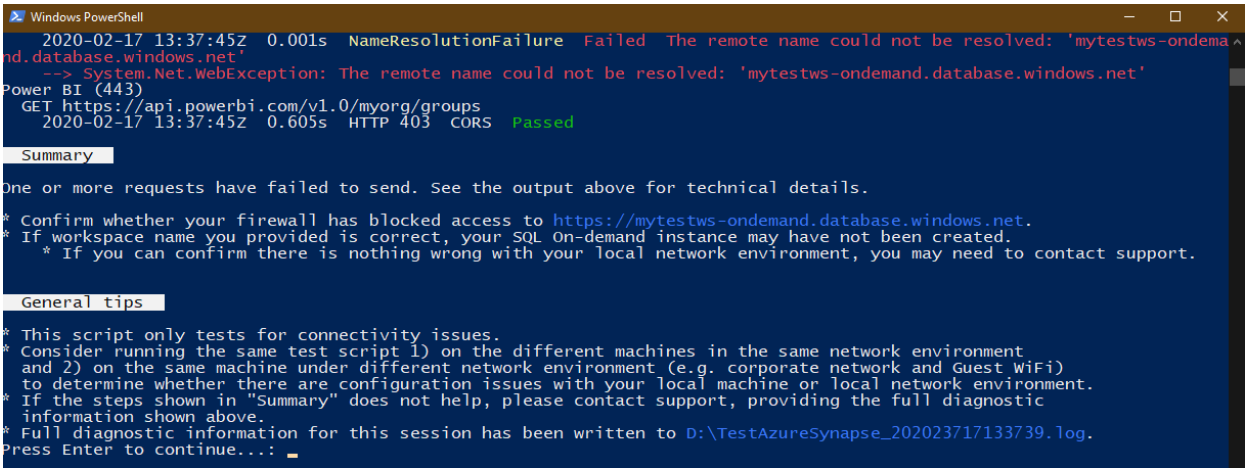
When prompted, enter the Azure Synapse workspace name that is currently experiencing a problem or that you want to test for connectivity, and press enter.



The diagnostic session will be started. Wait for it to complete.



In the end, a diagnosis summary will be shown. If your PC cannot connect to one or more of the endpoints, it will show some suggestions in the "Summary" section.



Additionally, a diagnostic log file for this session will be generated in the same folder as the troubleshooting script. Its location is shown in "General tips" section (`D:\TestAzureSynapse_2020....log`). You may send this file to technical support if necessary.

If you're a network administrator and tuning your firewall configuration for Azure Synapse Studio, the technical details shown above the "Summary" section may help.

- All the test items (requests) marked with "Passed" mean they have passed connectivity tests, regardless of the HTTP status code. For the failed requests, the reason is shown in yellow, such as `NamedResolutionFailure` or `ConnectFailure` . These reasons may help you figure out whether there are misconfigurations with your network environment.

Next steps

If the previous steps don't help to resolve your issue, create a support ticket.

Troubleshoot connectivity between Azure Synapse Analytics Synapse Studio and storage

8/19/2022 • 2 minutes to read • Edit Online

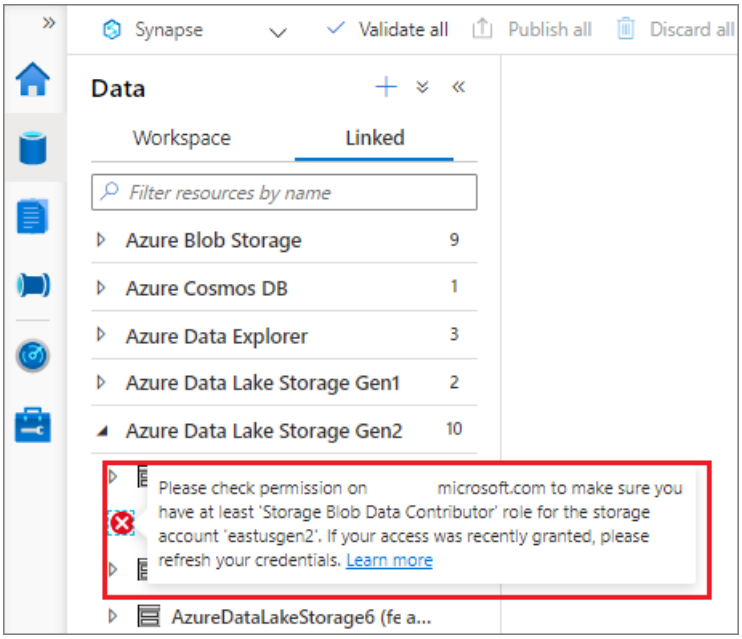
In Synapse Studio, you can explore data resources located in your linked storage. This guide will help you solve connectivity issues when you're trying to access your data resources.

Case #1: Storage account lacks proper permissions

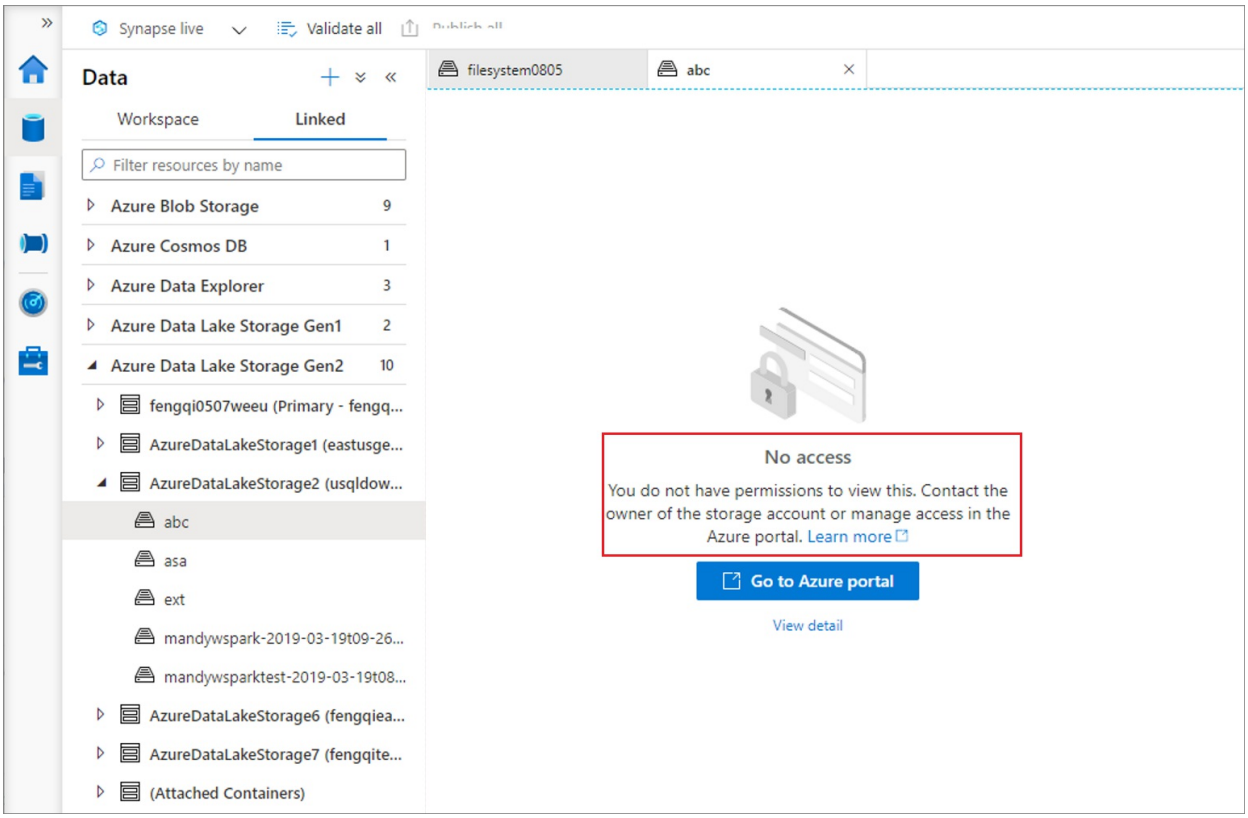
If your storage account lacks the proper permissions, you aren't able to expand the storage structure through "Data" --> "Linked" in Synapse Studio. See the issue symptom screenshot below.

The detailed error message may vary, but the general meaning of the error message is: "This request is not authorized to perform this operation.".

In the linked storage node:



In the storage container node:

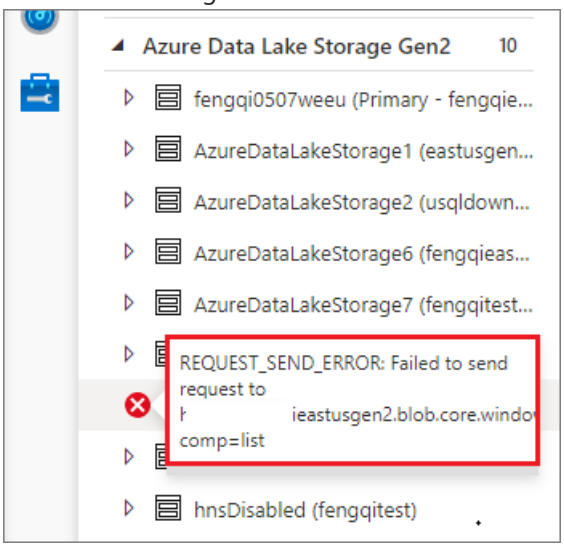


SOLUTION: To assign your account to the proper role, see Use the Azure portal to assign an Azure role for access to blob and queue data

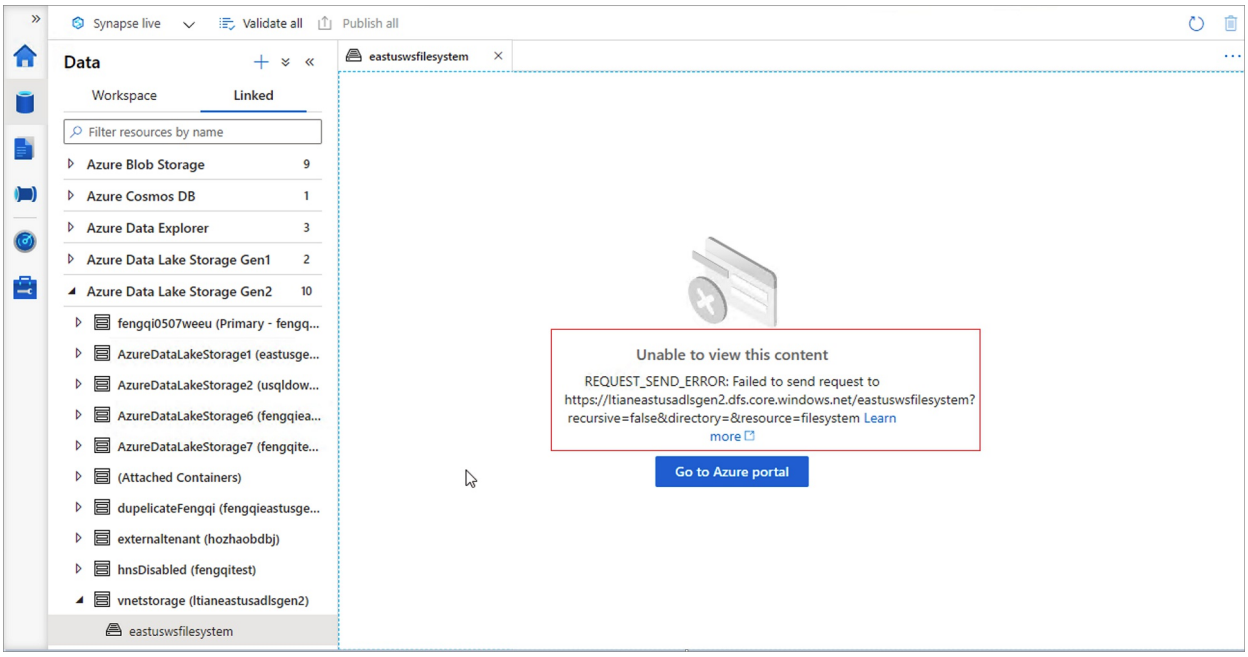
Case #2: Failed to send the request to storage server

When selecting the arrow to expand the storage structure in "Data" --> "Linked" in Synapse Studio, you may see the "REQUEST_SEND_ERROR" issue in the left panel. See the issue symptom screenshot below:

In the linked storage node:



In the storage container node:



There might be several possible reasons behind this issue:

The storage resource is behind a vNet and a storage private endpoint needs to configure

SOLUTION: In this case, you need to configure the storage private endpoint for your storage account. For how to configure the storage private endpoint for vNet, see Use the Azure portal to assign an Azure role for access to blob and queue data.

You can use the command "nslookup <storage-account-name>.dfs.core.windows.net" to check the connectivity after the storage private endpoint is configured. It should return a string similar to: "<storage-account-name>.privatelink.dfs.core.windows.net".

The storage resource is not behind a vNet but the Blob service (Azure AD) endpoint is not accessible due to firewall configured

SOLUTION: In this case, you need to open your storage account in the Azure portal. In the left navigation scroll down to **Support + troubleshooting** and select **Connectivity check** to check the **Blob service (Azure AD)** connectivity status. If it is not accessible, follow the promoted guide to check the **Firewalls and virtual networks** configuration under your storage account page. For more information about storage firewalls, see Configure Azure Storage firewalls and virtual networks.

Other issues to check

- The storage resource you are accessing is Azure Data Lake Storage Gen2 and is behind a firewall and vNet (with storage private endpoint configured) at the same time.
- The container resource you are accessing has been deleted or doesn't exist.
- Crossing-tenant: the workspace tenant that user used to login is not same with the tenant of the storage account.

Next steps

If the previous steps don't help to resolve your issue, create a support ticket.

Troubleshoot Azure Synapse Analytics workspaces created using SDK

8/19/2022 • 2 minutes to read • Edit Online

This article provides troubleshooting steps for launching Synapse Studio from a Synapse workspace that was created with a software development kit (SDK).

Prerequisites

- Synapse workspace created using SDK

Workaround

To launch Synapse Studio from your SDK created workspace, complete the following steps:

1. Create workspace by running `az synapse workspace create`.
2. Extract managed identity ID by running

   ```
   $identity=$(az synapse workspace show --name {workspace name} --resource-group {resource group name} --query "identity.principalId")
   ```

   .
3. Grant Storage Blob Data Contributor role to the managed identity storage account by running

   ```
   az role assignment create --role "Storage Blob Data Contributor" --assignee-object-id {identity } --scope {storage account resource id}.
   ```
4. Add firewall rule by running

   ```
   az synapse firewall-rule create --name allowAll --start-ip-address 0.0.0.0 --end-ip-address 255.255.255.255
   ```

   .

Next steps

- What is Azure Synapse
- Get Started

Troubleshoot reading UTF-8 text from CSV or Parquet files using serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

This article provides troubleshooting steps for reading UTF-8 text from CSV or Parquet files using serverless SQL pool in Azure Synapse Analytics.

When UTF-8 text is read from a CSV or PARQUET file using serverless SQL pool, some special characters like ü and ö are incorrectly converted if the query returns VARCHAR columns with non-UTF8 collations. This is a known issue in SQL Server and Azure SQL. Non-UTF8 collation is the default in Synapse SQL so customer queries will be affected. Customers who use standard English characters and some subset of extended Latin characters may not notice the conversion errors. The incorrect conversion is explained in more detail in Always use UTF-8 collations to read UTF-8 text in serverless SQL pool

Workaround

The workaround to this issue is to always use UTF-8 collation when reading UTF-8 text from CSV or PARQUET files.

- In many cases, you just need to set UTF8 collation on the database (metadata operation).

```
alter database MyDB
       COLLATE Latin1_General_100_BIN2_UTF8;
```

- You can explicitly define collation on VARCHAR column in OPENROWSET or external table:

```
select geo_id, cases = sum(cases)
from openrowset(
         bulk 'latest/ecdc_cases.parquet', data_source = 'covid', format = 'parquet'
     ) with ( cases int,
              geo_id VARCHAR(6) COLLATE Latin1_General_100_BIN2_UTF8 ) as rows
group by geo_id
```

- If you did not specify UTF8 collation on external tables that read UTF8 data, you need to re-create impacted external tables and set UTF8 collation on VARCHAR columns (metadata operation).

Next steps

- Query Parquet files with Synapse SQL
- Query CSV files with Synapse SQL
- CETAS with Synapse SQL
- Quickstart: Use serverless SQL pool

Move an Azure Synapse Analytics workspace from one region to another

8/19/2022 • 15 minutes to read • Edit Online

This article is a step-by-step guide that shows you how to move an Azure Synapse Analytics workspace from one Azure region to another.

> **NOTE**
> The steps in this article don't actually move the workspace. The steps show you how to create a new workspace in a new region by using Azure Synapse Analytics-dedicated SQL pool backups and artifacts from the source region.

Prerequisites

- Integrate the source region Azure Synapse workspace with Azure DevOps or GitHub. For more information, see Source control in Synapse Studio.
- Have Azure PowerShell and Azure CLI modules installed on the server where scripts are executed.
- Make sure all dependent services, for example, Azure Machine Learning, Azure Storage, and Azure Private Link hubs, are re-created in the target region or moved to the target region if the service supports a region move.
- Move Azure Storage to a different region. For more information, see Move an Azure Storage account to another region.
- Ensure the dedicated SQL pool name and the Apache Spark pool name are the same in the source region and the target region workspace.

Scenarios for a region move

- **New compliance requirements**: Organizations require data and services to be placed in the same region as part of new compliance requirements.
- **Availability of a new Azure region**: Scenarios where a new Azure region is available and there are project or business requirements to move the workspace and other Azure resources to the newly available Azure region.
- **Wrong region selected**: The wrong region was selected when the Azure resources were created.

Steps to move an Azure Synapse workspace to another region

Moving an Azure Synapse workspace from one region to another region is a multistep process. Here are the high-level steps:

1. Create a new Azure Synapse workspace in the target region along with a Spark pool with the same configurations as used in the source region workspace.
2. Restore the dedicated SQL pool to the target region by using restore points or geo-backups.
3. Re-create all the required logins on the new logical SQL Server.
4. Create serverless SQL pool and Spark pool databases and objects.
5. Add an Azure DevOps Service Principal to the Azure Synapse role-based access control (RBAC) Synapse Artifact Publisher role if you're using an Azure DevOps release pipeline to deploy the artifacts.
6. Deploy code artifact (SQL Scripts, Notebooks), linked services, pipelines, datasets, Spark Job definitions triggers, and credentials from Azure DevOps release pipelines to the target region Azure Synapse workspace.
7. Add Azure Active Directory (Azure AD) users or groups to Azure Synapse RBAC roles. Granting Storage Blob Contributor access to system-assigned managed identity (SA-MI) on Azure Storage and Azure Key Vault if you're authenticating by using managed identity.
8. Grant Storage Blob Reader or Storage Blob Contributor roles to required Azure AD users on default attached storage or on the Storage account that has data to be queried by using a serverless SQL pool.
9. Re-create self-hosted integration runtime (SHIR).

10. Manually upload all required libraries and jars in the target Azure Synapse workspace.
11. Create all managed private endpoints if the workspace is deployed in a managed virtual network.
12. Test the new workspace on the target region and update any DNS entries, which are pointing to the source region workspace.
13. If there's a private endpoint connection created on the source workspace, create one on the target region workspace.
14. You can delete the workspace in the source region after you test it thoroughly and route all the connections to the target region workspace.

Prepare

Step 1: Create an Azure Synapse workspace in a target region

In this section, you'll create the Azure Synapse workspace by using Azure PowerShell, the Azure CLI, and the Azure portal. You'll create a resource group along with an Azure Data Lake Storage Gen2 account that will be used as the default storage for the workspace as part of the PowerShell script and the CLI script. If you want to automate the process of deployment, invoke these PowerShell or CLI scripts from the DevOps release pipeline.

Azure portal

To create a workspace from the Azure portal, follow the steps in Quickstart: Create a Synapse workspace.

Azure PowerShell

The following script creates the resource group and Azure Synapse workspace by using the New-AzResourceGroup and New-AzSynapseWorkspace cmdlets.

Create a resource group

```
$storageAccountName= "<YourDefaultStorageAccountName>"
$resourceGroupName="<YourResourceGroupName>"
$regionName="<YourTargetRegionName>"
$containerName="<YourFileSystemName>" # This is the file system name
$workspaceName="<YourTargetRegionWorkspaceName>"

$sourcRegionWSName="<Your source region workspace name>"
$sourceRegionRGName="<YourSourceRegionResourceGroupName>"
$sqlUserName="<SQLUserName>"
$sqlPassword="<SQLStrongPassword>"

$sqlPoolName ="<YourTargetSQLPoolName>" #Both Source and target workspace SQL pool name will be same
$sparkPoolName ="<YourTargetWorkspaceSparkPoolName>"
$sparkVersion="2.4"

New-AzResourceGroup -Name $resourceGroupName -Location $regionName
```

Create a Data Lake Storage Gen2 account

```
#If the Storage account is already created, then you can skip this step.
New-AzStorageAccount -ResourceGroupName $resourceGroupName `
  -Name $storageAccountName `
  -Location  $regionName `
  -SkuName Standard_LRS `
  -Kind StorageV2 `
  -EnableHierarchicalNamespace $true
```

Create an Azure Synapse workspace

```
$password = ConvertTo-SecureString $sqlPassword -AsPlainText -Force
$creds = New-Object System.Management.Automation.PSCredential ($sqlUserName, $password)

New-AzSynapseWorkspace -ResourceGroupName $resourceGroupName `
                       -Name $workspaceName -Location $regionName `
                       -DefaultDataLakeStorageAccountName $storageAccountName `
                       -DefaultDataLakeStorageFilesystem $containerName `
                       -SqlAdministratorLoginCredential $creds
```

If you want to create the workspace with a Managed Virtual Network, add the extra parameter "ManagedVirtualNetwork" to the script. To learn more about the options available, see Managed Virtual

Network Config.

```
#Creating a managed virtual network configuration
$config = New-AzSynapseManagedVirtualNetworkConfig -PreventDataExfiltration -AllowedAadTenantIdsForLinking
ContosoTenantId

#Creating an Azure Synapse workspace
New-AzSynapseWorkspace -ResourceGroupName $resourceGroupName `
                       -Name $workspaceName -Location $regionName `
                       -DefaultDataLakeStorageAccountName $storageAccountName `
                       -DefaultDataLakeStorageFilesystem $containerName `
                       -SqlAdministratorLoginCredential $creds `
                             -ManagedVirtualNetwork $config
```

Azure CLI

This Azure CLI script creates a resource group, a Data Lake Storage Gen2 account, and a file system. Then it creates the Azure Synapse workspace.

Create a resource group

```
az group create --name $resourceGroupName --location $regionName
```

Create a Data Lake Storage Gen2 account

The following script creates a storage account and container.

```
# Checking if name is not used only then creates it.
$StorageAccountNameAvailable=(az storage account check-name --name $storageAccountName --subscription
$subscriptionId | ConvertFrom-Json).nameAvailable

if($StorageAccountNameAvailable)
{
Write-Host "Storage account Name is available to be used...creating storage account"

#Creating a Data Lake Storage Gen2 account
$storgeAccountProvisionStatus=az storage account create `
  --name $storageAccountName `
  --resource-group $resourceGroupName `
  --location $regionName `
  --sku Standard_GRS `
  --kind StorageV2 `
  --enable-hierarchical-namespace $true

($storgeAccountProvisionStatus| ConvertFrom-Json).provisioningState
}
else
{
    Write-Host "Storage account Name is NOT available to be used...use another name --    exiting the
script..."
    EXIT
}

#Creating a container in a Data Lake Storage Gen2 account

$key=(az storage account keys list -g $resourceGroupName -n $storageAccountName|ConvertFrom-Json)[0].value

$fileShareStatus=(az storage share create --account-name $storageAccountName --name $containerName --
account-key $key)

if(($fileShareStatus|ConvertFrom-Json).created -eq "True")
{
      Write-Host f"Successfully created the fileshare - '$containerName'"
}
```

Create an Azure Synapse workspace

```
az synapse workspace create `
  --name $workspaceName `
  --resource-group $resourceGroupName `
  --storage-account $storageAccountName `
  --file-system $containerName `
  --sql-admin-login-user $sqlUserName `
  --sql-admin-login-password $sqlPassword `
  --location $regionName
```

To enable a managed virtual network, include the parameter `--enable-managed-virtual-network` in the preceding script. For more options, see workspace managed virtual network.

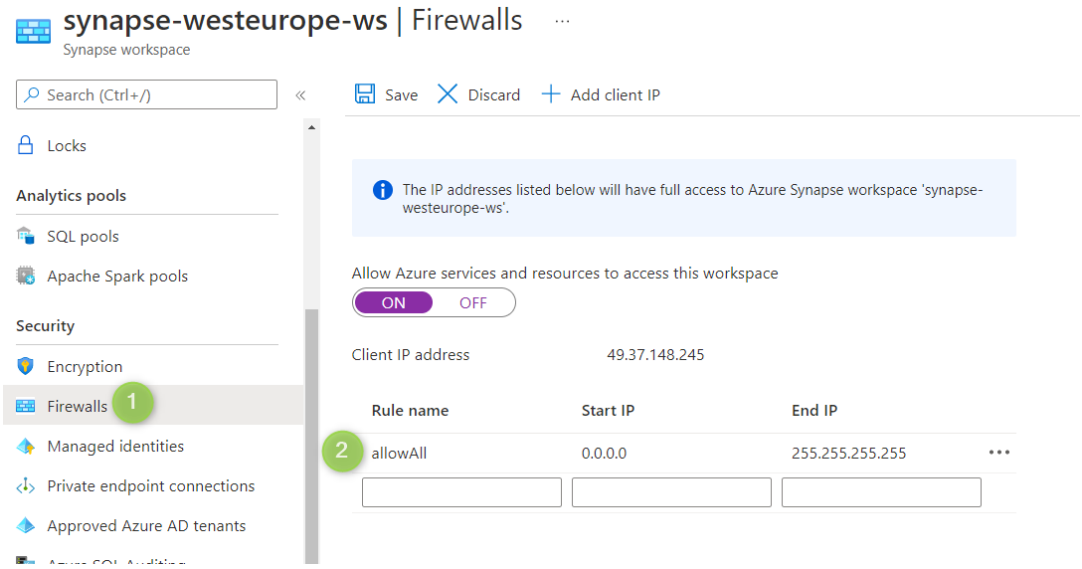
```
az synapse workspace create `
  --name $workspaceName `
  --resource-group $resourceGroupName `
  --storage-account $storageAccountName `
  --file-system $FileShareName `
  --sql-admin-login-user $sqlUserName `
  --sql-admin-login-password $sqlPassword `
  --location $regionName `
  --enable-managed-virtual-network true `
  --allowed-tenant-ids "Contoso"
```

Step 2: Create an Azure Synapse workspace firewall rule

After the workspace is created, add the firewall rules for the workspace. Restrict the IPs to a certain range. You can add a firewall from the Azure portal or by using PowerShell or the CLI.

Azure portal

Select the firewall options and add the range of IP addresses as shown in the following screenshot.



Azure PowerShell

Run the following PowerShell commands to add firewall rules by specifying the start and end IP addresses. Update the IP address range as per your requirements.

```
$WorkspaceWeb = (Get-AzSynapseWorkspace -Name $workspaceName -ResourceGroupName
$resourceGroup).ConnectivityEndpoints.Web
$WorkspaceDev = (Get-AzSynapseWorkspace -Name $workspaceName -ResourceGroupName
$resourceGroup).ConnectivityEndpoints.Dev

# Adding firewall rules
$FirewallParams = @{
  WorkspaceName = $workspaceName
  Name = 'Allow Client IP'
  ResourceGroupName = $resourceGroup
  StartIpAddress = "0.0.0.0"
  EndIpAddress = "255.255.255.255"
}
New-AzSynapseFirewallRule @FirewallParams
```

Run the following script to update the managed identity SQL control settings of the workspace:

```
Set-AzSynapseManagedIdentitySqlControlSetting -WorkspaceName $workspaceName -Enabled $true
```

Azure CLI

```
az synapse workspace firewall-rule create --name allowAll --workspace-name $workspaceName  `
--resource-group $resourceGroupName --start-ip-address 0.0.0.0 --end-ip-address 255.255.255.255
```

Run the following script to update the managed identity SQL control settings of the workspace:

```
az synapse workspace managed-identity grant-sql-access `
--workspace-name $workspaceName --resource-group $resourceGroupName
```

Step 3: Create an Apache Spark pool

Create the Spark pool with the same configuration that's used in the source region workspace.

Azure portal

To create a Spark pool from the Azure portal, see Quickstart: Create a new serverless Apache Spark pool using the Azure portal.

You can also create the Spark pool from Synapse Studio by following the steps in Quickstart: Create a serverless Apache Spark pool by using Synapse Studio.

Azure PowerShell

The following script creates a Spark pool with two workers and one driver node, and a small cluster size with 4 cores and 32-GB RAM. Update the values to match your source region workspace Spark pool.

```
#Creating a Spark pool with 3 nodes (2 worker + 1 driver) and a small cluster size with 4 cores and 32 GB
RAM.
New-AzSynapseSparkPool `
    -WorkspaceName  $workspaceName `
    -Name $sparkPoolName `
    -NodeCount 3 `
    -SparkVersion $sparkVersion `
    -NodeSize Small
```

Azure CLI

```
az synapse spark pool create --name $sparkPoolName --workspace-name $workspaceName --resource-group
$resourceGroupName `
--spark-version $sparkVersion --node-count 3 --node-size small
```

Move

Step 4: Restore a dedicated SQL pool

Restore from geo-redundant backups

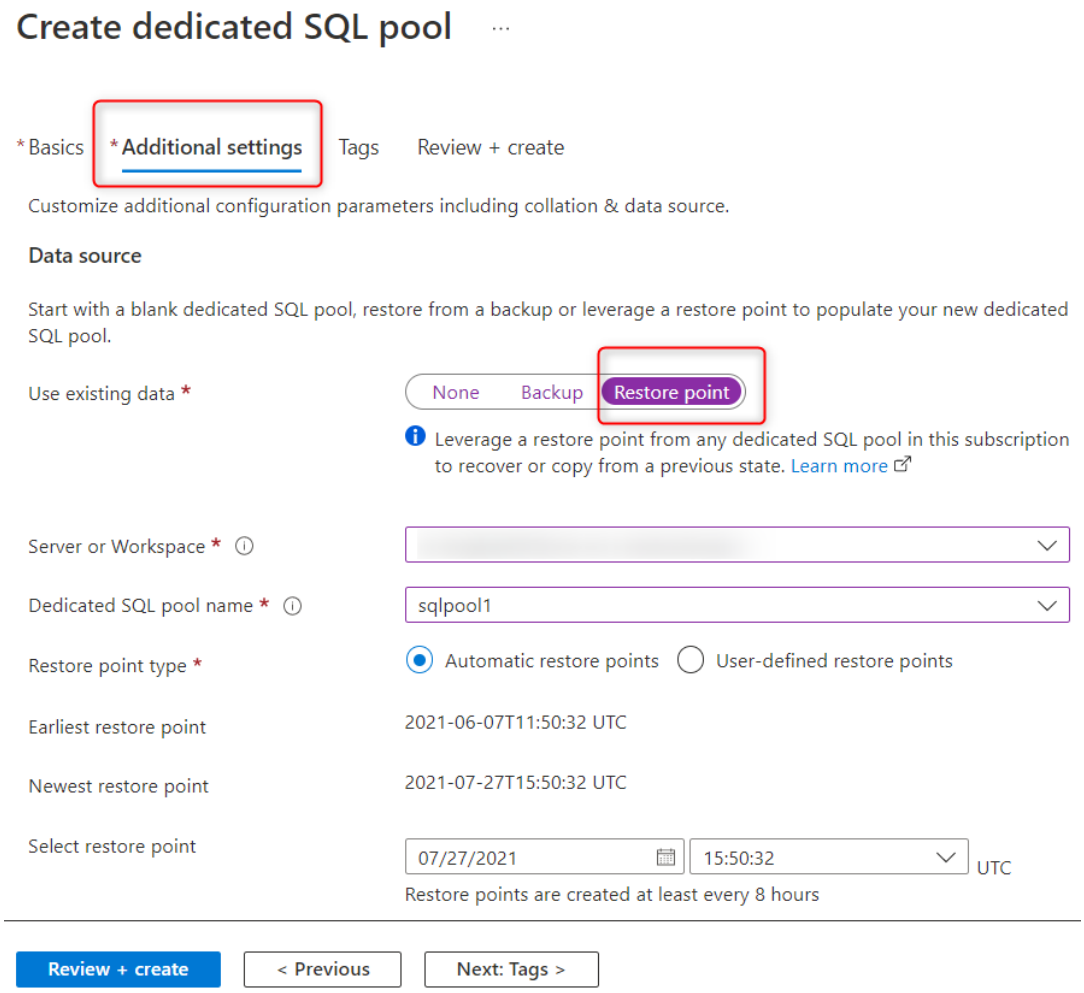
To restore the dedicated SQL pools from geo-backup by using the Azure portal and PowerShell, see Geo-restore a dedicated SQL pool in Azure Synapse Analytics.

Restore by using restore points from the source region workspace dedicated SQL pool

Restore the dedicated SQL pool to the target region workspace by using the restore point of the source region workspace dedicated SQL pool. You can use the Azure portal, Synapse Studio, or PowerShell to restore from restore points. If the source region isn't accessible, you can't restore by using this option.

Synapse Studio

From Synapse Studio, you can restore the dedicated SQL pool from any workspace in the subscription by using *restore points*. While you create the dedicated SQL pool, under **Additional settings**, select **Restore point** and select the workspace as shown in the following screenshot. If you created a user-defined restore point, use it to restore the SQL pool. Otherwise, you can select the latest automatic restore point.



Azure PowerShell

Run the following PowerShell script to restore the workspace. This script uses the latest restore point from the source workspace dedicated SQL pool to restore the SQL pool on the target workspace. Before you run the script, update the performance level from DW100c to the required value.

> **IMPORTANT**
> The dedicated SQL pool name should be the same on both the workspaces.

Get the restore points:

```
$restorePoint=Get-AzSynapseSqlPoolRestorePoint -WorkspaceName $sourceRegionWSName -Name $sqlPoolName|Sort-
Object  -Property RestorePointCreationDate -Descending `
                                                                                | SELECT
RestorePointCreationDate -ExpandProperty  RestorePointCreationDate -First 1
```

Transform the Azure Synapse SQL pool resource ID to SQL database ID because currently the command only

accepts the SQL database ID. For example:

```
/subscriptions/<SubscriptionId>/resourceGroups/<ResourceGroupName>/providers/Microsoft.Sql/servers/<WorkspaceName>/databases/<Datal
```

```
$pool = Get-AzSynapseSqlPool -ResourceGroupName $sourceRegionRGName -WorkspaceName $sourcRegionWSName -Name
$sqlPoolName
$databaseId = $pool.Id `
    -replace "Microsoft.Synapse", "Microsoft.Sql" `
    -replace "workspaces", "servers" `
    -replace "sqlPools", "databases"


$restoredPool = Restore-AzSynapseSqlPool -FromRestorePoint `
                                         -RestorePoint $restorePoint `
                                         -TargetSqlPoolName $sqlPoolName `
                                         -ResourceGroupName $resourceGroupName `
                                         -WorkspaceName $workspaceName `
                                         -ResourceId $databaseId `
                                         -PerformanceLevel DW100c -AsJob
```

The following tracks the status of the restore operation:

```
Get-Job | Where-Object Command -In ("Restore-AzSynapseSqlPool") | `
Select-Object Id,Command,JobStateInfo,PSBeginTime,PSEndTime,PSJobTypeName,Error |Format-Table
```

After the dedicated SQL pool is restored, create all the SQL logins in Azure Synapse. To create all the logins, follow the steps in Create login.

Step 5: Create a serverless SQL pool, Spark pool database, and objects

You can't back up and restore serverless SQL pool databases and Spark pools. As a possible workaround, you could:

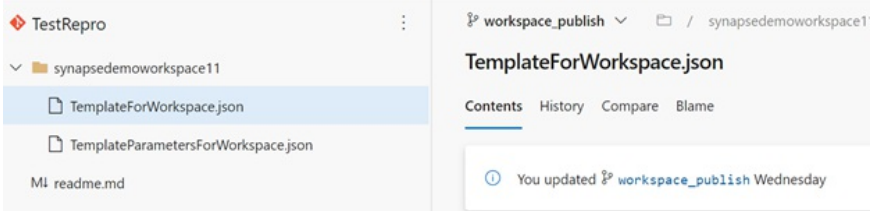
1. Create notebooks and SQL scripts, which have the code to re-create all the required Spark pool, serverless SQL pool databases, tables, roles, and users with all the role assignments. Check in these artifacts to Azure DevOps or GitHub.
2. If the Storage account name is changed, make sure the code artifacts are pointing to the correct storage account name.
3. Create pipelines, which invoke these code artifacts in a specific sequence. When these pipelines are executed on the target region workspace, the Spark SQL databases, serverless SQL pool databases, external data sources, views, roles, and users and permissions will be created on the target region workspace.
4. When you integrate the source region workspace with Azure DevOps, these code artifacts will be part of the repo. Later, you can deploy these code artifacts to the target region workspace by using the DevOps Release pipeline as mentioned in step 6.
5. On the target region workspace, trigger these pipelines manually.

Step 6: Deploy artifacts and pipelines by using CI/CD

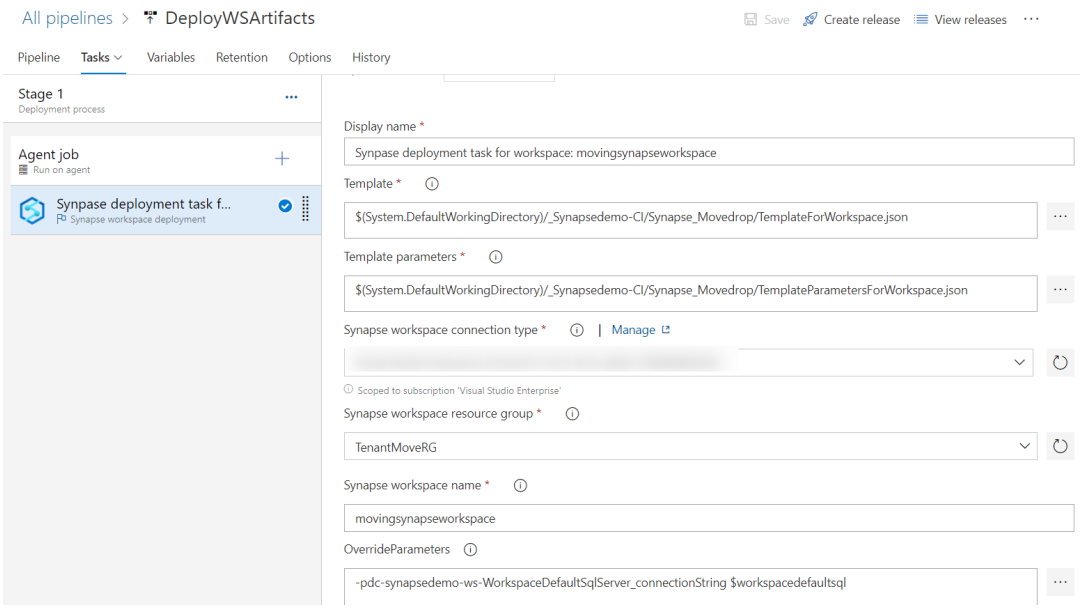
To learn how to integrate an Azure Synapse workspace with Azure DevOps or GitHub and how to deploy the artifacts to a target region workspace, follow the steps in Continuous integration and continuous delivery (CI/CD) for an Azure Synapse workspace.

After the workspace is integrated with Azure DevOps, you'll find a branch with the name workspace_publish. This branch contains the workspace template that includes definitions for the artifacts like Notebooks, SQL Scripts, Datasets, Linked Services, Pipelines, Triggers, and Spark job definition.

This screenshot from the Azure DevOps repo shows the workspace template files for the artifacts and other components.



You can use the workspace template to deploy artifacts and pipelines to a workspace by using the Azure DevOps release pipeline as shown in the following screenshot.



If the workspace isn't integrated with GitHub or Azure DevOps, you'll have to manually re-create or write custom PowerShell or Azure CLI scripts to deploy all the artifacts, pipelines, linked services, credentials, triggers, and Spark definitions on the target region workspace.

> **NOTE**
> This process requires you to keep updating the pipelines and code artifacts to include any changes made to Spark and serverless SQL pools, objects, and roles in the source region workspaces.

Step 7: Create a shared integration runtime

To create a SHIR, follow the steps in Create and configure a self-hosted integration runtime.

Step 8: Assign an Azure role to managed identity

Assign `Storage Blob Contributor` access to the managed identity of the new workspace on the default attached Data Lake Storage Gen2 account. Also assign access on other storage accounts where SA-MI is used for authentication. Assign `Storage Blob Contributor` or `Storage Blob Reader` access to Azure AD users and groups for all the required storage accounts.

Azure portal

Follow the steps in Grant permissions to workspace managed identity to assign a Storage Blob Data Contributor role to the managed identity of the workspace.

Azure PowerShell

Assign a Storage Blob Data Contributor role to the managed identity of the workspace.

Adding Storage Blob Data Contributor to the workspace managed identity on the storage account. The execution of `New-AzRoleAssignment` errors out with the message `Exception of type 'Microsoft.Rest.Azure.CloudException' was thrown.` however it creates the required permissions on the storage account.

```
$workSpaceIdentityObjectID= (Get-AzSynapseWorkspace -ResourceGroupName $resourceGroupName -Name
$workspaceName).Identity.PrincipalId
$scope =
"/subscriptions/$($subscriptionId)/resourceGroups/$($resourceGroupName)/providers/Microsoft.Storage/storageA
ccounts/$($storageAccountName)"

$roleAssignedforManagedIdentity=New-AzRoleAssignment -ObjectId $workSpaceIdentityObjectID `
    -RoleDefinitionName "Storage Blob Data Contributor" `
    -Scope $scope -ErrorAction SilentlyContinue
```

Azure CLI

Get the role name, resource ID, and principal ID for the workspace managed identity, then add the Storage Blob Data Contributor Azure role to the SA-MI.

```
# Getting Role name
$roleName =az role definition list --query "[?contains(roleName, 'Storage Blob Data Contributor')].
{roleName:roleName}" --output tsv

#Getting resource id for storage account
$scope= (az storage account show --name $storageAccountName|ConvertFrom-Json).id

#Getting principal ID for workspace managed identity
$workSpaceIdentityObjectID=(az synapse workspace show --name $workspaceName --resource-group
$resourceGroupName|ConvertFrom-Json).Identity.PrincipalId

# Adding Storage Blob Data Contributor Azure role to SA-MI
az role assignment create --assignee $workSpaceIdentityObjectID `
--role $roleName `
--scope $scope
```

Step 9: Assign Azure Synapse RBAC roles

Add all the users who need access to the target workspace with separate roles and permissions. The following PowerShell and CLI script adds an Azure AD user to the Synapse Administrator role in the target region workspace.

To get all the Azure Synapse RBAC role names, see Azure Synapse RBAC roles.

Synapse Studio

To add or delete Azure Synapse RBAC assignments from Synapse Studio, follow the steps in How to manage Azure Synapse RBAC role assignments in Synapse Studio.

Azure PowerShell

The following PowerShell script adds the Synapse Administrator role assignment to an Azure AD user or group. You can use -RoleDefinitionId instead of -RoleDefinitionName with the following command to add the users to the workspace:

```
New-AzSynapseRoleAssignment `
   -WorkspaceName $workspaceName  `
   -RoleDefinitionName "Synapse Administrator" `
   -ObjectId 1c02d2a6-ed3d-46ec-b578-6f36da5819c6

Get-AzSynapseRoleAssignment -WorkspaceName $workspaceName
```

To get the ObjectIds and RoleIds in the source region workspace, run the `Get-AzSynapseRoleAssignment` command. Assign the same Azure Synapse RBAC roles to the Azure AD users or groups in the target region workspace.

Instead of using `-ObjectId` as the parameter, you can also use `-SignInName`, where you provide the email address or the user principal name of the user. To find out more about the available options, see Azure Synapse RBAC - PowerShell cmdlet.

Azure CLI

Get the Object ID of the user and assign the required Azure Synapse RBAC permissions to the Azure AD user. You can provide the email address of the user (username@contoso.com) for the `--assignee` parameter.

```
az synapse role assignment create `
--workspace-name $workspaceName `
--role "Synapse Administrator" --assignee adasdasdd42-0000-000-xxx-xxxxxxx

az synapse role assignment create `
--workspace-name $workspaceName `
--role "Synapse Contributor" --assignee "user1@contoso.com"
```

To learn more about available options, see Azure Synapse RBAC - CLI.

Step 10: Upload workspace packages

Upload all required workspace packages to the new workspace. To automate the process of uploading the workspace packages, see the Microsoft Azure Synapse Analytics Artifacts client library.

Step 11: Permissions

To set up the access control for the target region Azure Synapse workspace, follow the steps in How to set up access control for your Azure Synapse workspace.

Step 12: Create managed private endpoints

To re-create the managed private endpoints from the source region workspace in your target region workspace, see Create a managed private endpoint to your data source.

Discard

If you wish to discard the target region workspace, delete the target region workspace. To do so, go to the resource group from your dashboard in the portal and select the workspace and select Delete at the top of the Resource group page.

Clean up

To commit the changes and complete the move of the workspace, delete the source region workspace after testing the workspace in the target region. To do so, go to the resource group that has the source region workspace from your dashboard in the portal and select the workspace and select Delete at the top of the Resource group page.

Next steps

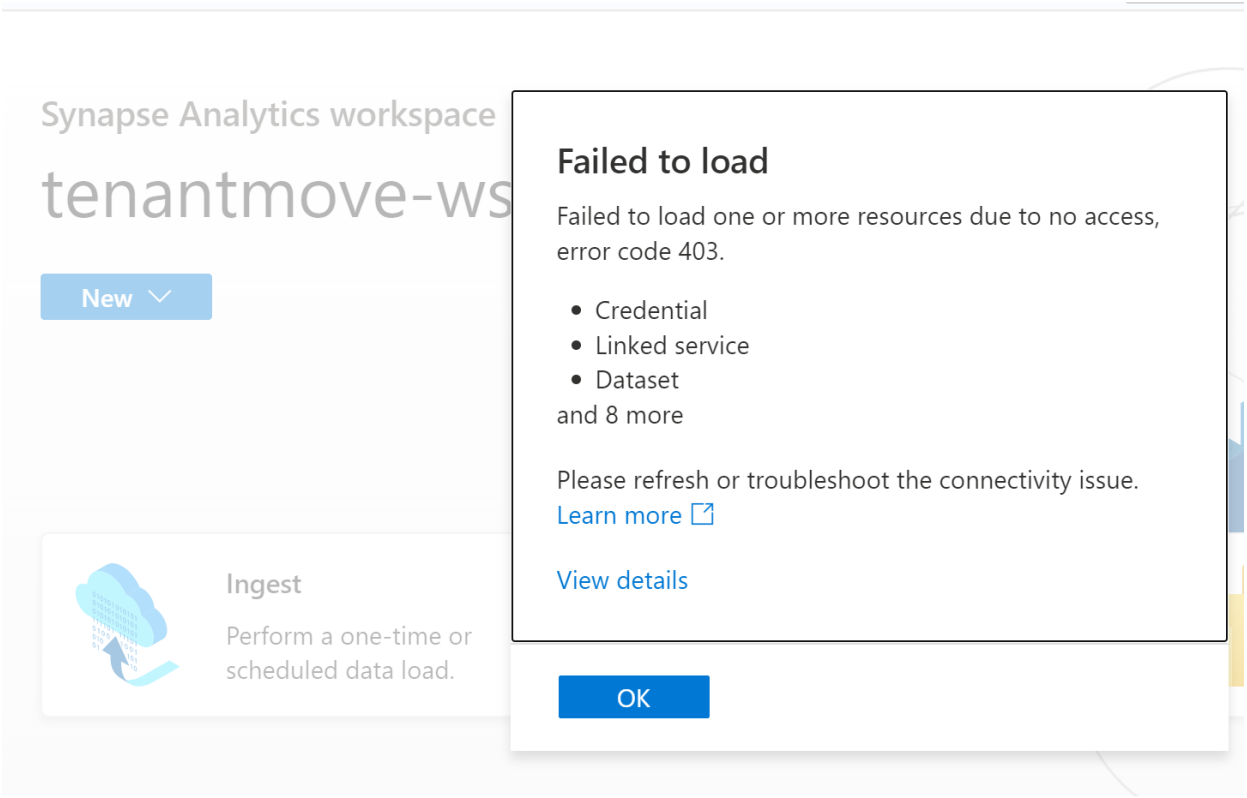
- Learn more about Azure Synapse managed virtual networks.
- Learn more about Azure Synapse managed private endpoints.
- Learn more about how to connect to workspace resources from a restricted network.

Recovering Synapse Analytics workspace after transferring a subscription to a different Azure AD directory (tenant)

8/19/2022 • 5 minutes to read • Edit Online

This article describes how to recover the Synapse Analytics workspace after transferring its subscription to a different Azure AD directory. The Synapse Analytics workspace will not be accessible after transferring a subscription to a different Azure AD directory (tenant).

When you try to launch the Synapse studio after the move, you will see the error: "Failed to load one or more resources due to no access, error code 403."



Follow the steps in this article after transferring a subscription across tenant to recover the Synapse Analytics workspace.

Transferring a subscription to a different Azure AD directory (tenant) is a complex process that must be carefully planned and executed. Azure Synapse Analytics require security principals (identities) to operate normally. When a subscription is moved to a different tenant, all principal IDs change, role assignments are deleted from Azure resource, and system assigned managed identities are dropped.

To understand the impact of transferring a subscription to another tenant see Transfer an Azure subscription to a different Azure AD directory

This article covers the steps involved in recovering a Synapse Analytics workspace after moving the subscription across tenants.

Pre-requisites

- To know more about service or resources impacted by tenant move see Transfer an Azure subscription to a different Azure AD directory.
- Save all the role assignment for Azure Active Directory (Azure AD) users, groups, and managed identities. This information can be used to assign the required permissions on Azure resources like Azure Synapse Analytics and ADLS Gen2 after tenant move. See Step 1: Prepare for the transfer
- Save all the permissions necessary for Azure AD users in dedicated and serverless SQL pool. Azure AD users will be deleted from the dedicated and serverless SQL pools after tenant move.

Steps for recovering Synapse Analytics workspace

After transferring the subscription to another tenant, follow the below steps to recover the Azure Synapse Analytics workspace.

1. Disable and re-enable the system Assigned Managed Identity. More information later in this article.
2. Assign Azure RBAC (role based access control) permissions to the required Azure AD users, groups, and managed identities on the Synapse Analytics workspace and required Azure resources.
3. Set the SQL Active Directory admin.
4. Re-create Azure AD users and groups based on their equivalent users and groups in the new Azure AD tenant for the dedicated and serverless SQL pools.
5. Assign Azure RBAC to Azure AD users, groups to Synapse Analytics workspace. This step should be first step after recovering the workspace. Without this step, launching Synapse Studio will throw 403 messages, due to Azure AD users not having permissions on the workspace:

   ```
   {"error":{"code":"Unauthorized","message":"The principal '<subscriptionid>' does not    have the
   required Synapse RBAC permission to perform this action. Required permission:    Action:
   Microsoft.Synapse/workspaces/read, Scope: workspaces/tenantmove-ws-1/*."}}
   ```

6. Assign Azure RBAC roles to Azure AD users, groups, service principals to all the resources used in the workspace artifacts, such as ADLS Gen2. For more information on Azure RBAC in ADLS Gen2, see Role-based access control (Azure RBAC).
7. Add Synapse RBAC role assignments to Azure AD users and groups. For more information, see How to manage Synapse RBAC role assignments in Synapse Studio
8. Recreate all the Azure AD logins and users in dedicated and serverless SQL pool. For more information, see SQL Authentication in Azure Synapse Analytics
9. Recreate all user assigned managed identity and assign user-assigned managed identity to the Synapse Analytics workspace. For more information, see Credentials in Azure Data Factory and Azure Synapse

> **NOTE**
> Ensure the following steps are executed only after confirming subscription is successfully moved to another tenant.

Disable and re-enable the system assigned managed identity for the Synapse Analytics workspace

This section shows you how to use Azure CLI or Azure PowerShell to disable and re-enable the System-assigned Managed Identity for your Azure Synapse Analytics workspace. Consider the following steps in either Azure CLI or Azure PowerShell.

- Azure CLI
- Azure PowerShell

```
$resourceGroupName="Provide the Resource group name"
$workspaceName="Provide the workspace name"
$subscriptionId="Provide the subscription Id"

$url =
"https://management.azure.com/subscriptions/$subscriptionId/resourceGroups/$resourceGroupName/providers/Micr
osoft.Synapse/workspaces/$workspaceName\?api-version=2021-06-01"
```

This next sample disables System Assigned Managed Identity for the workspace.

```
az rest --method patch --headers  Content-Type=application/json   `
--url  $url `
--body '{ \"identity\":{\"type\":\"None\"}}'
```

Workspace `provisioningState` should be **Succeeded** and the identity type should **None** after preceding command is executed. If you execute the following command, `provisioningState` value might be shown as Provisioning and will take few minutes to change the status to Succeeded. Value of the `provisioningState` should be Succeeded before re-enabling the System Assigned Managed Identity for the workspace.

To get the status of the workspace to get the provisioning status and identity type, use the following code snippet:

```
az rest --method GET --uri $uri
```

The resulting JSON should be similar to:

```
   {
  "id": "/subscriptions/<subscriptionid>/resourceGroups/TenantMove-RG/providers/Microsoft
Synapse/workspaces/tenantmove-ws",
  "identity": {
    "type": "None"
  },
  "location": "eastus",
  "name": "tenantmove-ws",
  "properties": {
    "connectivityEndpoints": {
      "dev": "https://tenantmove-ws.dev.azuresynapse.net",
      "sql": "tenantmove-ws.sql.azuresynapse.net",
      "sqlOnDemand": "tenantmove-ws-ondemand.sql.azuresynapse.net",
      "web": "https://web.azuresynapse.net?
workspace=%2fsubscriptions%2<subscriptionid>b%2fresourceGroups%2fTenantMove-
RG%2fproviders%2fMicrosoft.Synapse%2fworkspaces%2ftenantmove-ws"
    },
    "cspWorkspaceAdminProperties": {
      "initialWorkspaceAdminObjectId": "<object id>"
    },
    "defaultDataLakeStorage": {
      "accountUrl": "https://tenantmovedemowsstorage.dfs.core.windows.net",
      "filesystem": "demo",
      "resourceId": "/subscriptions/<subscriptionid>/resourceGroups/TenantMove-
RG/providers/Microsoft.Storage/storageAccounts/tenantmovedemowsstorage"
    },
    "encryption": {
      "doubleEncryptionEnabled": false
    },
    "extraProperties": {
      "WorkspaceType": "Normal"
    },
    "managedResourceGroupName": "tenantmove-ws-managed-rg",
    "privateEndpointConnections": [],
    "provisioningState": "Succeeded",
    "publicNetworkAccess": "Enabled",
    "sqlAdministratorLogin": "sqladminuser",
    "trustedServiceBypassEnabled": false,
    "workspaceUID": "<workspace UID>"
  },
  "resourceGroup": "TenantMove-RG",
  "tags": {},
  "type": "Microsoft.Synapse/workspaces"
}
```


The next command will re-enable the System Assigned Managed Identity for the workspace:

```
az rest --method patch --headers  Content-Type=application/json   `
--url  $url `
--body '{ \"identity\":{\"type\":\"SystemAssigned\"}}'
```

The next command will get you workspace status. The `provisioningState` value should be Succeeded. The `provisioningState` value will change from Provisioning to Succeeded. Identity type will be changed to **SystemAssigned**.

```
az rest --method GET --uri $uri
```

Next steps

- Transfer an Azure subscription to a different Azure AD directory
- Move an Azure Synapse Analytics workspace from one region to another

- Assign Azure RBAC (role based access control) permissions to the required Azure AD users, groups, and managed identities
- How to manage Synapse RBAC role assignments in Synapse Studio

What are Azure Synapse database templates?

8/19/2022 • 2 minutes to read • Edit Online

Data takes many forms as it moves from source systems to data warehouses and data marts with the intent to solve business problems. Database templates can help with the transformation of data into insights.

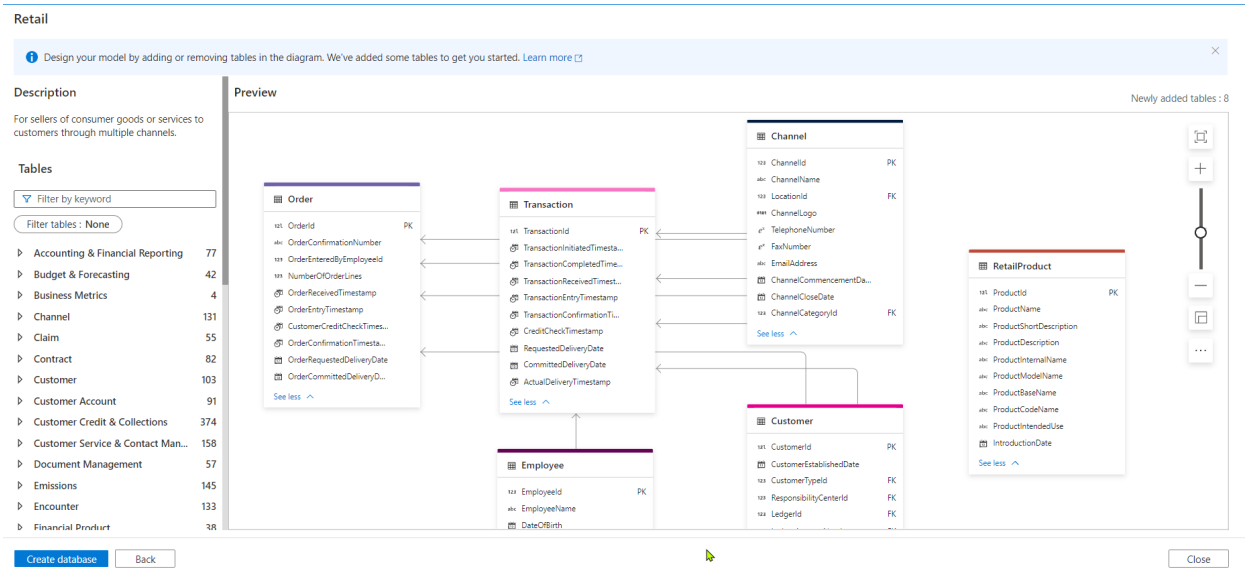
Database templates are a set of business and technical data definitions that are pre-designed to meet the needs of a particular industry. They act as blueprints that provide common elements derived from best practices, government regulations, and the complex data and analytic needs of an industry-specific organization.

These schema blueprints can be used by organizations to plan, architect, and design data solutions for data governance, reporting, business intelligence, and advanced analytics. The data models provide integrated business-wide information architectures that can help you implement, in a timely and predictable way, a proven industry data architecture.

For example, if you're building a product recommendation solution for your retail customers, you'll need to understand what the customer purchased and the transaction that led to the purchase. You may also need information about the store where the purchase was made, and whether the customer is part of a loyalty program. Just to accomplish this use case, consider the following schemas:

- Product
- Transaction
- TransactionLineItem
- Customer
- CustomerLoyalty
- Store

You can set up this use case by selecting the six tables in the **Retail** database template.



A typical database template addresses the core requirements of a specific industry and consists of:

- One or more enterprise templates.
- Tables grouped by **business areas**.

Available database templates

Currently, you can choose from the following database templates in Azure Synapse Studio to start creating your lake database:

- **Agriculture** - For companies engaged in growing crops, raising livestock, and dairy production.
- **Automotive** - For companies manufacturing automobiles, heavy vehicles, tires, and other automotive components.
- **Banking** - For companies providing a wide range of banking and related financial services.
- **Consumer Goods** - For manufacturers or producers of goods bought and used by consumers.
- **Energy & Commodity Trading** - For traders of energy, commodities, or carbon credits.
- **Freight & Logistics** - For companies that provide freight and logistics services.
- **Fund Management** - For companies that manage investment funds for investors.
- **Genomics** - For companies acquiring and analyzing genomic data about human beings or other species.
- **Healthcare Insurance** - For organizations providing insurance to cover healthcare needs (sometimes know as Payors).
- **Healthcare Provider** - For organizations providing healthcare services.
- **Life Insurance & Annuities** - For companies that provide life insurance, sell annuities, or both.
- **Manufacturing** - For companies engaged in discrete manufacturing of a wide range of products.
- **Oil & Gas** - For companies that are involved in various phases of the Oil & Gas value chain.
- **Pharmaceuticals** - For companies engaged in creating, manufacturing, and marketing pharmaceutical and bio-pharmaceutical products and medical devices.
- **Property & Casualty Insurance** - For companies that provide insurance against risks to property and various forms of liability coverage.
- **R&D and Clinical Trials** - For companies involved in research and development and clinical trials of pharmaceutical products or devices.
- **Retail** - For sellers of consumer goods or services to customers through multiple channels.
- **Utilities** - For gas, electric, and water utilities; power generators; and water desalinators.

As emission and carbon management is an important discussion in all industries, so we've included those components in all the available database templates. These components make it easy for companies who need to track and report their direct and indirect greenhouse gas emissions.

Next steps

Continue to explore the capabilities of the database designer using the links below.

- Database templates concept
- Quick start

Map Data in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

What is the Map data tool?

The Map Data tool is a guided process to help users create ETL mappings and mapping data flows from their source data to Synapse lake database tables without writing code. This process starts with the user choosing the destination tables in Synapse lake databases and then mapping their source data into these tables.

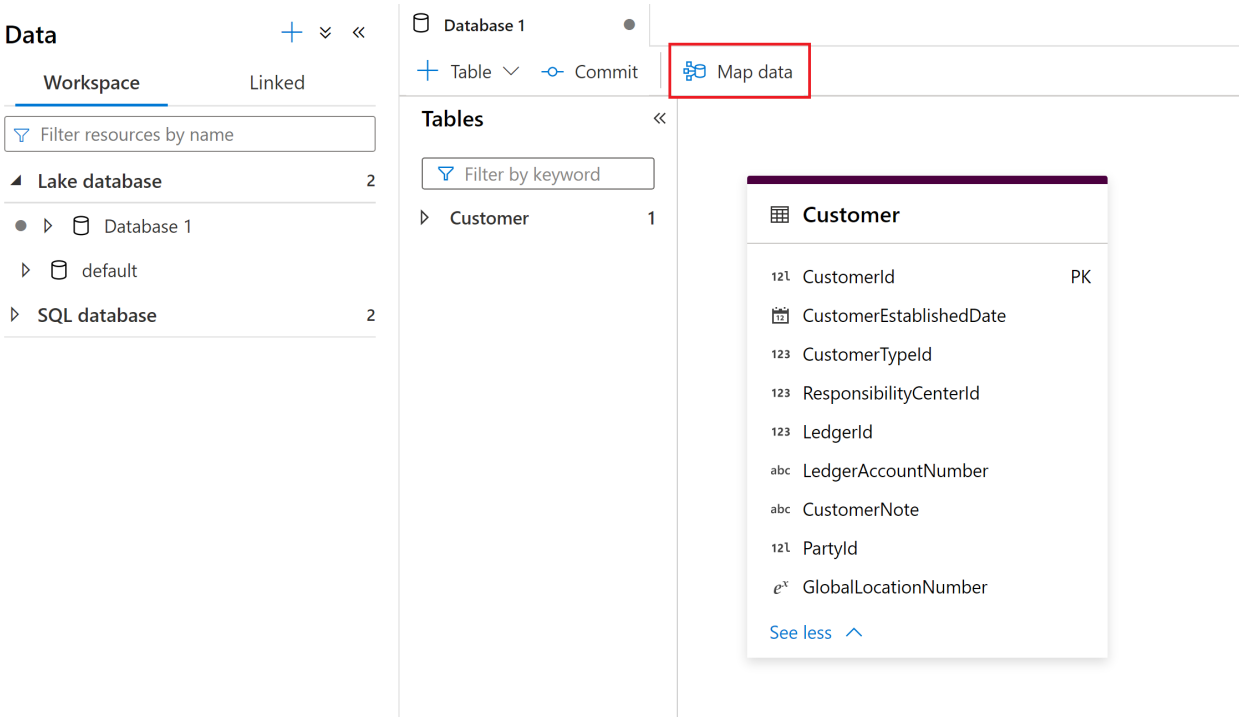
For more information on Synapse lake databases see Overview of Azure Synapse database templates - Azure Synapse Analytics | Microsoft Docs

Map Data provides for a guided experience where the user can generate a mapping data flow without having to start with a blank canvas and quickly generate a scalable mapping data flow runnable in Synapse pipelines.

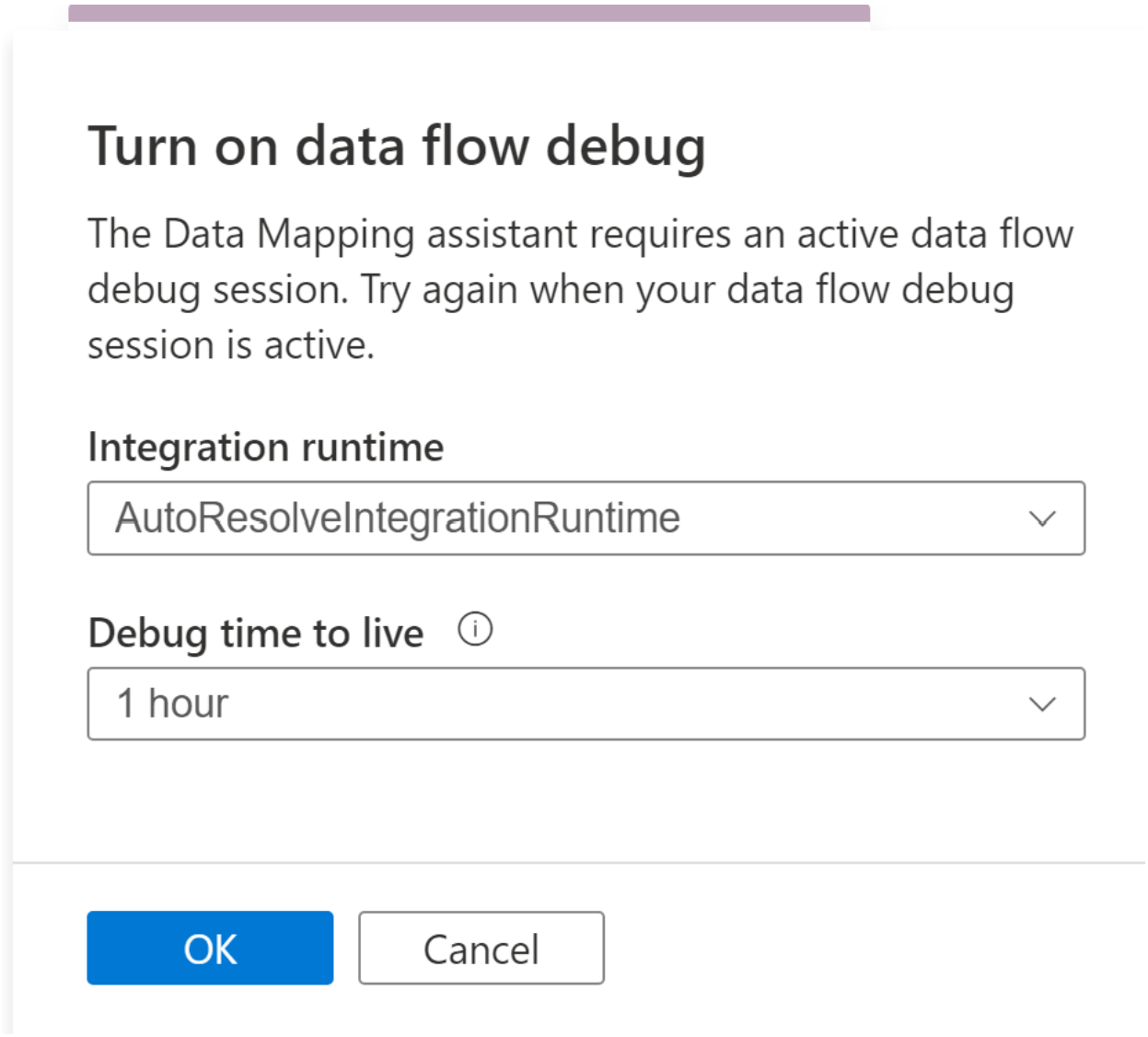
> **NOTE**
> The Map Data feature in Synapse Analytics pipelines is currently in public preview

Getting started

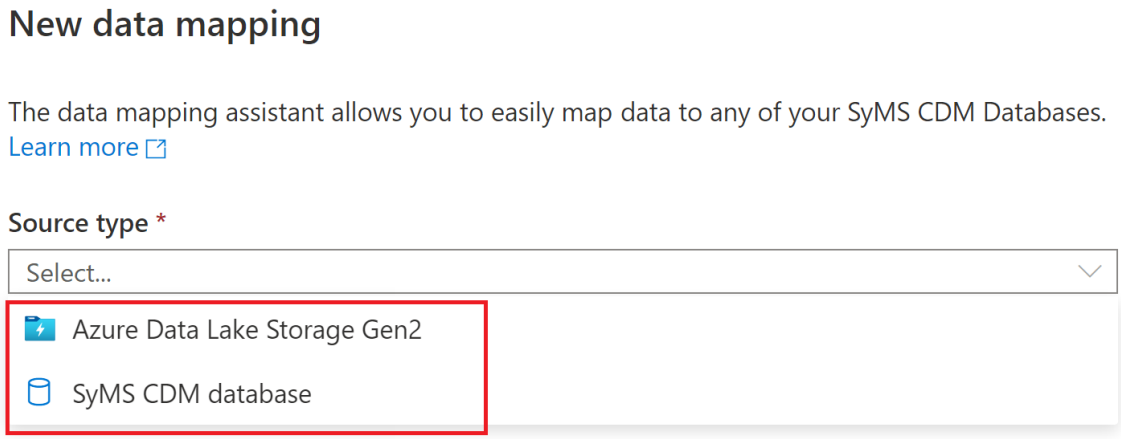
The Map Data tool is started from within the Synapse lake database experience. From here, you can select the Map Data tool to begin the process.



Map Data needs compute available to assist users with previewing data and reading schema of their source files. Upon using Map Data for the first time in a session you will need to warm up a cluster.



To begin, choose your data source that you want to map to your lake database tables. Currently supported data sources are Azure Data Lake Storage Gen 2 and Synapse lake databases.



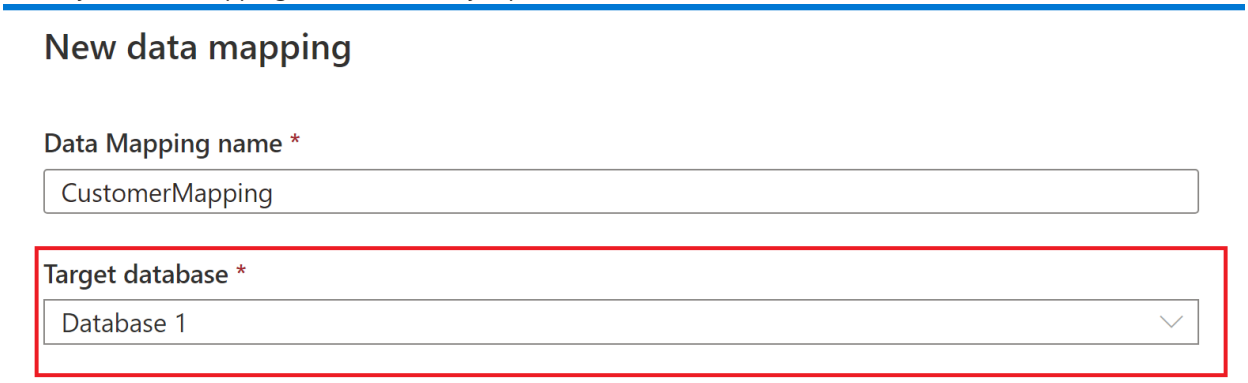
File type options

When choosing a file store such as Azure Data Lake Storage Gen 2 the following file types are supported:

- Common Data Model
- Delimited Text
- Parquet

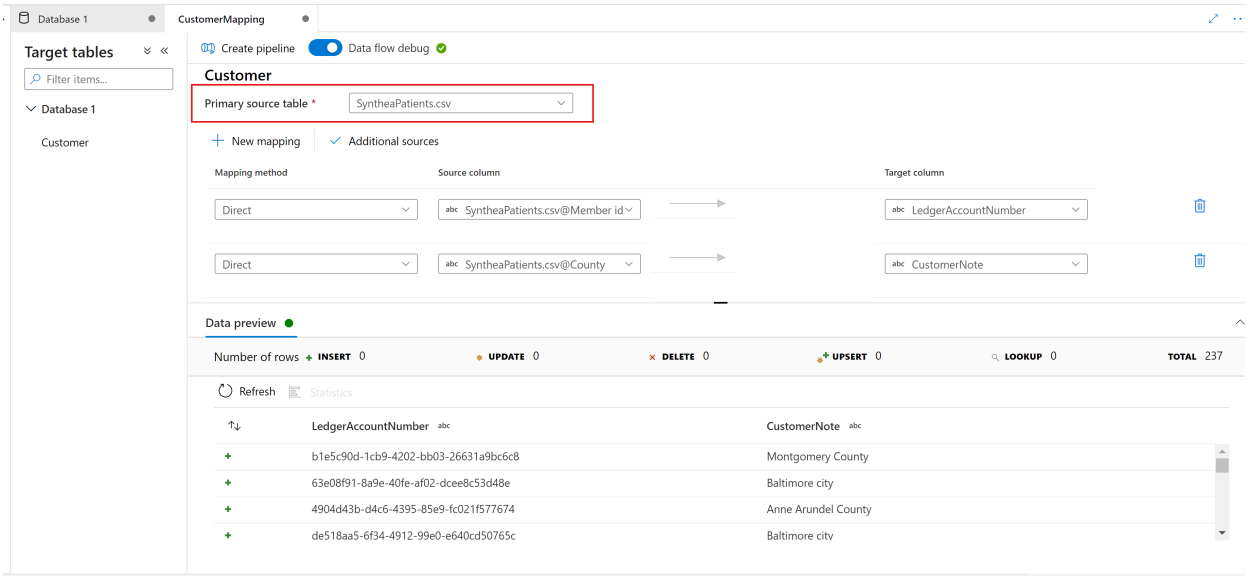
Create data mapping

Name your data mapping and select the Synapse lake database destination.



Source to target mapping

Choose a Primary source table to map to the Synapse lake database destination table.



New mapping

Use the New Mapping button to add a mapping method to create a mapping or transformation.

Additional source

Use the Additional source button to join to and add another source to your mapping.

Preview data

The **Data Preview** tab gives you an interactive snapshot of the data of each transform. For more information, see Data preview in debug mode.

Mapping methods

The following mapping methods are supported:

- Direct
- Surrogate Key
- Lookup
- Unpivot
- Aggregate

    - Sum
    - Minimum
    - Maximum
    - First
    - Last
    - Standard Deviation
    - Average
    - Mean
- Derived Column
    - Trim
    - Upper
    - Lower
    - Advanced

Create pipeline

Once you are done with your Map Data transformations select the Create pipeline button to generate a mapping data flow and pipeline to debug and run your transformation.

Quickstart: Create a new lake database leveraging database templates

8/19/2022 • 2 minutes to read • Edit Online

This quick start gives you a complete sample scenario on how you can apply database templates to create a lake database, align data to your new model, and use the integrated experience to analyze the data.

Prerequisites

- At least **Synapse User** role permissions are required for exploring a lake database template from Gallery.
- **Synapse Administrator**, or **Synapse Contributor** permissions are required on the Azure Synapse workspace for creating a lake database.
- **Storage Blob Data Contributor** permissions are required on data lake when using create table **From data lake** option.

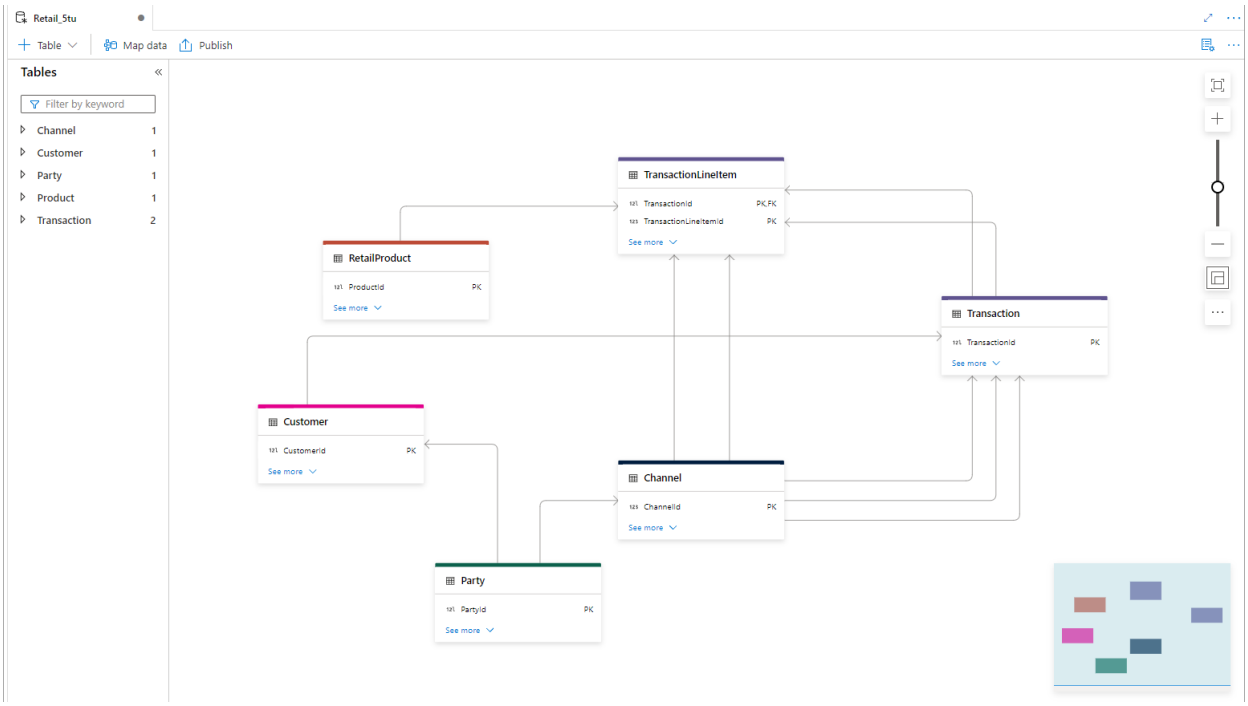
Create a lake database from database templates

Use the new database templates functionality to create a lake database that you can use to configure your data model for the database.

For our scenario we will use the `Retail` database template and select the following entities:

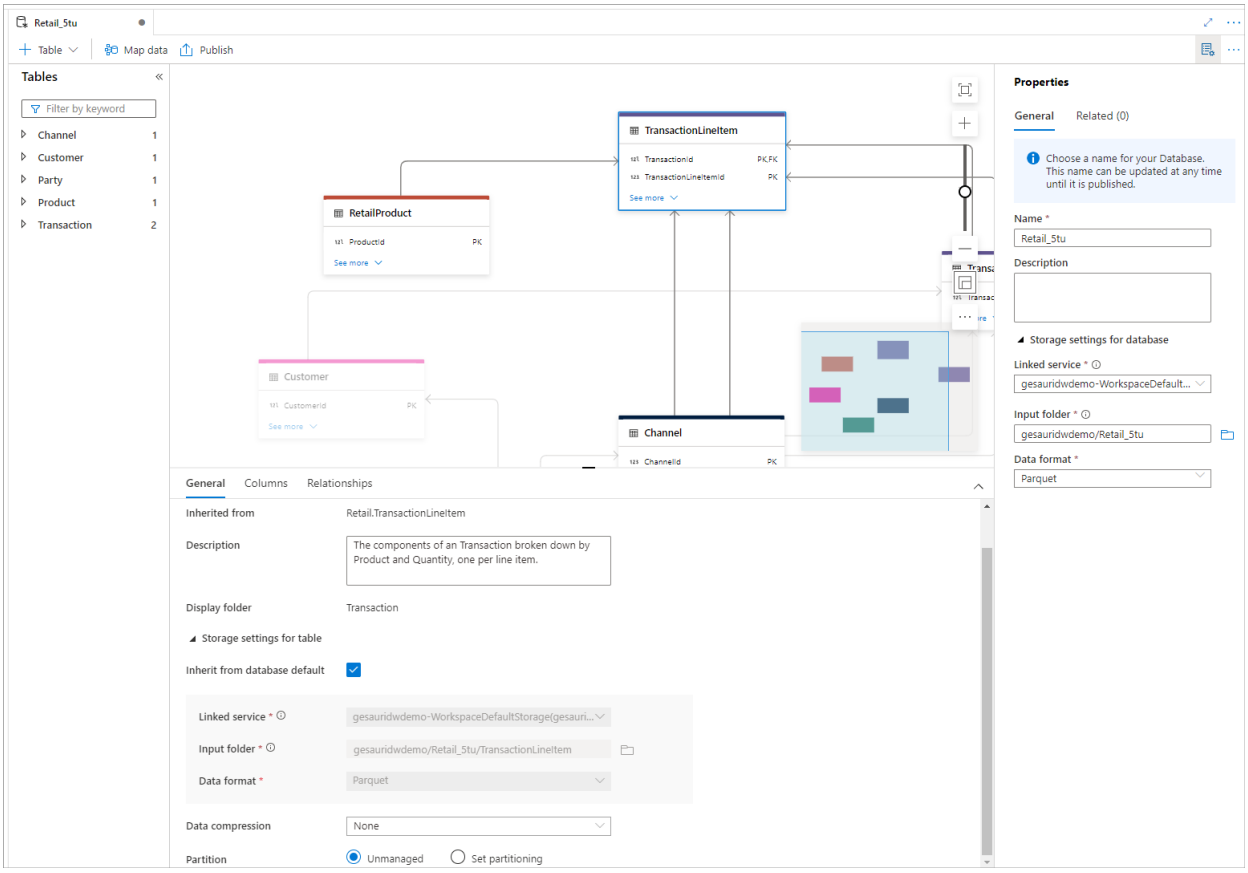
- **RetailProduct** - A product is anything that can be offered to a market that might satisfy a need by potential customers. That product is the sum of all physical, psychological, symbolic, and service attributes associated with it.
- **Transaction** - The lowest level of executable work or customer activity. A transaction consists of one or more discrete events.
- **TransactionLineItem** - The components of a Transaction broken down by Product and Quantity, one per line item.
- **Party** - A party is an individual, organization, legal entity, social organization, or business unit of interest to the business.
- **Customer** - A customer is an individual or legal entity that has or has purchased a product or service.
- **Channel** - A channel is a means by which products or services are sold and/or distributed.

The easiest way to find entities is by using the search box above the different business areas that contain the tables.



Configure lake database

After you have created the database, make sure the storage account and the filepath is set to a location where you wish to store the data. The path will default to the primary storage account within Azure Synapse Analytics but can be changed to your needs.



To save your layout and make it available within Azure Synapse, **Publish** all changes. This step completes the setup of the lake database and makes it available to all components within Azure Synapse Analytics and outside.

Ingest data to lake database

To ingest data to the lake database, you can execute pipelines with code free data flow mappings, which have a

Workspace DB connector to load data directly to the database table. You can also use the interactive Spark notebooks to ingest data to the lake database tables:

```
%%sql
INSERT INTO `retail_mil`.`customer` VALUES (1,'2021-02-18',1022,557,101,'Tailspin Toys (Head
Office)','Waldemar Fisar',90410,466);
```

Query the data

After the lake database is created, there are different ways to query the data. Currently, SQL databases in serverless SQL pools are supported and automatically understand the newly created lake database format.

```
SELECT TOP (100) [ProductId]
,[ProductName]
,[ProductDescription]
,[ProductInternalName]
,[ItemSku]
,[PrimaryBrandId]
FROM [Retail_mil].[dbo].[RetailProduct]
```

The other way to access the data within Azure Synapse is to open a new Spark notebook and use the integrated experience there:

```
df = spark.sql("SELECT * FROM `Retail_mil`.`RetailProduct`")
df.show(10)
```

Train machine learning models

You can use lake database to train your machine learning models and score the data. For more details Train machine learning models

Next steps

Continue to explore the capabilities of the database designer using the links below.

- Database templates concept
- Lake database concept
- How to create a lake database

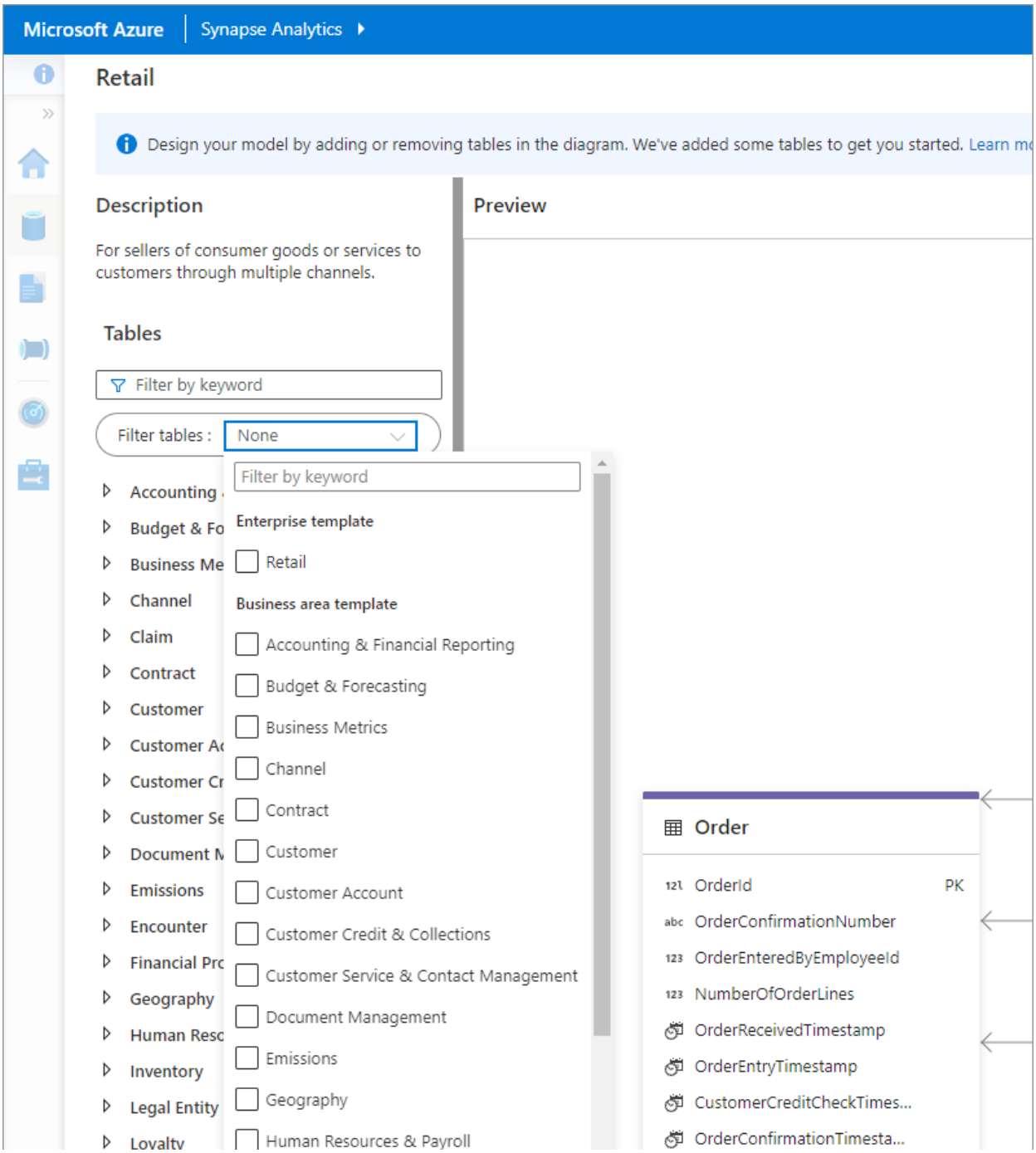
Lake database templates

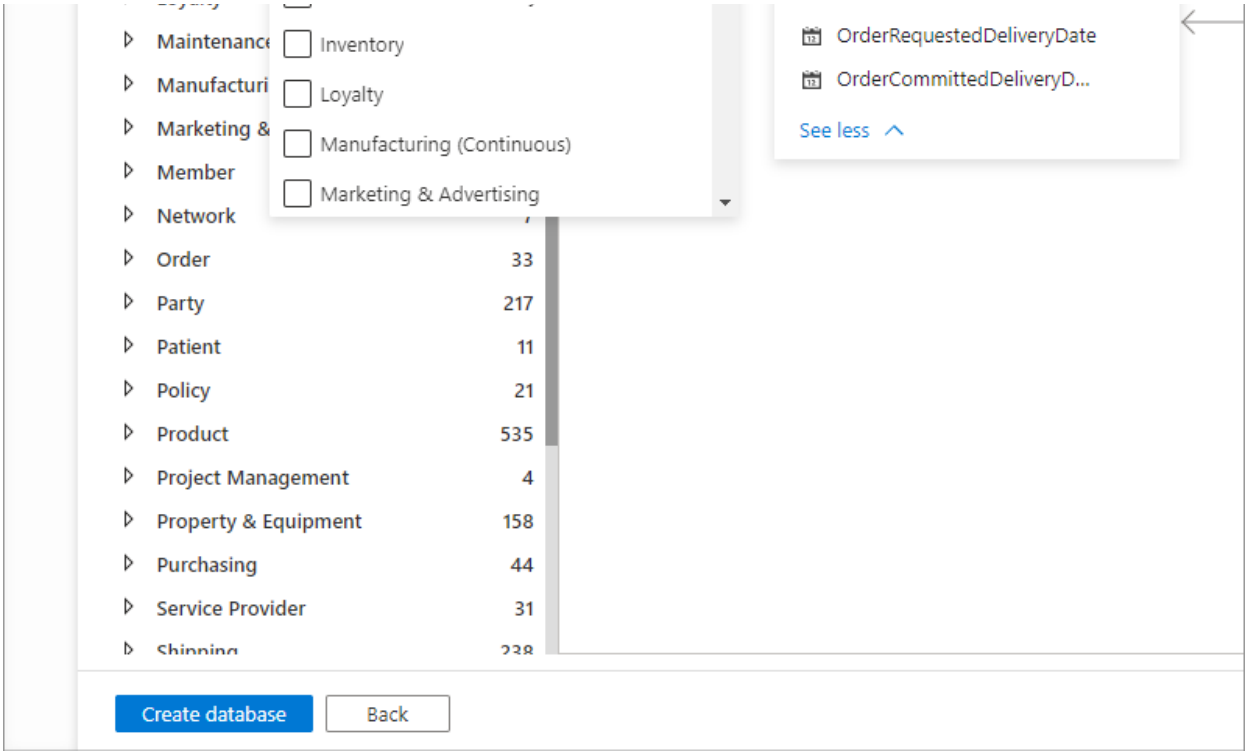
8/19/2022 • 2 minutes to read • Edit Online

Azure Synapse Analytics provides industry specific database templates to help standardize data in the lake. These templates provide schemas for predefined business areas, enabling data to be loaded into a lake database in a structured way. Use these templates to create your lake database and use Azure Synapse analytical runtime to provide insights to business users.

Enterprise templates

Enterprise database templates contain a subset of tables that are most likely to be of interest to an organization within a specific industry. It provides a high-level overview and describes the connectivity between the related business areas. These templates serve as an accelerator for many types of large projects. For example, the retail template has one enterprise template called "Retail".





Table

A table is an object with an independent existence that can be differentiated from other objects. For example, Customer, Store, Channel, and so on.

Column

Each table is described by a set of columns that represent the values and data that make up the table. Each column has a name, description, data type and is associated with a table. There are around 30,000 columns in the database templates. For example, CustomerId is a column in the table Customer.

Primary key

The primary key helps to uniquely identify the whole table. It must uniquely identify tuples in a table. For example, a key on the CustomerId column enforces uniqueness, to identify each customer in the table Customer.

Foreign key

A foreign key is a column or a combination of columns whose values match a primary key in a different table. It helps to establish a link between two tables. For example, CustomerId in Transaction table represents a customer who has completed a transaction. A foreign key always has a relationship to a primary key, for example, to the CustomerId primary key in the Customer table.

Composite key

A composite key is one that is composed of two or more columns that are together required to uniquely identify a record in a table. For example, in an Order table, both OrderNumber and ProductId may be required to uniquely identify a record.

Relationships

Relationships are associations or interactions between any two tables. For example, the tables Customer and CustomerEmail are related to each other. There are two tables involved in a relationship. There's a parent table

and a child table, often connected by a foreign key. You might say that the relationship is From table To table.

Table partitions

Lake database allows for the underlying data to be partitioned for a table for better performance. You can set partition configuration in the storage settings of a table in database editor.

Next steps

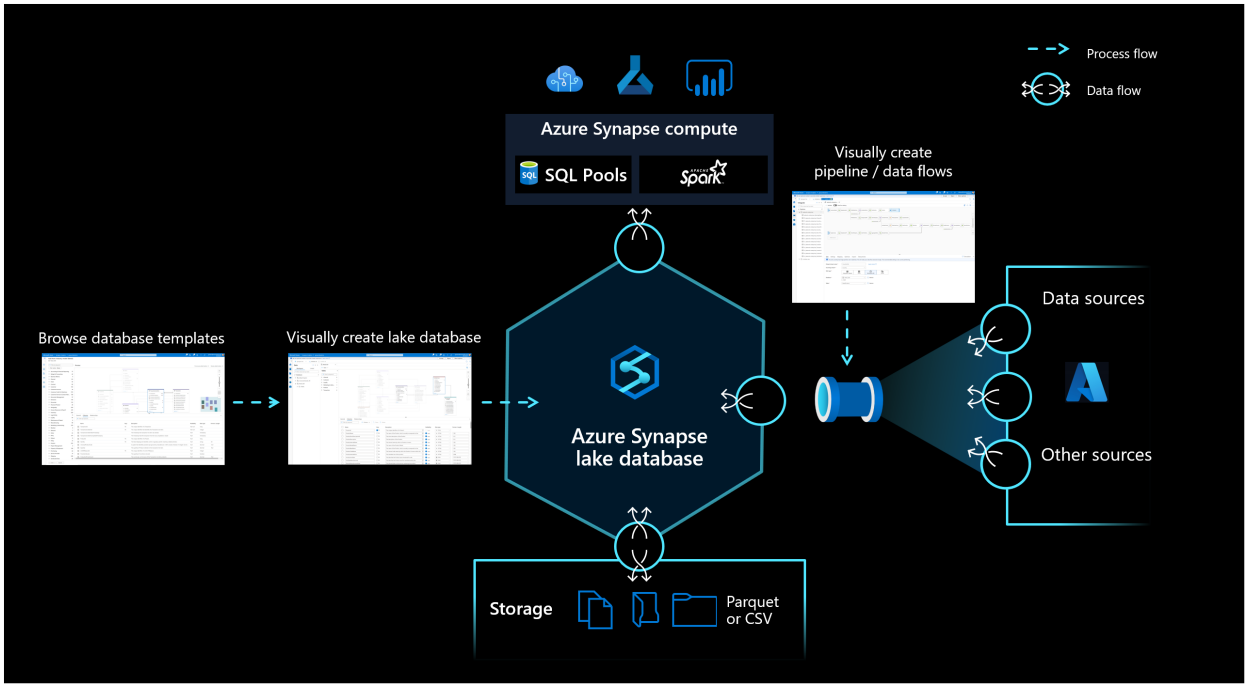
Continue to explore the capabilities of the database designer using the links below.

- Quick start
- Lake database Concept

Lake database

8/19/2022 • 2 minutes to read • Edit Online

The lake database in Azure Synapse Analytics enables customers to bring together database design, meta information about the data that is stored and a possibility to describe how and where the data should be stored. Lake database addresses the challenge of today's data lakes where it is hard to understand how data is structured.



Database designer

The new database designer gives you the possibility to create a data model for your lake database and add additional information to it. Every Entity and Attribute can be described to provide more information about the model, which not only contains Entities but relationships as well. In particular, the inability to model relationships has been a challenge for the interaction on the data lake. This challenge is now addressed with an integrated designer that provides possibilities that have been available in databases but not on the lake. Also the capability to add descriptions and possible demo values to the model allows people who are interacting with it in the future to have information where they need it to get a better understanding about the data.

Data storage

Lake databases use a data lake on the Azure Storage account to store the data of the database. The data can be stored in Parquet or CSV format and different settings can be used to optimize the storage. Every lake database uses a linked service to define the location of the root data folder. For every entity, separate folders are created by default within this database folder on the data lake. By default all tables within a lake database use the same format but the formats and location of the data can be changed per entity if that is requested.

Database compute

The lake database is exposed in Synapse SQL serverless SQL pool and Apache Spark providing users with the capability to decouple storage from compute. The metadata that is associated with the lake database makes it easy for different compute engines to not only provide an integrated experience but also use additional

information (for example, relationships) that was not originally supported on the data lake.

Next steps

Continue to explore the capabilities of the database designer using the links below.

- Create lake database quick start
- Database templates Concept

How-to: Create an empty lake database

8/19/2022 • 4 minutes to read • Edit Online

In this article, you'll learn how to create an empty lake database in Azure Synapse Analytics using the database designer. The database designer allows you to easily create and deploy a database without writing any code.

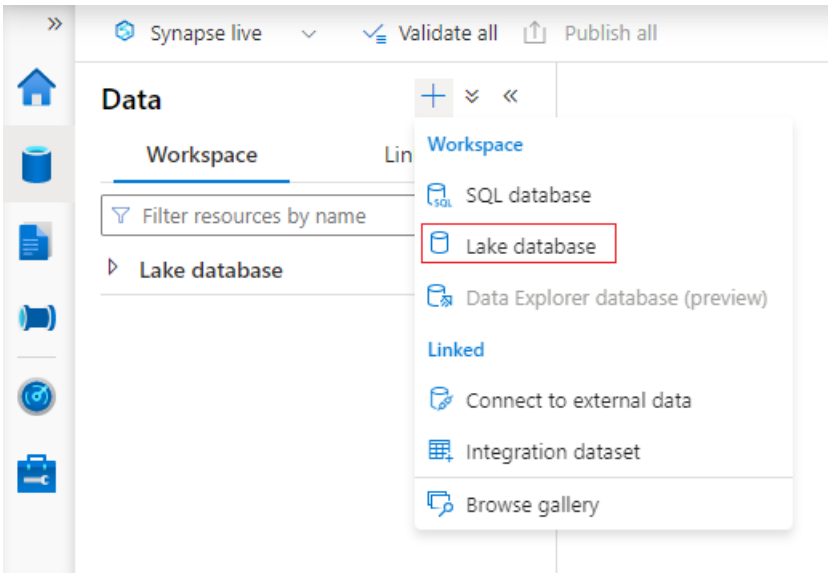
Prerequisites

- At least Synapse User role permissions are required for exploring a lake database template from Gallery.
- Synapse Administrator, Synapse Contributor, or Synapse Artifact Publisher permissions are required on the Synapse workspace for creating a lake database.
- Storage Blob Data Contributor permissions are required on data lake when using the create table **From data lake** option.

Create lake database from database template

1. From your Azure Synapse Analytics workspace **Home** hub, select the **Data** tab on the left. The **Data** tab will open and you will see the list of databases that already exist in your workspace.

2. Hover over the **+** button and select, then choose **Lake database**.



3. The database designer tab will open with an empty database.

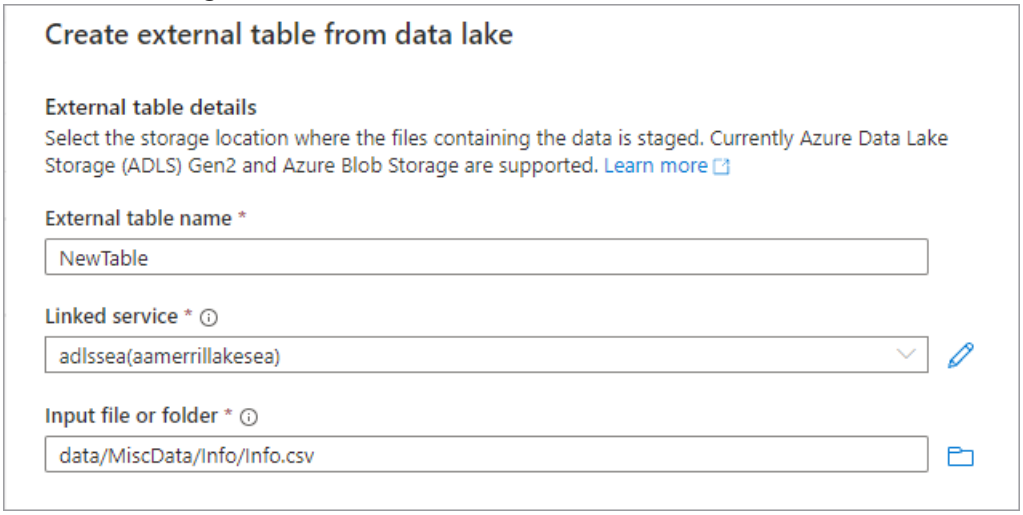
4. The database designer has **Properties** on the right that need to be configured.

   - **Name** - Give your database a name. Names cannot be edited after the database is published, so make sure the name you choose is correct.
   - **Description** - Giving your database a description is optional, but it allows users to understand the purpose of the database.
   - **Storage settings for database** - This section contains the default storage information for tables in the database. This default is applied to each table in the database unless it's overridden on the table itself.
   - **Linked service** - It's the default linked service used to store your data in Azure Data Lake Storage. The default linked service associated with the Synapse workspace will be shown, but you can change the **Linked Service** to any ADLS storage account you like.

- **Input folder** - Used to set the default container and folder path within that linked service using the file browser.
- **Data format** - Lake databases in Azure Synapse support parquet and delimited text as the storage formats for data.

> **NOTE**
> You can always override the default storage settings on a table by table basis, and the default remains customizable. If you are not sure what to choose, you can revisit this later.

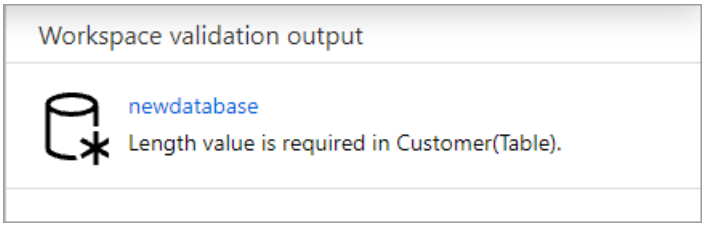
5. To add a table to the database, select the **+ Table** button.
   - **Custom** will add a new table to the canvas.
   - **From template** will open the gallery and let you select a database template to use when adding a new table. For more information, see Create lake database from database template.
   - **From data lake** lets you import a table schema using data already in your lake.
6. Select **Custom**. A new table will appear on the canvas called Table_1.
7. You can then customize Table_1, including the table name, description, storage settings, columns, and relationships. For more information, see Modify a lake database.
8. Add a new table from the data lake by selecting **+ Table** and then **From data lake**.
9. The **Create external table from data lake** pane will appear. Fill out the pane with the below details and select **Continue**.
   - **External table name** the name you want to give the table you're creating.
   - **Linked service** the linked service containing the Azure Data Lake Storage location where your data file lives.
   - **Input file or folder** use the file browser to navigate to and select a file on your lake you want to create a table using.



   - On the next screen, Azure Synapse will preview the file and detect the schema.
   - You'll land on the **New external table** page where you can update any settings related to the data format, and **Preview Data** to check if Synapse identified the file correctly.
   - When you're happy with the settings, select **Create**.
   - A new table with the name you selected will be added to the canvas, and the **Storage settings for table** section will show the file that you specified.
10. With the database customized, it's now time to publish it. If you're using Git integration with your Synapse workspace, you must commit your changes and merge them into the collaboration branch. Learn more about source control in Azure Synapse. If you're using Synapse Live mode, you can select

"publish".

- Your database will be validated for errors before it's published. Any errors found will be showing in the notifications tab with instructions on how to remedy the error.



- Publishing will create your database schema in the Azure Synapse Metastore. After publishing, the database and table objects will be visible to other Azure services and allow the metadata from your database to flow into apps like Power BI or Microsoft Purview.

11. You've now created an empty lake database in Azure Synapse, and added tables to it using the **Custom** and **From data lake** options.

Next steps

Continue to explore the capabilities of the database designer using the links below.

- Modify a lake database
- Learn more about lake databases
- Create a lake database from lake database template

How-to: Create a lake database from database templates

8/19/2022 • 5 minutes to read • Edit Online

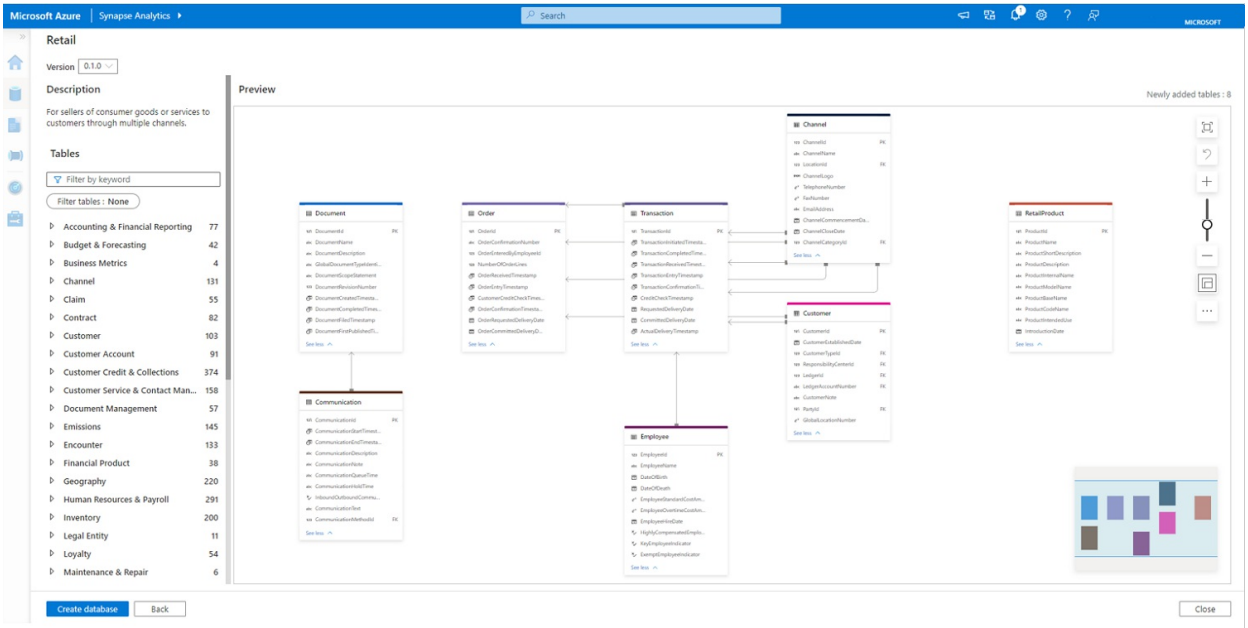
In this article, you'll learn how to use the Azure Synapse database templates to simplify the creation of a lake database. Using database templates provides your database with rich semantic information, allowing end users of the data to more easily understand what data is available and how to use it.

Prerequisites

- At least Synapse User role permissions are required for exploring a lake database template from Gallery.
- Synapse Administrator, or Synapse Contributor permissions are required on the Synapse workspace for creating a lake database.
- Storage Blob Data Contributor permissions are required on data lake when using the create table **From data lake** option.

Create lake database from database template

1. From your Azure Synapse Analytics workspace **Home** hub, select **Knowledge center** and then **Browse gallery**. You'll land on the **Lake database templates** tab.
2. **Lake database templates** category lists standardized database templates available for a specific industry.
   a. You can also visit the **Lake database templates** tab from **Data** hub, **+** Add new resource, **Browse gallery** menu.
3. Select the industry you're interested in (for example, **Retail**) and select **Continue** to navigate to the exploration of the data model.
4. You'll land on the database canvas and can explore the tables available in the template. By default, the canvas will show a selection of the most used tables in that template. The canvas has various tools to help you navigate the entity-relationship diagram.
   - **Zoom to fit** to fit all tables on the canvas in the viewing area
   - **Undo last action** to undo **one** recent change
   - **Increase zoom** to zoom in to the canvas
   - **Decrease zoom** to zoom out of the canvas
   - **Zoom slider** to control the zoom level
   - **Zoom preview** to provide a preview of the canvas
   - **Expand all/Collapse all** to view more or less columns within a table on the canvas
   - **Clear canvas** to clear-off all the tables on the canvas

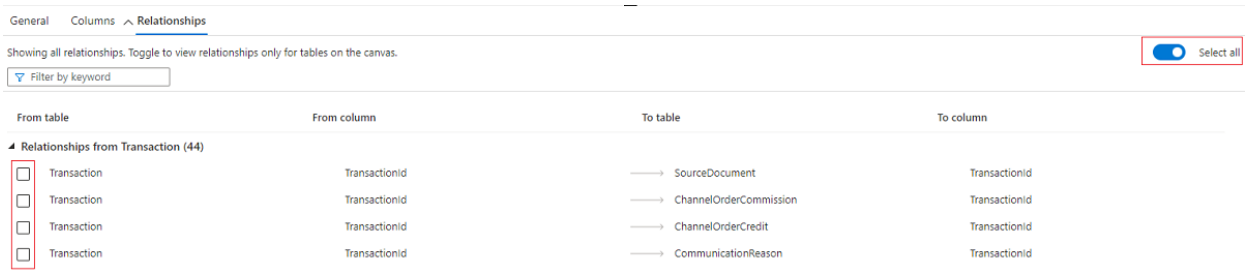


5. On the left, you'll see list of folders containing the items of the template you can add to the canvas. There are several controls to help.

   - **Search box** to search for tables based on a term. The term will be searched across the template tables, columns, and descriptions.
   - **Filter tables** to select whole sections of the model to explore
     - **Enterprise templates** select tables from each **Business area** to show a "best-of".
     - **Business area templates** select tables primarily from a single **Business area** but also include related tables from other **Business areas**.
   - **Business areas** are folders containing tables related to that business construct. For example, Budget & Forecasting contains tables related to managing budgets.
   - You can expand business area folders to view the tables, and select the checkbox to add them to the canvas.
   - Selected tables can be removed via the checkbox.
   - You can also click on the ellipses next to the business area folder and **Select All** or **UnSelect All** to add/remove all tables under that business area to the canvas.

6. You can select a table on the canvas. It opens the table properties pane with the tabs General, Columns, and Relationships.

   - The General tab has information on the table such as its name and description.
   - The Columns tab has the details about all the columns that make up the table such as column names and datatypes.
   - The Relationships tab lists the incoming and outgoing relationships of the table with other tables on the canvas.
     - Use the **Select all** toggle to view all the 'from' & 'to' relationships to that table.
     - Using the check boxes next to each relationship in the relationship tab, add the required table - relationship to the canvas.



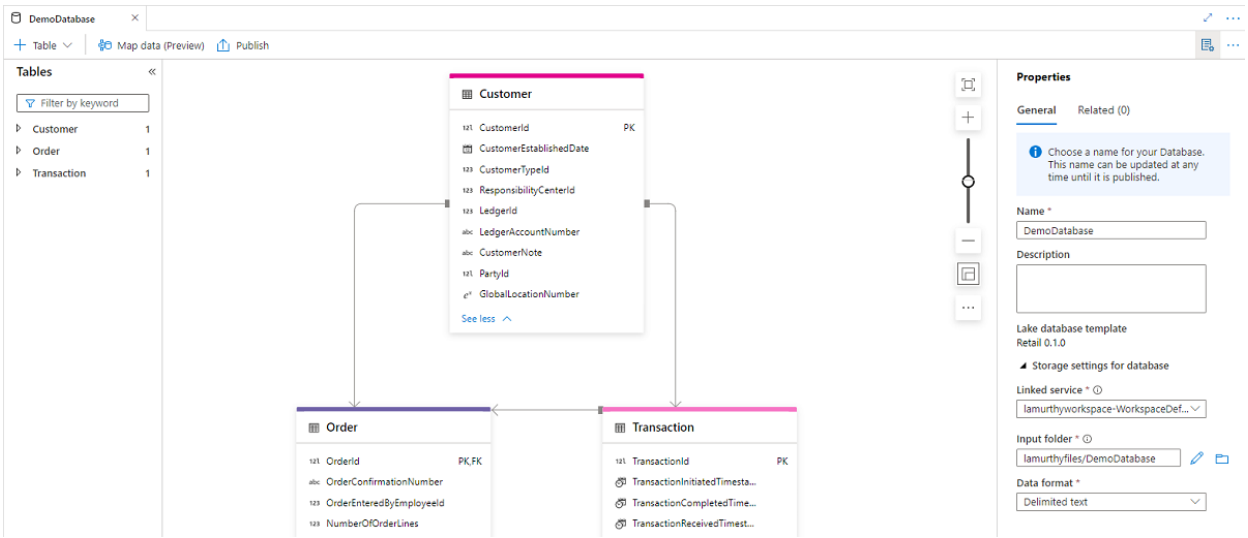
7. To quickly add tables that are related to the tables on canvas, select the ellipses to the right of the table name and then select **Add related tables**. All tables with existing relationships are added to the canvas. If this adds too many tables to the canvas, use the **Undo Last Action** to undo this change.

8. Once the canvas has all the tables that meet your requirements, select **Create database** to continue with creation of lake database. The new database will show up in the database designer and you can customize it per your business needs.

9. The database designer has more **Properties** on the right that need to be configured.

   - **Name** give your database a name. Names cannot be edited after the database is published, so make sure the name you choose is correct.
   - **Description** Giving your database a description is optional, but it allows users to understand the purpose of the database.
   - **Storage settings for database** is a section containing the default storage information for tables in the database. This default is applied to each table in the database unless it's overridden on the table itself.
   - **Linked service** is the default linked service used to store your data in Azure Data Lake Storage. The default linked service associated with the Synapse workspace will be shown, but you can change the **Linked Service** to any ADLS storage account you like.
   - **Input folder** used to set the default container and folder path within that linked service using the file browser or manually editing the path with the pencil icon.
   - **Data format** lake databases in Azure Synapse support parquet and delimited text as the storage formats for data.

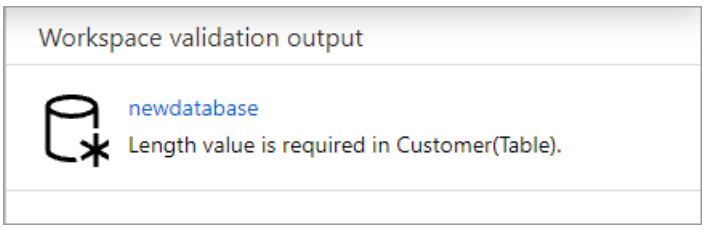
> **NOTE**
>
> You can always override the default storage settings on a table by table basis, and the default remains customizable. If you are not sure what to choose, you can revisit this later. If you are unsure of the folder hierarchy in the data lake, you can also specify wildcards to traverse the directory structure.



10. You can begin to customize tables, columns, and relationships inherited from the database template. You can also add custom tables, columns, relationships as needed in the database. For more information on modifying a lake database, see Modify a lake database.

11. With the database customized, it's now time to publish it. If you're using Git integration with your Synapse workspace, you must commit your changes and merge them into the collaboration branch. Learn more about source control in Azure Synapse. If you're using Synapse Live mode, you can select **Publish**.

- Your database will be validated for errors before it's published. Any errors found will be showing in the notifications tab with instructions on how to remedy the error.



- Publishing will create your database schema in the Azure Synapse Metastore. After publishing, the database and table objects will be visible to other Azure services and allow the metadata from your database to flow into apps like Power BI or Microsoft Purview.

12. You've now created a lake database using a lake database template in Azure Synapse.

Next steps

Continue to explore the capabilities of the database designer using the links below.

- Modify a lake database
- Learn more about lake databases

How-to: Modify a lake database

8/19/2022 • 10 minutes to read • Edit Online

In this article, you'll learn how to modify an existing lake database in Azure Synapse using the database designer. The database designer allows you to easily create and deploy a database without writing any code.

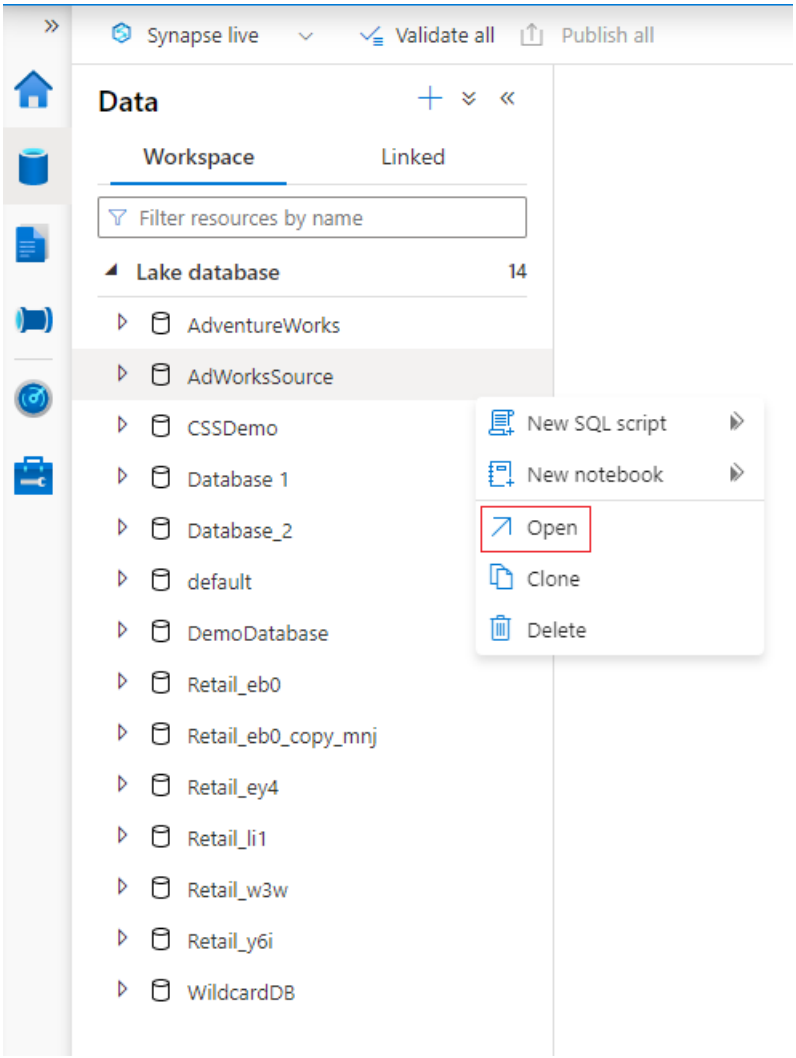
Prerequisites

- Synapse Administrator, or Synapse Contributor permissions are required on the Synapse workspace for creating a lake database.
- Storage Blob Data Contributor permissions are required on data lake when using the create table **From data lake** option.

Modify database properties

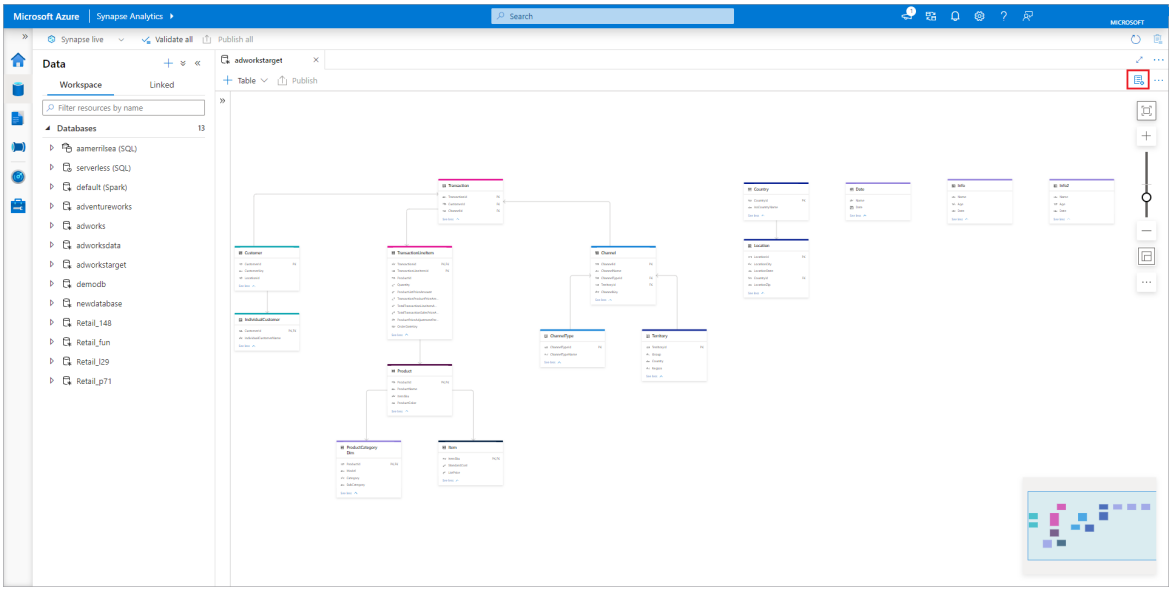
1. From your Azure Synapse Analytics workspace **Home** hub, select the **Data** tab on the left. The **Data** tab will open you'll see the list of databases that already exist in your workspace.

2. Hover over the **Databases** section and select the ellipsis **...** next to the database you want to modify, then choose **Open**.



3. The database designer tab will open with your selected database loaded on the canvas.

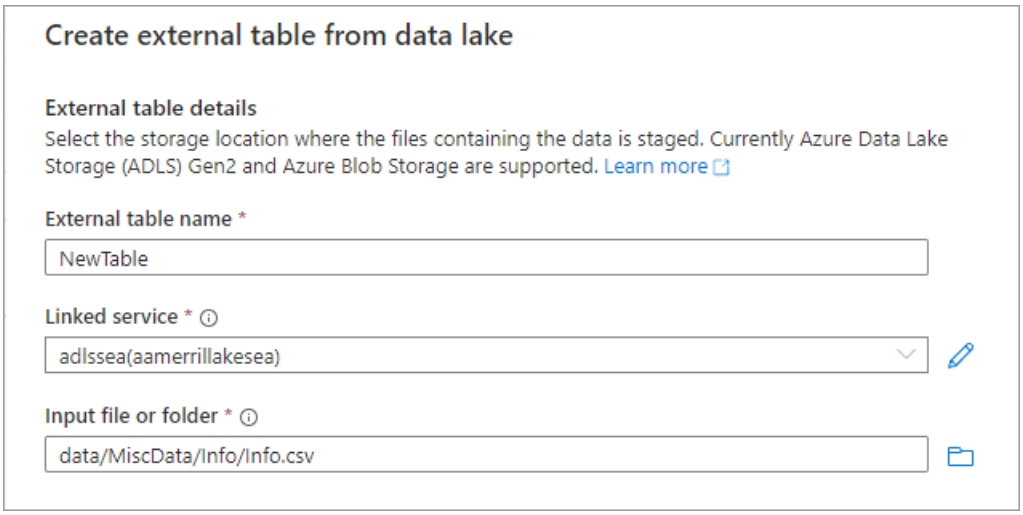
4. The database designer has **Properties** pane that can be opened by selecting the **Properties** icon in the upper right of the tab.



   - **Name** Names cannot be edited after the database is published, so make sure the name you choose is correct.
   - **Description** Giving your database a description is optional, but it allows users to understand the purpose of the database.
   - **Storage settings for database** is a section containing the default storage information for tables in the database. The default settings are applied to each table in the database unless it's overridden on the table itself.
   - **Linked service** is the default linked service used to store your data in Azure Data Lake Storage. The default linked service associated with the Synapse workspace will be shown, but you can change the **Linked Service** to any ADLS storage account you like.
   - **Input folder** used to set the default container and folder path within that linked service using the file browser or manually editing the path with the pencil icon.
   - **Data format** lake databases in Azure Synapse support parquet and delimited text as the storage formats for data.
5. To add a table to the database, select the **+ Table** button.

   - **Custom** will add a new table to the canvas.
   - **From template** will open the gallery and let you select a database template to use when adding a new table. For more information, see Create lake database from database template.
   - **From data lake** lets you import a table schema using data already in your lake.
6. Select **Custom**. A new table will appear on the canvas called Table_1.
7. You can then customize Table_1, including the table name, description, storage settings, columns, and relationships. See the Customize tables within a database section below.
8. Add a new table from the data lake by selecting **+ Table** and then **From data lake**.
9. The **Create external table from data lake** pane will appear. Fill out the pane with the below details and select **Continue**.

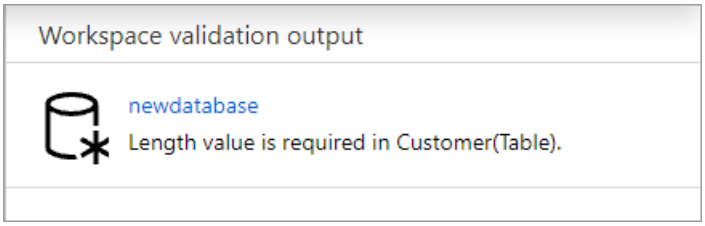
   - **External table name** the name you want to give the table you're creating.
   - **Linked service** the linked service containing the Azure Data Lake Storage location where your data file lives.
   - **Input file or folder** use the file browser to navigate to and select a file on your lake you want to create a table using.



- On the next screen, Azure Synapse will preview the file and detect the schema.
- You'll land on the **New external table** page where you can update any settings related to the data format, and **Preview Data** to check if Azure Synapse identified the file correctly.
- When you're happy with the settings, select **Create**.
- A new table with the name you selected will be added to the canvas, and the **Storage settings for table** section will show the file that you specified.

10. With the database customized, it's now time to publish it. If you're using Git integration with your Synapse workspace, you must commit your changes and merge them into the collaboration branch. Learn more about source control in Azure Synapse. If you're using Synapse Live mode, you can select "publish".

    - Your database will be validated for errors before it's published. Any errors found will be showing in the notifications tab with instructions on how to remedy the error.



    - Publishing will create your database schema in the Azure Synapse Metastore. After publishing, the database and table objects will be visible to other Azure services and allow the metadata from your database to flow into apps like Power BI or Microsoft Purview.

Customize tables within a database

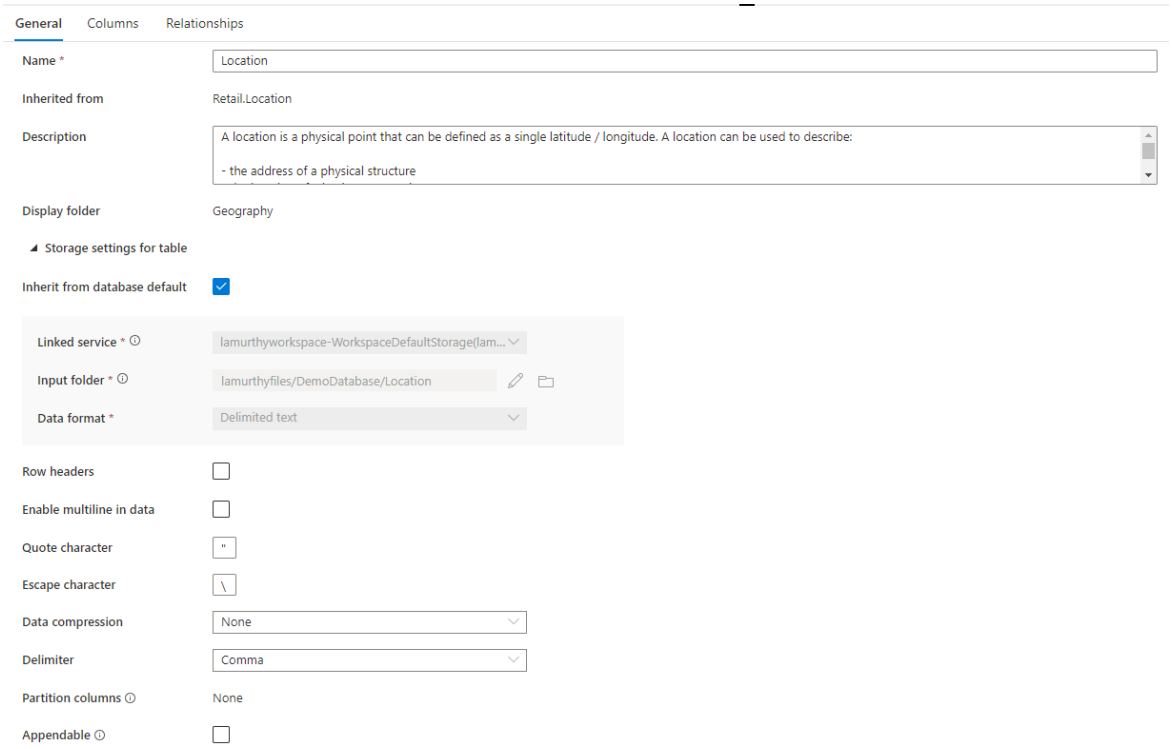
The database designer allows you to fully customize any of the tables in your database. When you select a table there are three tabs available, each containing settings related to the table's schema or metadata.

General

The **General** tab contains information specific to the table itself.

- **Name** the name of the table. The table name can be customized to any unique value within the database. Multiple tables with the same name aren't allowed.
- **Inherited from** (optional) this value will be present if the table was created from a database template. It cannot be edited and tells the user which template table it was derived from.
- **Description** a description of the table. If the table was created from a database template, this will contain a description of the concept represented by this table. This field is editable and can be changed to match the description that matches your business requirements.

- **Display folder** provides the name of the business area folder this table was grouped under as part of the database template. For custom tables, this value will be "Other".

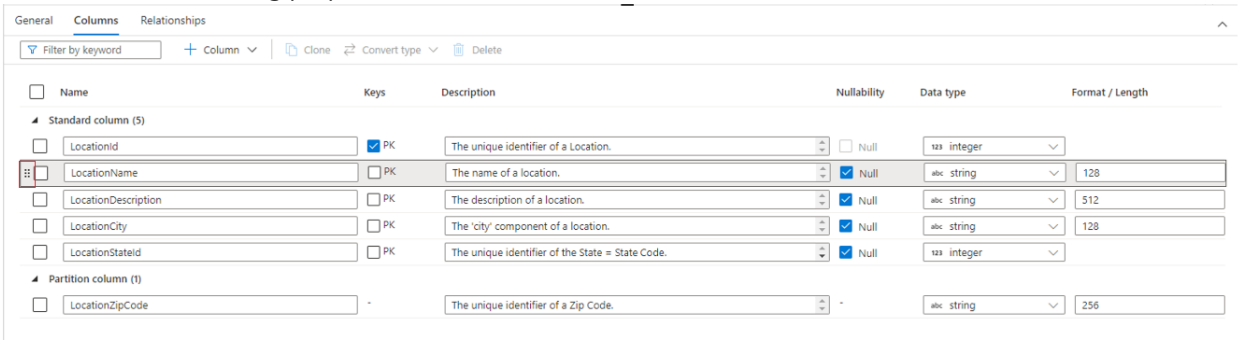


- In addition, there is a collapsible section called **Storage settings for table** that provides settings for the underlying storage information used by the table.
- **Inherit from database default** a checkbox that determines whether the storage settings below are inherited from the values set in the database **Properties** tab, or are set individually. If you want to customize the storage values, uncheck this box.
    - **Linked service** is the default linked service used to store your data in Azure Data Lake Storage. Change this to pick a different ADLS account.
    - **Input folder** the folder in ADLS where the data loaded to this table will live. You can either browse the folder location or edit it manually using the pencil icon.
    - **Data format** the data format of the data in the **Input folder** Lake databases in Azure Synapse support parquet and delimited text as the storage formats for data. If the data format doesn't match the data in the folder, queries to the table will fail.
- For a **Data format** of Delimited text, there are further settings:
    - **Row headers** check this box if the data has row headers.
    - **Enable multiline in data** check this box if the data has multiple lines in a string column.
    - **Quote Character** specify the custom quote character for a delimited text file.
    - **Escape Character** specify the custom escape character for a delimited text file.
    - **Data compression** the compression type used on the data.
    - **Delimiter** the field delimiter used in the data files. Supported values are: Comma (,), tab (\t), and pipe (|).
    - **Partition columns** the list of partition columns will be displayed here.
    - **Appendable** check this box if you are querying Dataverse data from SQL Serverless.
- For Parquet data, there's the following setting:
    - **Data compression** the compression type used on the data.

Columns

The **Columns** tab is where the columns for the table are listed and can be modified. On this tab are two lists of

columns: **Standard columns** and **Partition columns**. **Standard columns** are any column that stores data, is a primary key, and otherwise isn't used for the partitioning of the data. **Partition columns** store data as well, but are used to partition the underlying data into folders based on the values contained in the column. Each column has the following properties.

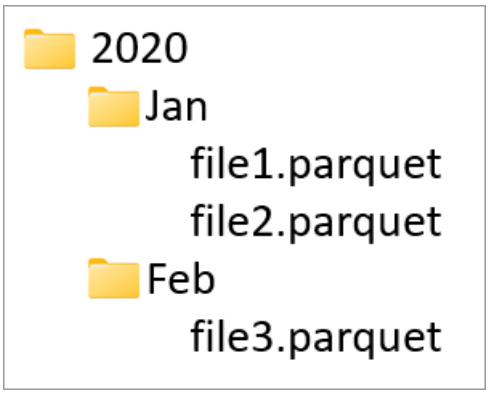


- **Name** the name of the column. Must be unique within the table.
- **Keys** indicates whether the column is a primary key (PK) and/or foreign key (FK) for the table. Not applicable to partition columns.
- **Description** a description of the column. If the column was created from a database template, the description of the concept represented by this column will be seen. This field is editable and can be changed to match the description that matches your business requirements.
- **Nullability** indicates whether there can be null values in this column. Not applicable to partition columns.
- **Data type** sets the data type for the Column based on the available list of Spark data types.
- **Format / Length** allows for customizing the format or maximum length of the column, depending on the data type. Date and timestamp data types have format dropdowns, and other types like string have a maximum length field. Not all data types have a value as some types are fixed length. At the top of the **Columns** tab is a command bar that can be used to interact with the columns.
- **Filter by keyword** filters the list of columns to items that match the keyword specified.
- **+ Column** lets you add a new column. There are three possible options.
  - **New column** creates a new custom standard column.
  - **From template** opens the exploration pane and lets you identify columns from a database template to include on your table. If your database wasn't created using a database template, this option will not appear.
  - **Partition column** adds a new custom partition column.
- **Clone** duplicates the selected column. Cloned columns are always of the same type as the selected column.
- **Convert type** is used to change the selected **standard column** to a **partition column** and the other way around. This option will be grayed out if you have selected multiple columns of different types or the selected column is ineligible to be converted because of a **PK** or **Nullability** flag set on the column.
- **Delete** deletes the selected columns from the table. This action is irreversible.

You can also re-arrange the order of the columns by drag and drop using the double vertical ellipses that show up on the left of the column name when you hover over or click on the column as shown in the image above.

Partition Columns

Partition columns are used to partition the physical data in your database based on values in those columns. Partition columns allow an easy way to distribute data on disk into more performant chunks. Partition columns in Azure Synapse are always at the end of the table schema. In addition, they are used from top to bottom when creating the partition folders. For example, if your partition columns were Year and Month, you would end up with a structure in ADLS like this:

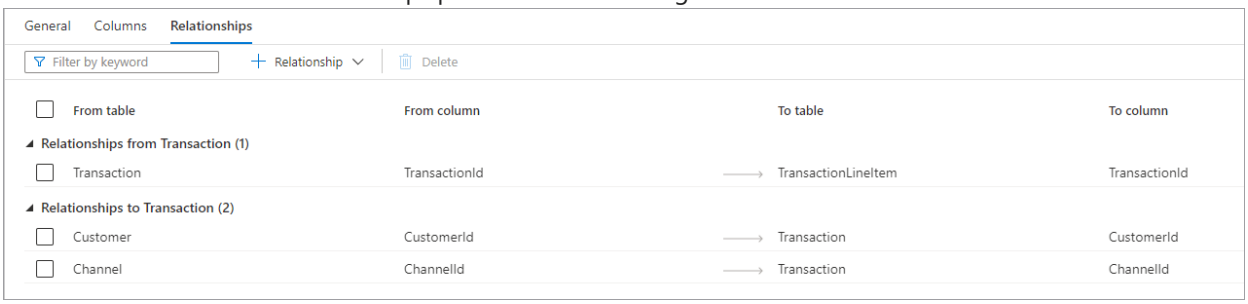


Where file1 and file2 contained all the rows where the values of Year and Month were 2020 and Jan respectively. As more partition columns are added to a table, the more files are added to this hierarchy, making the overall file size of the partitions smaller.

Azure Synapse doesn't enforce or create this hierarchy by adding partition columns to a table. Data must be loaded into the table using either Synapse Pipelines or a Spark notebook in order for the partition structure to be created.

Relationships

The relationships tab lets you specify relationships between tables in the database. Relationships in the database designer are informational, and don't enforce any constraints on the underlying data. They're read by other Microsoft applications can be used to accelerate transformations or provide business users insight into how tables are connected. The relationships pane has the following info.



- **Relationships from (Table)** is when one or more tables have foreign keys connected to this table. This is sometimes called a parent relationship.
- **Relationships to (Table)** is when a table that has foreign key and is connected to other table. This is sometimes called a child relationship.
- Both relationship types have the following properties.
  - **From table** the parent table in the relationship, or the "one" side.
  - **From column** the column in the parent table the relationship is based on.
  - **To table** the child table in the relationship, or the "many" side.
  - **To column** the column in the child table the relationship is based on. At the top of the **Relationships** tab, is the command bar that can be used to interact with the relationships
- **Filter by keyword** filters the list of columns to items that match the keyword specified.
- **+ Relationship** lets you add a new relationship. There are two options.
  - **From table** creates a new relationship from the table you're working on to a different table.
  - **To table** creates a new relationship from a different table to the one you're working on.
  - **From template** opens the exploration pane and lets you pick from relationships in the database template to include in your database. If your database wasn't created using a database template, this option won't appear.

Next steps

Continue to explore the capabilities of the database designer using the links below.

- Create an empty lake database
- Clone a lake database
- Create a lake database from lake database template

How-to: Clone a lake database

8/19/2022 • 2 minutes to read • Edit Online

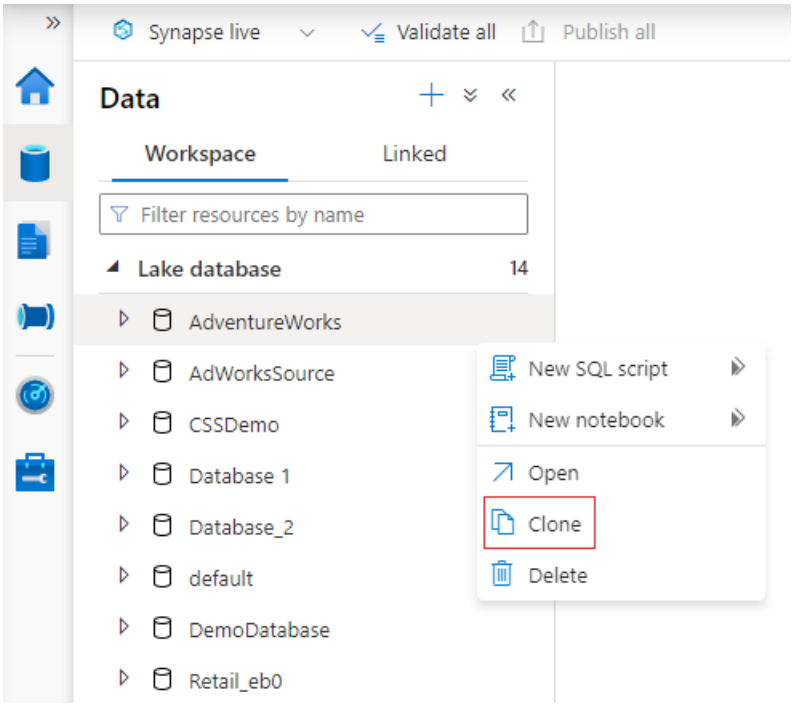
In this article, you'll learn how to clone an existing lake database in Azure Synapse using the database designer. The database designer allows you to easily create and deploy a database without writing any code.

Prerequisites

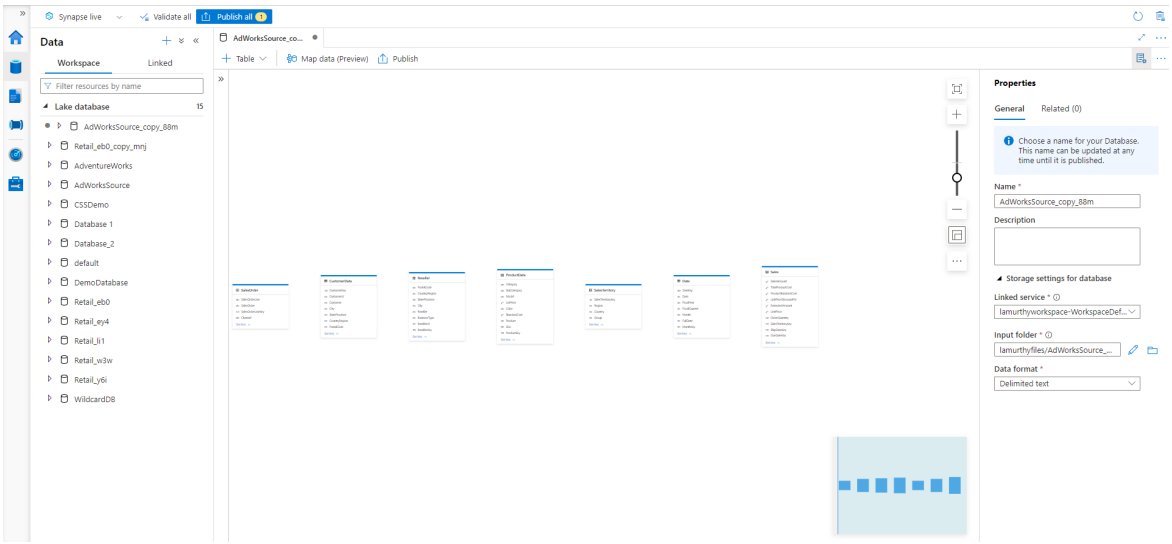
- Synapse Administrator, or Synapse Contributor permissions are required on the Synapse workspace for creating a lake database.

Clone database

1. From your Azure Synapse Analytics workspace **Home** hub, select the **Data** tab on the left. The **Data** tab will open. You'll see the list of databases that already exist in your workspace.
2. Hover over the **Databases** section and select the ellipsis **...** next to the database you want to clone, then choose **Clone**.



3. The database designer tab will open with a copy of the selected database loaded on the canvas.
4. The database designer opens the **Properties** pane by default. Update the details for this cloned lake database.



- **Name** Provide a new name for this database. Note that this name cannot be changed once the lake database is published.
- **Description** Give your database a description (optional).
- **Storage settings for database** provide the default storage information for tables in the database. The default settings are applied to each table in the database unless it's overridden on the table itself.
- **Linked service** provide the default linked service used to store your data in Azure Data Lake Storage. The default linked service associated with the Synapse workspace will be shown, but you can change the **Linked Service** to any ADLS storage account you like.
- **Input folder** set the default container and folder path within that linked service using the file browser or manually editing the path with the pencil icon.
- **Data format** select the data format. Lake databases in Azure Synapse support parquet and delimited text as the storage formats for data.

5. You can add additional tables to this cloned database by selecting the **+ Table** button. For more information, see Modify lake database.

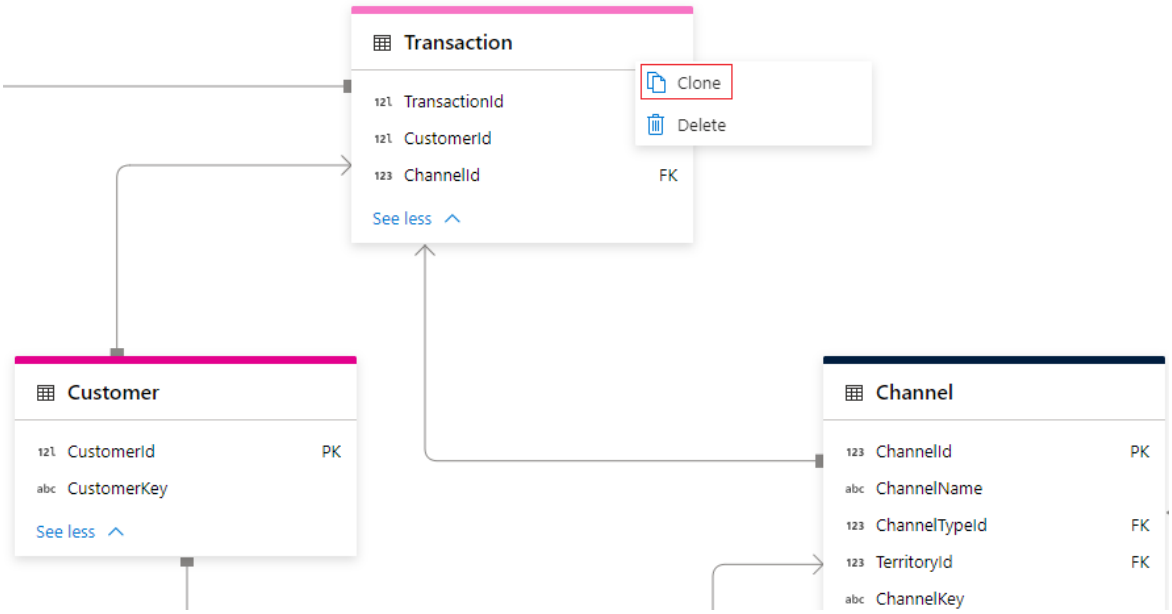
6. Once you have completed all the changes to your cloned database, it's now time to publish it. If you're using Git integration with your Synapse workspace, you must commit your changes and merge them into the collaboration branch. Learn more about source control in Azure Synapse. If you're using Synapse Live mode, you can select "publish".

Clone tables within a lake database

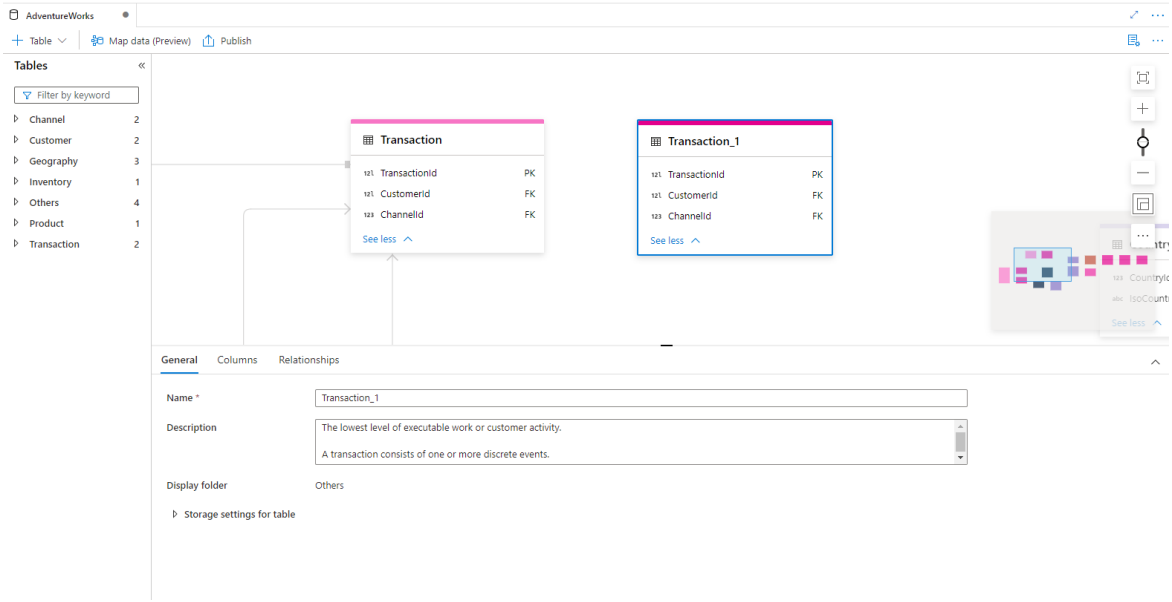
The database designer allows you to clone any of the tables in your database.

1. Select a table within your lake database and click on the ellipsis ... next to the table name.

2. Click **Clone**.



3. The database designer will create a copy of the table and opens the **General** tab by default.



4. Update the **Name**, **Description** and **Storage settings for table** as needed. For more information, see Modify lake database.

5. Similarly update the **Columns** tab and **Relationships** tab as needed.

6. Once you have completed all the changes to your cloned database, it's now time to publish it. If you're using Git integration with your Synapse workspace, you must commit your changes and merge them into the collaboration branch. Learn more about source control in Azure Synapse. If you're using Synapse Live mode, you can select "publish".

Next steps

Continue to explore the capabilities of the database designer using the links below.

- Create an empty lake database
- Learn more about lake databases
- Create a lake database from lake database template

Azure Synapse SQL architecture

8/19/2022 • 6 minutes to read • Edit Online

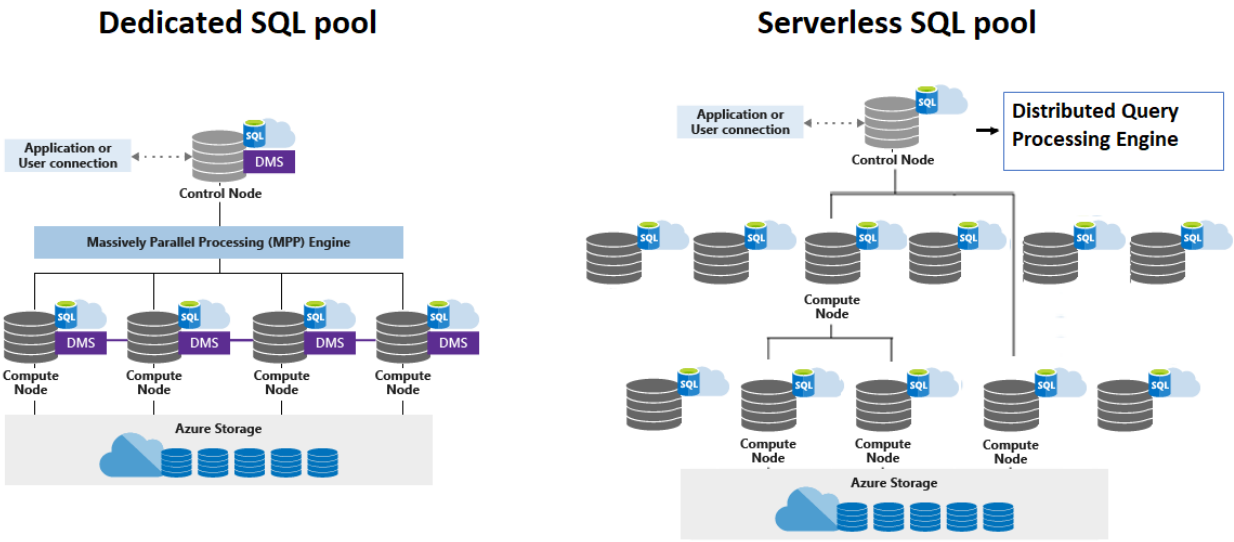
This article describes the architecture components of Synapse SQL.

Synapse SQL architecture components

Synapse SQL leverages a scale out architecture to distribute computational processing of data across multiple nodes. Compute is separate from storage, which enables you to scale compute independently of the data in your system.

For dedicated SQL pool, the unit of scale is an abstraction of compute power that is known as a data warehouse unit.

For serverless SQL pool, being serverless, scaling is done automatically to accommodate query resource requirements. As topology changes over time by adding, removing nodes or failovers, it adapts to changes and makes sure your query has enough resources and finishes successfully. For example, the image below shows serverless SQL pool utilizing 4 compute nodes to execute a query.



Synapse SQL uses a node-based architecture. Applications connect and issue T-SQL commands to a Control node, which is the single point of entry for Synapse SQL.

The Azure Synapse SQL Control node utilizes a distributed query engine to optimize queries for parallel processing, and then passes operations to Compute nodes to do their work in parallel.

The serverless SQL pool Control node utilizes Distributed Query Processing (DQP) engine to optimize and orchestrate distributed execution of user query by splitting it into smaller queries that will be executed on Compute nodes. Each small query is called task and represents distributed execution unit. It reads file(s) from storage, joins results from other tasks, groups, or orders data retrieved from other tasks.

The Compute nodes store all user data in Azure Storage and run the parallel queries. The Data Movement Service (DMS) is a system-level internal service that moves data across the nodes as necessary to run queries in parallel and return accurate results.

With decoupled storage and compute, when using Synapse SQL one can benefit from independent sizing of compute power irrespective of your storage needs. For serverless SQL pool scaling is done automatically, while for dedicated SQL pool one can:

- Grow or shrink compute power, within a dedicated SQL pool, without moving data.
- Pause compute capacity while leaving data intact, so you only pay for storage.
- Resume compute capacity during operational hours.

Azure Storage

Synapse SQL leverages Azure Storage to keep your user data safe. Since your data is stored and managed by Azure Storage, there is a separate charge for your storage consumption.

Serverless SQL pool allows you to query your data lake files, while dedicated SQL pool allows you to query and ingest data from your data lake files. When data is ingested into dedicated SQL pool, the data is sharded into **distributions** to optimize the performance of the system. You can choose which sharding pattern to use to distribute the data when you define the table. These sharding patterns are supported:

- Hash
- Round Robin
- Replicate

Control node

The Control node is the brain of the architecture. It is the front end that interacts with all applications and connections.

In Synapse SQL, the distributed query engine runs on the Control node to optimize and coordinate parallel queries. When you submit a T-SQL query to dedicated SQL pool, the Control node transforms it into queries that run against each distribution in parallel.

In serverless SQL pool, the DQP engine runs on Control node to optimize and coordinate distributed execution of user query by splitting it into smaller queries that will be executed on Compute nodes. It also assigns sets of files to be processed by each node.

Compute nodes

The Compute nodes provide the computational power.

In dedicated SQL pool, distributions map to Compute nodes for processing. As you pay for more compute resources, pool remaps the distributions to the available Compute nodes. The number of compute nodes ranges from 1 to 60, and is determined by the service level for the dedicated SQL pool. Each Compute node has a node ID that is visible in system views. You can see the Compute node ID by looking for the node_id column in system views whose names begin with sys.pdw_nodes. For a list of these system views, see Synapse SQL system views.

In serverless SQL pool, each Compute node is assigned task and set of files to execute task on. Task is distributed query execution unit, which is actually part of query user submitted. Automatic scaling is in effect to make sure enough Compute nodes are utilized to execute user query.

Data Movement Service

Data Movement Service (DMS) is the data transport technology in dedicated SQL pool that coordinates data movement between the Compute nodes. Some queries require data movement to ensure the parallel queries return accurate results. When data movement is required, DMS ensures the right data gets to the right location.

Distributions

A distribution is the basic unit of storage and processing for parallel queries that run on distributed data in

dedicated SQL pool. When dedicated SQL pool runs a query, the work is divided into 60 smaller queries that run in parallel.

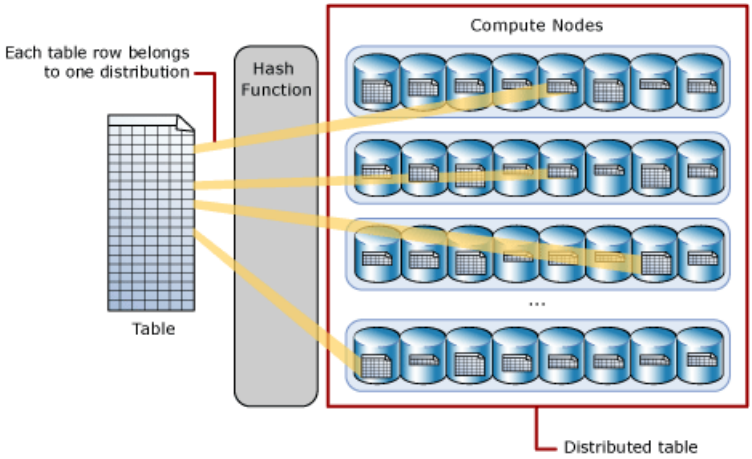
Each of the 60 smaller queries runs on one of the data distributions. Each Compute node manages one or more of the 60 distributions. A dedicated SQL pool with maximum compute resources has one distribution per Compute node. A dedicated SQL pool with minimum compute resources has all the distributions on one compute node.

Hash-distributed tables

A hash distributed table can deliver the highest query performance for joins and aggregations on large tables.

To shard data into a hash-distributed table, dedicated SQL pool uses a hash function to deterministically assign each row to one distribution. In the table definition, one of the columns is designated as the distribution column. The hash function uses the values in the distribution column to assign each row to a distribution.

The following diagram illustrates how a full (non-distributed table) gets stored as a hash-distributed table.



- Each row belongs to one distribution.
- A deterministic hash algorithm assigns each row to one distribution.
- The number of table rows per distribution varies as shown by the different sizes of tables.

There are performance considerations for the selection of a distribution column, such as distinctness, data skew, and the types of queries that run on the system.

Round-robin distributed tables

A round-robin table is the simplest table to create and delivers fast performance when used as a staging table for loads.

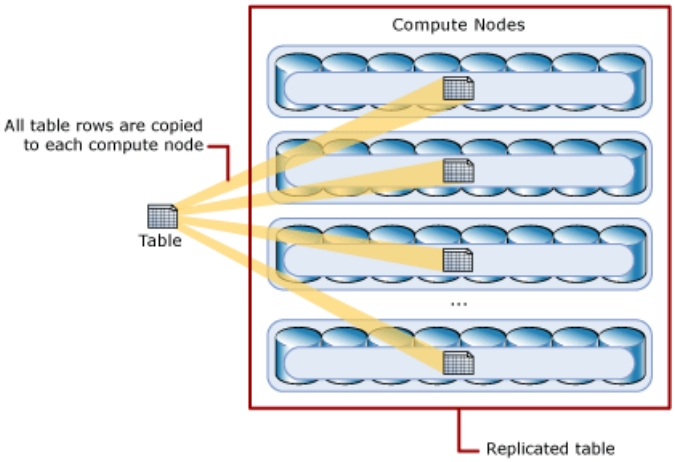
A round-robin distributed table distributes data evenly across the table but without any further optimization. A distribution is first chosen at random and then buffers of rows are assigned to distributions sequentially. It is quick to load data into a round-robin table, but query performance can often be better with hash distributed tables. Joins on round-robin tables require reshuffling data, which takes additional time.

Replicated tables

A replicated table provides the fastest query performance for small tables.

A table that is replicated caches a full copy of the table on each compute node. So, replicating a table removes the need to transfer data among compute nodes before a join or aggregation. Replicated tables are best utilized with small tables. Extra storage is required and there is additional overhead that is incurred when writing data, which make large tables impractical.

The diagram below shows a replicated table that is cached on the first distribution on each compute node.



Next steps

Now that you know a bit about Synapse SQL, learn how to quickly create a dedicated SQL pool and load sample data. Or start using serverless SQL pool. If you are new to Azure, you may find the Azure glossary helpful as you encounter new terminology.

Serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

Every Azure Synapse Analytics workspace comes with serverless SQL pool endpoints that you can use to query data in the Azure Data Lake (Parquet, Delta Lake, delimited text formats), Cosmos DB, or Dataverse.

Serverless SQL pool is a query service over the data in your data lake. It enables you to access your data through the following functionalities:

- A familiar T-SQL syntax to query data in place without the need to copy or load data into a specialized store.
- Integrated connectivity via the T-SQL interface that offers a wide range of business intelligence and ad-hoc querying tools, including the most popular drivers.

Serverless SQL pool is a distributed data processing system, built for large-scale data and computational functions. Serverless SQL pool enables you to analyze your Big Data in seconds to minutes, depending on the workload. Thanks to built-in query execution fault-tolerance, the system provides high reliability and success rates even for long-running queries involving large data sets.

Serverless SQL pool is serverless, hence there's no infrastructure to setup or clusters to maintain. A default endpoint for this service is provided within every Azure Synapse workspace, so you can start querying data as soon as the workspace is created.

There is no charge for resources reserved, you are only being charged for the data processed by queries you run, hence this model is a true pay-per-use model.

If you use Apache Spark for Azure Synapse in your data pipeline, for data preparation, cleansing or enrichment, you can query external Spark tables you've created in the process, directly from serverless SQL pool. Use Private Link to bring your serverless SQL pool endpoint into your managed workspace VNet.

Serverless SQL pool benefits

If you need to explore data in the data lake, gain insights from it or optimize your existing data transformation pipeline, you can benefit from using serverless SQL pool. It is suitable for the following scenarios:

- Basic discovery and exploration - Quickly reason about the data in various formats (Parquet, CSV, JSON) in your data lake, so you can plan how to extract insights from it.
- Logical data warehouse – Provide a relational abstraction on top of raw or disparate data without relocating and transforming data, allowing always up-to-date view of your data. Learn more about creating logical data warehouse.
- Data transformation - Simple, scalable, and performant way to transform data in the lake using T-SQL, so it can be fed to BI and other tools, or loaded into a relational data store (Synapse SQL databases, Azure SQL Database, etc.).

Different professional roles can benefit from serverless SQL pool:

- Data Engineers can explore the lake, transform and prepare data using this service, and simplify their data transformation pipelines. For more information, check this tutorial.
- Data Scientists can quickly reason about the contents and structure of the data in the lake, thanks to features such as OPENROWSET and automatic schema inference.
- Data Analysts can explore data and Spark external tables created by Data Scientists or Data Engineers using familiar T-SQL language or their favorite tools, which can connect to serverless SQL pool.
- BI Professionals can quickly create Power BI reports on top of data in the lake and Spark tables.

How to start using serverless SQL pool

Serverless SQL pool endpoint is provided within every Azure Synapse workspace. You can create a workspace and start querying data instantly using tools you are familiar with.

Make sure that you are applying the best practices to get the best performance.

Client tools

Serverless SQL pool enables existing SQL ad-hoc querying and business intelligence tools to tap into the data lake. As it provides familiar T-SQL syntax, any tool capable to establish TDS connection to SQL offerings can connect to and query Synapse SQL. You can connect with Azure Data Studio and run ad-hoc queries or connect with Power BI to gain insights in a matter of minutes.

T-SQL support

Serverless SQL pool offers T-SQL querying surface area, which is slightly enhanced/extended in some aspects to accommodate for experiences around querying semi-structured and unstructured data. Furthermore, some aspects of the T-SQL language aren't supported due to the design of serverless SQL pool, as an example, DML functionality is currently not supported.

- Workload can be organized using familiar concepts:
- Databases - serverless SQL pool endpoint can have multiple databases.
- Schemas - Within a database, there can be one or many object ownership groups called schemas.
- Views, stored procedures, inline table value functions
- External resources – data sources, file formats, and tables

Security can be enforced using:

- Logins and users
- Credentials to control access to storage accounts
- Grant, deny, and revoke permissions per object level
- Azure Active Directory integration

Supported T-SQL:

- Full SELECT surface area is supported, including a majority of SQL functions
- CETAS - CREATE EXTERNAL TABLE AS SELECT
- DDL statements related to views and security only

Serverless SQL pool has no local storage, only metadata objects are stored in databases. Therefore, T-SQL related to the following concepts isn't supported:

- Tables
- Triggers
- Materialized views
- DDL statements other than ones related to views and security
- DML statements

> **NOTE**
>
> Serverless SQL pool queries have a timeout. For more information on query timeout that may affect your workload, see serverless SQL pool system constraints. Currently you can't change the timeout.

Extensions

In order to enable smooth experience for in place querying of data residing in files in data lake, serverless SQL pool extends the existing OPENROWSET function by adding following capabilities:

Query multiple files or folders

Query PARQUET file format

Query DELTA format

Various delimited text formats (with custom field terminator, row terminator, escape char)

Cosmos DB analytical store

Read a chosen subset of columns

Schema inference

filename function

filepath function

Work with complex types and nested or repeated data structures

Security

Serverless SQL pool offers mechanisms to secure access to your data.

Azure Active Directory integration and multi-factor authentication

Serverless SQL pool enables you to centrally manage identities of database user and other Microsoft services with Azure Active Directory integration. This capability simplifies permission management and enhances security. Azure Active Directory (Azure AD) supports multi-factor authentication (MFA) to increase data and application security while supporting a single sign-on process.

Authentication

Serverless SQL pool authentication refers to how users prove their identity when connecting to the endpoint. Two types of authentication are supported:

- SQL Authentication

  This authentication method uses a username and password.

- Azure Active Directory Authentication:

  This authentication method uses identities managed by Azure Active Directory. For Azure AD users, multi-factor authentication can be enabled. Use Active Directory authentication (integrated security) whenever possible.

Authorization

Authorization refers to what a user can do within a serverless SQL pool database, and is controlled by your user account's database role memberships and object-level permissions.

If SQL Authentication is used, the SQL user exists only in serverless SQL pool and permissions are scoped to the objects in serverless SQL pool. Access to securable objects in other services (such as Azure Storage) can't be granted to SQL user directly since it only exists in scope of serverless SQL pool. The SQL user needs to use one of the supported authorization types to access the files.

If Azure AD authentication is used, a user can sign in to serverless SQL pool and other services, like Azure Storage, and can grant permissions to the Azure AD user.

Access to storage accounts

A user that is logged into the serverless SQL pool service must be authorized to access and query the files in Azure Storage. serverless SQL pool supports the following authorization types:

- **Shared access signature (SAS)** provides delegated access to resources in storage account. With a SAS, you can grant clients access to resources in storage account, without sharing account keys. A SAS gives you granular control over the type of access you grant to clients who have the SAS: validity interval, granted permissions, acceptable IP address range, acceptable protocol (https/http).
- **User Identity** (also known as "pass-through") is an authorization type where the identity of the Azure AD user that logged into serverless SQL pool is used to authorize access to the data. Before accessing the data, Azure Storage administrator must grant permissions to Azure AD user for accessing the data. This authorization type uses the Azure AD user that logged into serverless SQL pool, therefore it's not supported for SQL user types.
- **Workspace Identity** is an authorization type where the identity of the Synapse workspace is used to authorize access to the data. Before accessing the data, Azure Storage administrator must grant permissions to workspace identity for accessing the data.

Access to Cosmos DB

You need to create server-level or database-scoped credential with the Cosmos DB account read-only key to access Cosmos DB analytical store.

Next steps

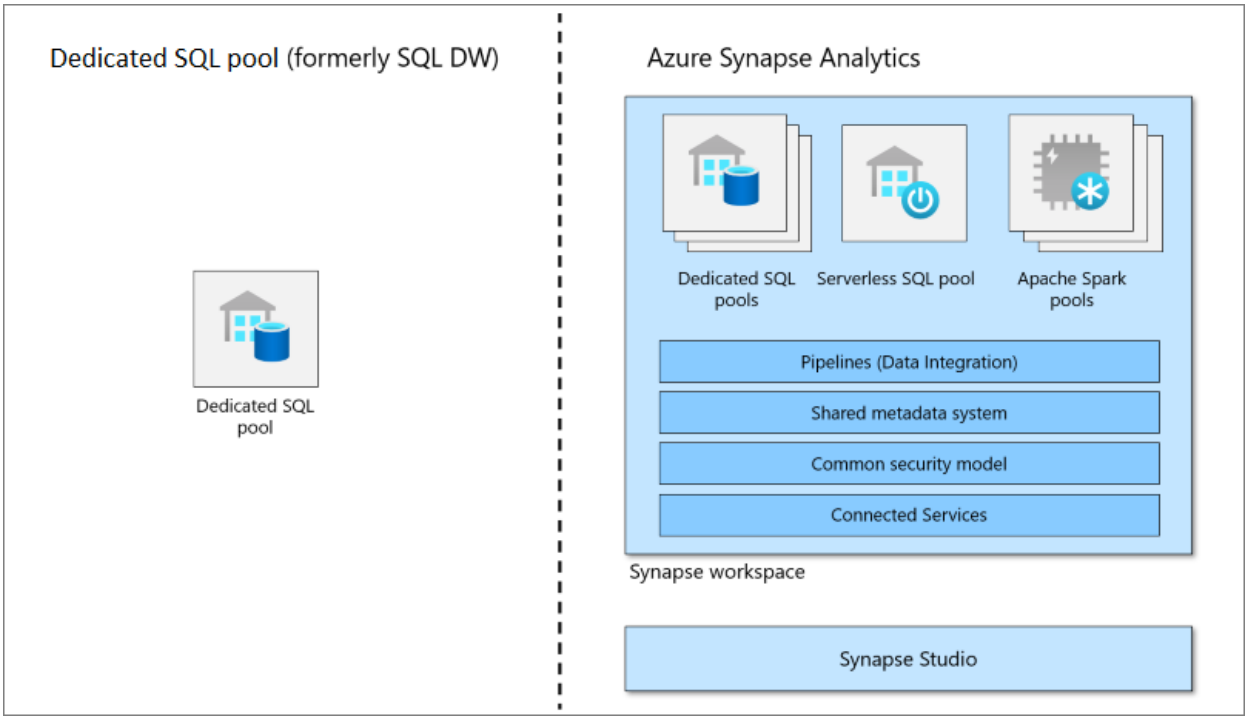
Additional information on endpoint connection and querying files can be found in the following articles:

- Connect to your endpoint
- Query your files

What is dedicated SQL pool (formerly SQL DW) in Azure Synapse Analytics?

8/19/2022 • 2 minutes to read • Edit Online

Azure Synapse Analytics is an analytics service that brings together enterprise data warehousing and Big Data analytics. Dedicated SQL pool (formerly SQL DW) refers to the enterprise data warehousing features that are available in Azure Synapse Analytics.



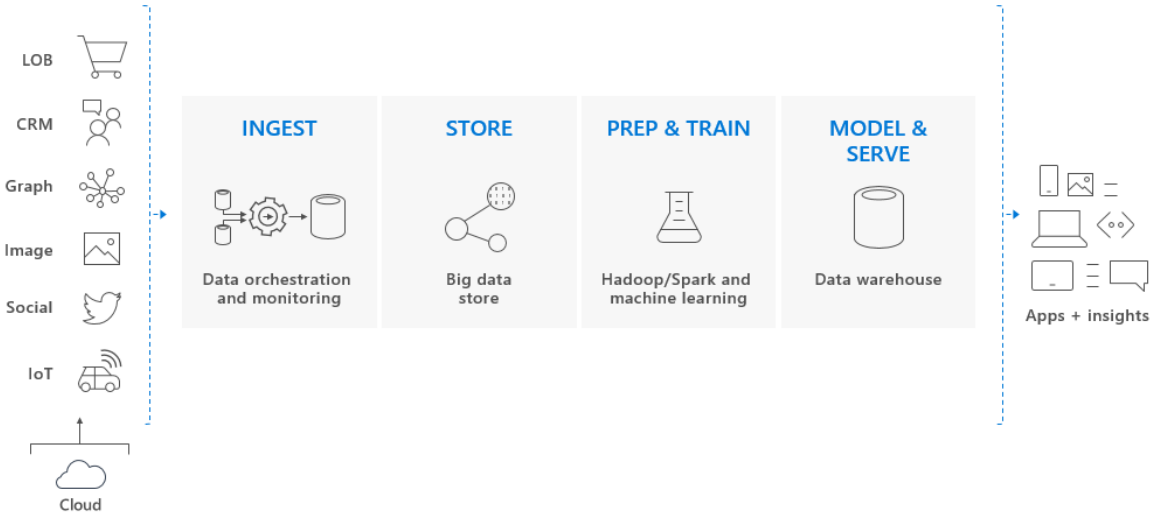
Dedicated SQL pool (formerly SQL DW) represents a collection of analytic resources that are provisioned when using Synapse SQL. The size of a dedicated SQL pool (formerly SQL DW) is determined by Data Warehousing Units (DWU).

Once your dedicated SQL pool is created, you can import big data with simple PolyBase T-SQL queries, and then use the power of the distributed query engine to run high-performance analytics. As you integrate and analyze the data, dedicated SQL pool (formerly SQL DW) will become the single version of truth your business can count on for faster and more robust insights.

> **NOTE**
> Not all features of the dedicated SQL pool in Azure Synapse workspaces apply to dedicated SQL pool (formerly SQL DW), and vice versa. To enable workspace features for an existing dedicated SQL pool (formerly SQL DW) refer to How to enable a workspace for your dedicated SQL pool (formerly SQL DW). Explore the Azure Synapse Analytics documentation and Get Started with Azure Synapse.

Key component of a big data solution

Data warehousing is a key component of a cloud-based, end-to-end big data solution.



In a cloud data solution, data is ingested into big data stores from a variety of sources. Once in a big data store, Hadoop, Spark, and machine learning algorithms prepare and train the data. When the data is ready for complex analysis, dedicated SQL pool uses PolyBase to query the big data stores. PolyBase uses standard T-SQL queries to bring the data into dedicated SQL pool (formerly SQL DW) tables.

Dedicated SQL pool (formerly SQL DW) stores data in relational tables with columnar storage. This format significantly reduces the data storage costs, and improves query performance. Once data is stored, you can run analytics at massive scale. Compared to traditional database systems, analysis queries finish in seconds instead of minutes, or hours instead of days.

The analysis results can go to worldwide reporting databases or applications. Business analysts can then gain insights to make well-informed business decisions.

Next steps

- Explore Azure Synapse architecture
- Quickly create a dedicated SQL pool
- Load sample data.
- Explore Videos
- Get Started with Azure Synapse

Or look at some of these other Azure Synapse resources.

- Search Blogs
- Submit a Feature requests
- Create a support ticket
- Search Microsoft Q&A question page
- Search Stack Overflow forum

Transact-SQL features supported in Azure Synapse SQL

8/19/2022 • 13 minutes to read • Edit Online

Azure Synapse SQL is a big data analytic service that enables you to query and analyze your data using the T-SQL language. You can use standard ANSI-compliant dialect of SQL language used on SQL Server and Azure SQL Database for data analysis.

Transact-SQL language is used in serverless SQL pool and dedicated model can reference different objects and has some differences in the set of supported features. In this page, you can find high-level Transact-SQL language differences between consumption models of Synapse SQL.

Database objects

Consumption models in Synapse SQL enable you to use different database objects. The comparison of supported object types is shown in the following table:

	DEDICATED	SERVERLESS
Tables	Yes	No, the in-database tables are not supported. Serverless SQL pool can query only external tables that reference data stored in Azure Data Lake storage or Dataverse.
Views	Yes. Views can use query language elements that are available in dedicated model.	Yes, you can create views over external tables, the queries with the OPENROWSET function, and other views. Views can use query language elements that are available in serverless model.
Schemas	Yes	Yes, schemas are supported. Use schemas to isolate different tenants and place their tables per schemas.
Temporary tables	Yes	Temporary tables might be used just to store some information from the system views, literals, or other temporary tables. UPDATE/DELETE on temp table is also supported. You can join temporary tables with the system views. You cannot select data from an external table to insert it into temporary table or join a temporary table with an external table - these operations will fail because external data and temporary tables cannot be mixed in the same query.

	DEDICATED	SERVERLESS
User defined procedures	Yes	Yes, stored procedures can be placed in any user databases (not `master` database). Procedures can just read external data and use query language elements that are available in serverless pool.
User defined functions	Yes	Yes, only inline table-valued functions are supported. Scalar user-defined functions are not supported.
Triggers	No	No, serverless SQL pools do not allow changing data, so the triggers cannot react on data changes.
External tables	Yes. See supported data formats.	Yes, external tables are available and can be used to read data from Azure Data Lake storage or Dataverse. See the supported data formats.
Caching queries	Yes, multiple forms (SSD-based caching, in-memory, resultset caching). In addition, Materialized View are supported.	No, only the file statistics are cached.
Result set caching	Yes	No, the query results are not cached. Only the file statistics are cached.
Materialized views	Yes	No, the Materialized views are not supported in the serverless SQL pools.
Table variables	No, use temporary tables	No, table variables are not supported.
Table distribution	Yes	No, table distributions are not supported.
Table indexes	Yes	No, indexes are not supported.
Table partitioning	Yes.	External tables do not support partitioning. You can partition files using Hive-partition folder structure and create partitioned tables in Spark. The Spark partitioning will be synchronized with the serverless pool. If you are not using Spark, you can partition your files in folder structure and can create partitioned views on folder partition structure, but the external tables cannot be created on partitioned folders.
Statistics	Yes	Yes, statistics are created on external files.

	DEDICATED	SERVERLESS
Workload management, resource classes, and concurrency control	Yes, see workload management, resource classes, and concurrency control.	No, you cannot manage the resources that are assigned to the queries. The serverless SQL pool automatically manages the resources.
Cost control	Yes, using scale-up and scale-down actions.	Yes, you can limit daily, weekly, or monthly usage of serverless pool using the Azure portal or T-SQL procedure.

Query language

Query languages used in Synapse SQL can have different supported features depending on consumption model. The following table outlines the most important query language differences in Transact-SQL dialects:

	DEDICATED	SERVERLESS
SELECT statement	Yes. `SELECT` statement is supported, but some Transact-SQL query clauses, such as FOR XML/FOR JSON, MATCH, OFFSET/FETCH are not supported.	Yes, `SELECT` statement is supported, but some Transact-SQL query clauses like FOR XML, MATCH, PREDICT, GROUPNG SETS, and the query hints are not supported.
INSERT statement	Yes	No. Upload new data to Data lake using Spark or other tools. Use Cosmos DB with the analytical storage for highly transactional workloads. You can use CETAS to create an external table and insert data.
UPDATE statement	Yes	No, update Parquet/CSV data using Spark and the changes will be automatically available in serverless pool. Use Cosmos DB with the analytical storage for highly transactional workloads.
DELETE statement	Yes	No, delete Parquet/CSV data using Spark and the changes will be automatically available in serverless pool. Use Cosmos DB with the analytical storage for highly transactional workloads.
MERGE statement	Yes (preview)	No, merge Parquet/CSV data using Spark and the changes will be automatically available in serverless pool.
CTAS statement	Yes	No, CREATE TABLE AS SELECT statement is not supported in the serverless SQL pool.
CETAS statement	Yes, you can perform initial load into an external table using CETAS.	Yes, you can perform initial load into an external table using CETAS. CETAS supports Parquet and CSV output formats.

	DEDICATED	SERVERLESS
Transactions	Yes	Yes, transactions are applicable only on the meta-data objects.
Labels	Yes	No, labels are not supported in serverless SQL pools.
Data load	Yes. Preferred utility is COPY statement, but the system supports both BULK load (BCP) and CETAS for data loading.	No, you cannot load data into the serverless SQL pool because data is stored on external storage. You can initially load data into an external table using CETAS statement.
Data export	Yes. Using CETAS.	Yes. You can export data from external storage (Azure data lake, Dataverse, Cosmos DB) into Azure data lake using CETAS.
Types	Yes, all Transact-SQL types except cursor, hierarchyid, ntext, text, and image, rowversion, Spatial Types, sql_variant, and xml	Yes, all Transact-SQL types are supported, except cursor, hierarchyid, ntext, text, and image, rowversion, Spatial Types, sql_variant, xml, and Table type. See how to map Parquet column types to SQL types here.
Cross-database queries	No	Yes, the cross-database queries and the 3-part-name references are supported including USE statement. The queries can reference the serverless SQL databases or the Lake databases in the same workspace. Cross-workspace queries are not supported.
Built-in/system functions (analysis)	Yes, all Transact-SQL Analytic, Conversion, Date and Time, Logical, Mathematical functions, except CHOOSE and PARSE	Yes, all Transact-SQL Analytic, Conversion, Date and Time, Logical, and Mathematical functions are supported.
Built-in/system functions (string)	Yes. All Transact-SQL String, JSON, and Collation functions, except STRING_ESCAPE and TRANSLATE	Yes. All Transact-SQL String, JSON, and Collation functions are supported.
Built-in/system functions (Cryptographic)	Some	`HASHBYTES` is the only supported cryptographic function in serverless SQL pools.
Built-in/system table-value functions	Yes, Transact-SQL Rowset functions, except OPENXML, OPENDATASOURCE, OPENQUERY, and OPENROWSET	Yes, all Transact-SQL Rowset functions are supported, except OPENXML, OPENDATASOURCE, and OPENQUERY.
Built-in/system aggregates	Transact-SQL built-in aggregates, except CHECKSUM_AGG and GROUPING_ID	Yes, all Transact-SQL built-in aggregates are supported.
Operators	Yes, all Transact-SQL operators except !> and !<	Yes, all Transact-SQL operators are supported.

	DEDICATED	SERVERLESS
Control of flow	Yes. All Transact-SQL Control-of-flow statement except CONTINUE, GOTO, RETURN, USE, and WAITFOR	Yes. All Transact-SQL Control-of-flow statements are supported. SELECT query in `WHILE (...)` condition is not supported.
DDL statements (CREATE, ALTER, DROP)	Yes. All Transact-SQL DDL statement applicable to the supported object types	Yes, all Transact-SQL DDL statement applicable to the supported object types are supported.

Security

Synapse SQL pools enable you to use built-in security features to secure your data and control access. The following table compares high-level differences between Synapse SQL consumption models.

	DEDICATED	SERVERLESS
Logins	N/A (only contained users are supported in databases)	Yes, server-level Azure AD and SQL logins are supported.
Users	N/A (only contained users are supported in databases)	Yes, database users are supported.
Contained users	Yes. **Note:** only one Azure AD user can be unrestricted admin	No, the contained users are not supported.
SQL username/password authentication	Yes	Yes, users can access serverless SQL pool using their usernames and passwords.
Azure Active Directory (Azure AD) authentication	Yes, Azure AD users	Yes, Azure AD logins and users can access serverless SQL pools using their Azure AD identities.
Storage Azure Active Directory (Azure AD) passthrough authentication	Yes	Yes, Azure AD passthrough authentication is applicable to Azure AD logins. The identity of the Azure AD user is passed to the storage if a credential is not specified. Azure AD passthrough authentication is not available for the SQL users.
Storage shared access signature (SAS) token authentication	No	Yes, using DATABASE SCOPED CREDENTIAL with shared access signature token in EXTERNAL DATA SOURCE or instance-level CREDENTIAL with shared access signature.
Storage Access Key authentication	Yes, using DATABASE SCOPED CREDENTIAL in EXTERNAL DATA SOURCE	No, use SAS token instead of storage access key.
Storage Managed Identity authentication	Yes, using Managed Service Identity Credential	Yes, The query can access the storage using the workspace Managed Identity credential.

	DEDICATED	SERVERLESS
Storage Application identity/Service principal (SPN) authentication	Yes	Yes, you can create a credential with a service principal application ID that will be used to authenticate on the storage.
Server roles	No	Yes, sysadmin, public, and other server-roles are supported.
SERVER SCOPED CREDENTIAL	No	Yes, the server scoped credentials are used by the `OPENROWSET` function that do not uses explicit data source.
Permissions - Server-level	No	Yes, for example, `CONNECT ANY DATABASE` and `SELECT ALL USER SECURABLES` enable a user to read data from any databases.
Database roles	Yes	Yes, you can use `db_owner`, `db_datareader` and `db_ddladmin` roles.
DATABASE SCOPED CREDENTIAL	Yes, used in external data sources.	Yes, database scoped credentials can be used in external data sources to define storage authentication method.
Permissions - Database-level	Yes	Yes, you can grant, deny, or revoke permissions on the database objects.
Permissions - Schema-level	Yes, including ability to GRANT, DENY, and REVOKE permissions to users/logins on the schema	Yes, you can specify schema-level permissions including ability to GRANT, DENY, and REVOKE permissions to users/logins on the schema.
Permissions - Object-level	Yes, including ability to GRANT, DENY, and REVOKE permissions to users	Yes, you can GRANT, DENY, and REVOKE permissions to users/logins on the system objects that are supported.
Permissions - Column-level security	Yes	Yes, column-level security is supported in serverless SQL pools. Column level encryption is also generally available, see Encrypt a column of data.
Row-level security	Yes	No, there is no built-in support for the row-level security. Use custom views as a workaround.
Data masking	Yes	No, built-in data masking is not supported in the serverless SQL pools. Use wrapper SQL views that explicitly mask some columns as a workaround.

	DEDICATED	SERVERLESS
Built-in/system security & identity functions	Some Transact-SQL security functions and operators: `CURRENT_USER`, `HAS_DBACCESS`, `IS_MEMBER`, `IS_ROLEMEMBER`, `SESSION_USER`, `SUSER_NAME`, `SUSER_SNAME`, `SYSTEM_USER`, `USER`, `USER_NAME`, `EXECUTE AS`, `OPEN/CLOSE MASTER KEY`	Some Transact-SQL security functions and operators are supported: `CURRENT_USER`, `HAS_DBACCESS`, `HAS_PERMS_BY_NAME`, `IS_MEMBER`, `IS_ROLEMEMBER`, `IS_SRVROLEMEMBER`, `SESSION_USER`, `SESSION_CONTEXT`, `SUSER_NAME`, `SUSER_SNAME`, `SYSTEM_USER`, `USER`, `USER_NAME`, `EXECUTE AS`, and `REVERT`. Security functions cannot be used to query external data (store the result in variable that can be used in the query).
Transparent Data Encryption (TDE)	Yes	No, Transparent Data Encryption is not supported.
Data Discovery & Classification	Yes	No, Data Discovery & Classification is not supported.
Vulnerability Assessment	Yes	No, Vulnerability Assessment is not available.
Advanced Threat Protection	Yes	No, Advanced Threat Protection is not supported.
Auditing	Yes	Yes, auditing is supported in serverless SQL pools.
Firewall rules	Yes	Yes, the firewall rules can be set on the serverless SQL endpoint.
Private endpoint	Yes	Yes, the private endpoint can be set on the serverless SQL pool.

Dedicated SQL pool and serverless SQL pool use standard Transact-SQL language to query data. For detailed differences, look at the Transact-SQL language reference.

Tools

You can use various tools to connect to Synapse SQL to query data.

	DEDICATED	SERVERLESS
Synapse Studio	Yes, SQL scripts	Yes, SQL scripts can be used in Synapse Studio. Use SSMS or ADS instead of Synapse Studio if you are returning a large amount of data as a result.
Power BI	Yes	Yes, you can use Power BI to create reports on serverless SQL pool. Import mode is recommended for reporting.

	DEDICATED	SERVERLESS
Azure Analysis Service	Yes	Yes, you can load data in Azure Analysis Service using the serverless SQL pool.
Azure Data Studio (ADS)	Yes	Yes, you can use Azure Data Studio (version 1.18.0 or higher) to query a serverless SQL pool. SQL scripts and SQL notebooks are supported.
SQL Server Management Studio (SSMS)	Yes	Yes, you can use SQL Server Management Studio (version 18.5 or higher) to query a serverless SQL pool. SSMS shows only the objects that are available in the serverless SQL pools.

> **NOTE**
> You can use SSMS to connect to serverless SQL pool and query. It is partially supported starting from version 18.5, you can use it to connect and query only.

Most of the applications use standard Transact-SQL language can query both dedicated and serverless consumption models of Synapse SQL.

Data access

Data that is analyzed can be stored on various storage types. The following table lists all available storage options:

	DEDICATED	SERVERLESS
Internal storage	Yes	No, data is placed in Azure Data Lake or Cosmos DB analytical storage.
Azure Data Lake v2	Yes	Yes, you can use external tables and the `OPENROWSET` function to read data from ADLS. Learn here how to setup access control.
Azure Blob Storage	Yes	Yes, you can use external tables and the `OPENROWSET` function to read data from Azure Blob Storage. Learn here how to setup access control.
Azure SQL/SQL Server (remote)	No	No, serverless SQL pool cannot reference Azure SQL database. You can reference serverless SQL pools from Azure SQL using elastic queries or linked servers.
Dataverse	No, you can load CosmosDB data into a dedicated pool using Azure Synapse Link in serverless SQL pool (via ADLS) or Spark.	Yes, you can read Dataverse tables using Azure Synapse link for Dataverse with Azure Data Lake.

	DEDICATED	SERVERLESS
Azure Cosmos DB transactional storage	No	No, you cannot access Cosmos DB containers to update data or read data from the Cosmos DB transactional storage. Use Spark pools to update the Cosmos DB transactional storage.
Azure Cosmos DB analytical storage	No, you can load CosmosDB data into a dedicated pool using Azure Synapse Link in serverless SQL pool (via ADLS), ADF, Spark or some other load tool.	Yes, you can query Cosmos DB analytical storage using Azure Synapse Link.
Apache Spark tables (in workspace)	No	Yes, serverless pool can read PARQUET and CSV tables using metadata synchronization.
Apache Spark tables (remote)	No	No, serverless pool can access only the PARQUET and CSV tables that are created in Apache Spark pools in the same Synapse workspace. However, you can manually create an external table that reference external Spark table location.
Databricks tables (remote)	No	No, serverless pool can access only the PARQUET and CSV tables that are created in Apache Spark pools in the same Synapse workspace. However, you can manually create an external table that reference Databricks table location.

Data formats

Data that is analyzed can be stored in various storage formats. The following table lists all available data formats that can be analyzed:

	DEDICATED	SERVERLESS
Delimited	Yes	Yes, you can query delimited files.
CSV	Yes (multi-character delimiters not supported)	Yes, you can query CSV files. For better performance use PARSER_VERSION 2.0 that provides faster parsing. If you are appending rows to your CSV files, make sure that you query the files as appendable.
Parquet	Yes	Yes, you can query Parquet files, including the files with nested types.
Hive ORC	Yes	No, serverless SQL pools cannot read Hive ORC format.
Hive RC	Yes	No, serverless SQL pools cannot read Hive RC format.

	DEDICATED	SERVERLESS
JSON	Yes	Yes, you can query JSON files using delimited text format and the T-SQL JSON functions.
Avro	No	No, serverless SQL pools cannot read Avro format.
Delta Lake	No	Yes, you can query delta lake files, including the files with nested types.
Common Data Model (CDM)	No	No, serverless SQL pool cannot read data stored using Common Data Model.

Next steps

Additional information on best practices for dedicated SQL pool and serverless SQL pool can be found in the following articles:

- Best practices for dedicated SQL pool
- Best practices for serverless SQL pool

Synapse SQL resource consumption

8/19/2022 • 7 minutes to read • Edit Online

This article describes resource consumption models of Synapse SQL.

Serverless SQL pool

Serverless SQL pool is a pay per query service that doesn't require you to pick the right size. The system automatically adjusts based on your requirements, freeing you up from managing your infrastructure and picking the right size for your solution.

Dedicated SQL pool - Data Warehouse Units (DWUs) and compute Data Warehouse Units (cDWUs)

Recommendations on choosing the ideal number of data warehouse units (DWUs) to optimize price and performance, and how to change the number of units.

Data Warehouse Units

A Synapse SQL pool represents a collection of analytic resources that are being provisioned. Analytic resources are defined as a combination of CPU, memory, and IO. These three resources are bundled into units of compute scale called Data Warehouse Units (DWUs). A DWU represents an abstract, normalized measure of compute resources and performance. A change to your service level alters the number of DWUs that are available to the system. In turn, this change adjusts the performance and cost of your system.

For higher performance, you can increase the number of data warehouse units. For less performance, reduce data warehouse units. Storage and compute costs are billed separately, so changing data warehouse units does not affect storage costs.

Performance for data warehouse units is based on these data warehouse workload metrics:

- How fast a standard data warehousing query can scan a large number of rows and then perform a complex aggregation. This operation is I/O and CPU intensive.
- How fast the data warehouse can ingest data from Azure Storage Blobs or Azure Data Lake. This operation is network and CPU intensive.
- How fast the `CREATE TABLE AS SELECT` T-SQL command can copy a table. This operation involves reading data from storage, distributing it across the nodes of the appliance and writing to storage again. This operation is CPU, IO, and network intensive.

Increasing DWUs:

- Linearly changes performance of the system for scans, aggregations, and CTAS statements
- Increases the number of readers and writers for PolyBase load operations
- Increases the maximum number of concurrent queries and concurrency slots.

Service Level Objective

The Service Level Objective (SLO) is the scalability setting that determines the cost and performance level of your data warehouse. The service levels for Gen2 are measured in compute data warehouse units (cDWU), for example DW2000c. Gen1 service levels are measured in DWUs, for example DW2000.

The Service Level Objective (SLO) is the scalability setting that determines the cost and performance level of your data warehouse. The service levels for Gen2 dedicated SQL pool are measured in data warehouse units

(DWU), for example DW2000c.

> **NOTE**
> Azure Synapse Analytics Gen2 recently added additional scale capabilities to support compute tiers as low as 100 cDWU. Existing data warehouses currently on Gen1 that require the lower compute tiers can now upgrade to Gen2 in the regions that are currently available for no additional cost. If your region is not yet supported, you can still upgrade to a supported region. For more information, see Upgrade to Gen2.

In T-SQL, the SERVICE_OBJECTIVE setting determines the service level and the performance tier for your dedicated SQL pool.

```
CREATE DATABASE mySQLDW
(Edition = 'Datawarehouse'
 ,SERVICE_OBJECTIVE = 'DW1000c'
)
;
```

Performance Tiers and Data Warehouse Units

Each performance tier uses a slightly different unit of measure for their data warehouse units. This difference is reflected on the invoice as the unit of scale directly translates to billing.

- Gen1 data warehouses are measured in Data Warehouse Units (DWUs).
- Gen2 data warehouses are measured in compute Data Warehouse Units (cDWUs).

Both DWUs and cDWUs support scaling compute up or down, and pausing compute when you don't need to use the data warehouse. These operations are all on-demand. Gen2 uses a local disk-based cache on the compute nodes to improve performance. When you scale or pause the system, the cache is invalidated and so a period of cache warming is required before optimal performance is achieved.

As you increase data warehouse units, you're linearly increasing computing resources. Gen2 provides the best query performance and highest scale. Gen2 systems also make the most use of the cache.

Capacity limits

Each SQL server (for example, myserver.database.windows.net) has a Database Transaction Unit (DTU) quota that allows a specific number of data warehouse units. For more information, see the workload management capacity limits.

Assess the number of data warehouse units you need

The ideal number of data warehouse units depends very much on your workload and the amount of data you've loaded into the system.

Steps for finding the best DWU for your workload:

1. Begin by selecting a smaller DWU.
2. Monitor your application performance as you test data loads into the system, observing the number of DWUs selected compared to the performance you observe.
3. Identify any additional requirements for periodic periods of peak activity. Workloads that show significant peaks and troughs in activity may need to be scaled frequently.

SQL pool is a scale-out system that can provision vast amounts of compute and query sizeable quantities of data. To see its true capabilities for scaling, especially at larger DWUs, we recommend scaling the data set as you scale to ensure that you have enough data to feed the CPUs. For scale testing, we recommend using at least 1 TB.

> **NOTE**
> Query performance only increases with more parallelization if the work can be split between compute nodes. If you find that scaling is not changing your performance, you may need to tune your table design and/or your queries. For query tuning guidance, see Manage user queries.

Permissions

Changing the data warehouse units requires the permissions described in ALTER DATABASE.

Azure built-in roles such as SQL DB Contributor and SQL Server Contributor can change DWU settings.

View current DWU settings

To view the current DWU setting:

1. Open SQL Server Object Explorer in Visual Studio.
2. Connect to the master database associated with the logical SQL server.
3. Select from the sys.database_service_objectives dynamic management view. Here is an example:

```
SELECT  db.name [Database]
,        ds.edition [Edition]
,        ds.service_objective [Service Objective]
FROM    sys.database_service_objectives   AS ds
JOIN    sys.databases                     AS db ON ds.database_id = db.database_id
;
```

Change data warehouse units

Azure portal

To change DWUs:

1. Open the Azure portal, open your database, and select **Scale**.
2. Under **Scale**, move the slider left or right to change the DWU setting.
3. Select **Save**. A confirmation message appears. Select **yes** to confirm or **no** to cancel.

PowerShell

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

To change the DWUs, use the Set-AzSqlDatabase PowerShell cmdlet. The following example sets the service level objective to DW1000 for the database MySQLDW that is hosted on server MyServer.

```
Set-AzSqlDatabase -DatabaseName "MySQLDW" -ServerName "MyServer" -RequestedServiceObjectiveName "DW1000c"
```

For more information, see PowerShell cmdlets for Azure Synapse Analytics

T-SQL

With T-SQL you can view the current DWUsettings, change the settings, and check the progress.

To change the DWUs:

1. Connect to the master database associated with your server.

2. Use the ALTER DATABASE TSQL statement. The following example sets the service level objective to DW1000c for the database MySQLDW.

```
ALTER DATABASE MySQLDW
MODIFY (SERVICE_OBJECTIVE = 'DW1000c')
;
```

REST APIs

To change the DWUs, use the Create or Update Database REST API. The following example sets the service level objective to DW1000c for the database MySQLDW, which is hosted on server MyServer. The server is in an Azure resource group named ResourceGroup1.

```
PUT https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group-
name}/providers/Microsoft.Sql/servers/{server-name}/databases/{database-name}?api-version=2014-04-01-preview
HTTP/1.1
Content-Type: application/json; charset=UTF-8

{
    "properties": {
        "requestedServiceObjectiveName": DW1000
    }
}
```

For more REST API examples, see REST APIs for Azure Synapse Analytics.

Check status of DWU changes

DWU changes may take several minutes to complete. If you're scaling automatically, consider implementing logic to ensure that certain operations have been completed before proceeding with another action.

Checking the database state through various endpoints allows you to correctly implement automation. The portal provides notification upon completion of an operation and the databases current state but doesn't allow for programmatic checking of state.

You can't check the database state for scale-out operations with the Azure portal.

To check the status of DWU changes:

1. Connect to the master database associated with your server.
2. Submit the following query to check database state.

```
SELECT    *
FROM      sys.databases
;
```

1. Submit the following query to check status of operation

```
SELECT    *
FROM      sys.dm_operation_status
WHERE     resource_type_desc = 'Database'
AND       major_resource_id = 'MySQLDW'
;
```

This DMV returns information about various management operations on your dedicated SQL pool such as the operation and the state of the operation, which is either IN_PROGRESS or COMPLETED.

The scaling workflow

When you start a scale operation, the system first kills all open sessions, rolling back any open transactions to ensure a consistent state. For scale operations, scaling only occurs after this transactional rollback has completed.

- For a scale-up operation, the system detaches all compute nodes, provisions the additional compute nodes, and then reattaches to the storage layer.
- For a scale-down operation, the system detaches all compute nodes and then reattaches only the needed nodes to the storage layer.

Next steps

To learn more about managing performance, see Resource classes for workload management and Memory and concurrency limits.

Capacity limits for dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

Maximum values allowed for various components of dedicated SQL pool in Azure Synapse Analytics.

Workload management

CATEGORY	DESCRIPTION	MAXIMUM
Data Warehouse Units (DWU)	Max DWU for a single dedicated SQL pool	Gen1: DW6000 Gen2: DW30000c
Data Warehouse Units (DWU)	Default DTU per server	54,000 By default, each SQL server (for example, myserver.database.windows.net) has a DTU Quota of 54,000, which allows up to DW6000c. This quota is simply a safety limit. You can increase your quota by creating a support ticket and selecting *Quota* as the request type. To calculate your DTU needs, multiply the 7.5 by the total DWU needed, or multiply 9 by the total cDWU needed. For example: DW6000 x 7.5 = 45,000 DTUs DW7500c x 9 = 67,500 DTUs. You can view your current DTU consumption from the SQL server option in the portal. Both paused and unpaused databases count toward the DTU quota.
Database connection	Maximum Concurrent open sessions	1024 The number of concurrent open sessions will vary based on the selected DWU. DWU1000c and above support a maximum of 1024 open sessions. DWU500c and below, support a maximum concurrent open session limit of 512. Note, there are limits on the number of queries that can execute concurrently. When the concurrency limit is exceeded, the request goes into an internal queue where it waits to be processed.

CATEGORY	DESCRIPTION	MAXIMUM
Database connection	Maximum memory for prepared statements	20 MB
Workload management	Maximum concurrent queries	128 A maximum of 128 concurrent queries will execute and remaining queries will be queued. The number of concurrent queries can decrease when users are assigned to higher resource classes or when the data warehouse unit setting is lowered. Some queries, like DMV queries, are always allowed to run and do not impact the concurrent query limit. For more information on concurrent query execution, see the concurrency maximums article.
tempdb	Maximum GB	399 GB per DW100c. At DWU1000c, tempdb is sized to 3.99 TB.

Database objects

CATEGORY	DESCRIPTION	MAXIMUM
Database	Max size	Gen1: 240 TB compressed on disk. This space is independent of tempdb or log space, and therefore this space is dedicated to permanent tables. Clustered columnstore compression is estimated at 5X. This compression allows the database to grow to approximately 1 PB when all tables are clustered columnstore (the default table type). Gen2: Unlimited storage for columnstore tables. Rowstore portion of the database is still limited to 240 TB compressed on disk.
Table	Max size	Unlimited size for columnstore tables. 60 TB for rowstore tables compressed on disk.
Table	Tables per database	100,000
Table	Columns per table	1024 columns
Table	Bytes per column	Dependent on column data type. Limit is 8000 for char data types, 4000 for nvarchar, or 2 GB for MAX data types.

CATEGORY	DESCRIPTION	MAXIMUM
Table	Bytes per row, defined size	8060 bytes The number of bytes per row is calculated in the same manner as it is for SQL Server with page compression. Like SQL Server, row-overflow storage is supported, which enables **variable length columns** to be pushed off-row. When variable length rows are pushed off-row, only 24-byte root is stored in the main record. For more information, see Row-Overflow Data Exceeding 8 KB.
Table	Partitions per table	15,000 For high performance, we recommend minimizing the number of partitions you need while still supporting your business requirements. As the number of partitions grows, the overhead for Data Definition Language (DDL) and Data Manipulation Language (DML) operations grows and causes slower performance.
Table	Characters per partition boundary value.	4000
Index	Non-clustered indexes per table.	50 Applies to rowstore tables only.
Index	Clustered indexes per table.	1 Applies to both rowstore and columnstore tables.
Index	Index key size.	900 bytes. Applies to rowstore indexes only. Indexes on varchar columns with a maximum size of more than 900 bytes can be created if the existing data in the columns does not exceed 900 bytes when the index is created. However, later INSERT or UPDATE actions on the columns that cause the total size to exceed 900 bytes will fail.
Index	Key columns per index.	16 Applies to rowstore indexes only. Clustered columnstore indexes include all columns.
Statistics	Size of the combined column values.	900 bytes.

CATEGORY	DESCRIPTION	MAXIMUM
Statistics	Columns per statistics object.	32
Statistics	Statistics created on columns per table.	30,000
Stored Procedures	Maximum levels of nesting.	8
View	Columns per view	1,024

Loads

CATEGORY	DESCRIPTION	MAXIMUM
Polybase Loads	MB per row	1 Polybase loads rows that are smaller than 1 MB. Loading LOB data types into tables with a Clustered Columnstore Index (CCI) is not supported.
Polybase Loads	Total number of files	1,000,000 Polybase loads can not exceed more than 1M files. You may experience the following error: **Operation failed as split count exceeding upper bound of 1000000**.

Queries

CATEGORY	DESCRIPTION	MAXIMUM
Query	Queued queries on user tables.	1000
Query	Concurrent queries on system views.	100
Query	Queued queries on system views	1000
Query	Maximum parameters	2098
Batch	Maximum size	65,536*4096
SELECT results	Columns per row	4096 You can never have more than 4096 columns per row in the SELECT result. There is no guarantee that you can always have 4096. If the query plan requires a temporary table, the 1024 columns per table maximum might apply.

CATEGORY	DESCRIPTION	MAXIMUM
SELECT	Nested subqueries	32 You can never have more than 32 nested subqueries in a SELECT statement. There is no guarantee that you can always have 32. For example, a JOIN can introduce a subquery into the query plan. The number of subqueries can also be limited by available memory.
SELECT	Columns per JOIN	1024 columns You can never have more than 1024 columns in the JOIN. There is no guarantee that you can always have 1024. If the JOIN plan requires a temporary table with more columns than the JOIN result, the 1024 limit applies to the temporary table.
SELECT	Bytes per GROUP BY columns.	8060 The columns in the GROUP BY clause can have a maximum of 8060 bytes.
SELECT	Bytes per ORDER BY columns	8060 bytes The columns in the ORDER BY clause can have a maximum of 8060 bytes
Identifiers per statement	Number of referenced identifiers	65,535 The number of identifiers that can be contained in a single expression of a query is limited. Exceeding this number results in SQL Server error 8632. For more information, see Internal error: An expression services limit has been reached.
String literals	Number of string literals in a statement	20,000 The number of string constants in a single expression of a query is limited. Exceeding this number results in SQL Server error 8632.

Metadata

DMV's will reset when a dedicated SQL pool is paused or when it is scaled.

SYSTEM VIEW	MAXIMUM ROWS
sys.dm_pdw_dms_cores	100
sys.dm_pdw_dms_workers	Total number of DMS workers for the most recent 1000 SQL requests.
sys.dm_pdw_errors	10,000
sys.dm_pdw_exec_requests	10,000
sys.dm_pdw_exec_sessions	10,000
sys.dm_pdw_request_steps	Total number of steps for the most recent 1000 SQL requests that are stored in sys.dm_pdw_exec_requests.
sys.dm_pdw_sql_requests	The most recent 1000 SQL requests that are stored in sys.dm_pdw_exec_requests.

Next steps

For recommendations on using Azure Synapse, see the Cheat Sheet.

Cost management for serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

This article explains how you can estimate and manage costs for serverless SQL pool in Azure Synapse Analytics:

- Estimate amount of data processed before issuing a query
- Use cost control feature to set the budget

Understand that the costs for serverless SQL pool in Azure Synapse Analytics are only a portion of the monthly costs in your Azure bill. If you are using other Azure services, you're billed for all the Azure services and resources used in your Azure subscription, including the third-party services. This article explains how to plan for and manage costs for serverless SQL pool in Azure Synapse Analytics.

Data processed

Data processed is the amount of data that the system temporarily stores while a query is run. Data processed consists of the following quantities:

- Amount of data read from storage. This amount includes:
  - Data read while reading data.
  - Data read while reading metadata (for file formats that contain metadata, like Parquet).
- Amount of data in intermediate results. This data is transferred among nodes while the query runs. It includes the data transfer to your endpoint, in an uncompressed format.
- Amount of data written to storage. If you use CETAS to export your result set to storage, then the amount of data written out is added to the amount of data processed for the SELECT part of CETAS.

Reading files from storage is highly optimized. The process uses:

- Prefetching, which might add some overhead to the amount of data read. If a query reads a whole file, then there's no overhead. If a file is read partially, like in TOP N queries, then a bit more data is read by using prefetching.
- An optimized comma-separated value (CSV) parser. If you use PARSER_VERSION='2.0' to read CSV files, then the amounts of data read from storage slightly increase. An optimized CSV parser reads files in parallel, in chunks of equal size. Chunks don't necessarily contain whole rows. To ensure all rows are parsed, the optimized CSV parser also reads small fragments of adjacent chunks. This process adds a small amount of overhead.

Statistics

The serverless SQL pool query optimizer relies on statistics to generate optimal query execution plans. You can create statistics manually. Otherwise, serverless SQL pool creates them automatically. Either way, statistics are created by running a separate query that returns a specific column at a provided sample rate. This query has an associated amount of data processed.

If you run the same or any other query that would benefit from created statistics, then statistics are reused if possible. There's no additional data processed for statistics creation.

When statistics are created for a Parquet column, only the relevant column is read from files. When statistics are

created for a CSV column, whole files are read and parsed.

Rounding

The amount of data processed is rounded up to the nearest MB per query. Each query has a minimum of 10 MB of data processed.

What data processed doesn't include

- Server-level metadata (like logins, roles, and server-level credentials).
- Databases you create in your endpoint. Those databases contain only metadata (like users, roles, schemas, views, inline table-valued functions [TVFs], stored procedures, database-scoped credentials, external data sources, external file formats, and external tables).
  - If you use schema inference, then file fragments are read to infer column names and data types, and the amount of data read is added to the amount of data processed.
- Data definition language (DDL) statements, except for the CREATE STATISTICS statement because it processes data from storage based on the specified sample percentage.
- Metadata-only queries.

Reducing the amount of data processed

You can optimize your per-query amount of data processed and improve performance by partitioning and converting your data to a compressed column-based format like Parquet.

Examples

Imagine three tables.

- The population_csv table is backed by 5 TB of CSV files. The files are organized in five equally sized columns.
- The population_parquet table has the same data as the population_csv table. It's backed by 1 TB of Parquet files. This table is smaller than the previous one because data is compressed in Parquet format.
- The very_small_csv table is backed by 100 KB of CSV files.

Query 1: SELECT SUM(population) FROM population_csv

This query reads and parses whole files to get values for the population column. Nodes process fragments of this table, and the population sum for each fragment is transferred among nodes. The final sum is transferred to your endpoint.

This query processes 5 TB of data plus a small amount overhead for transferring sums of fragments.

Query 2: SELECT SUM(population) FROM population_parquet

When you query compressed and column-based formats like Parquet, less data is read than in query 1. You see this result because serverless SQL pool reads a single compressed column instead of the whole file. In this case, 0.2 TB is read. (Five equally sized columns are 0.2 TB each.) Nodes process fragments of this table, and the population sum for each fragment is transferred among nodes. The final sum is transferred to your endpoint.

This query processes 0.2 TB plus a small amount of overhead for transferring sums of fragments.

Query 3: SELECT * FROM population_parquet

This query reads all columns and transfers all data in an uncompressed format. If the compression format is 5:1, then the query processes 6 TB because it reads 1 TB and transfers 5 TB of uncompressed data.

Query 4: SELECT COUNT(*) FROM very_small_csv

This query reads whole files. The total size of files in storage for this table is 100 KB. Nodes process fragments of this table, and the sum for each fragment is transferred among nodes. The final sum is transferred to your endpoint.

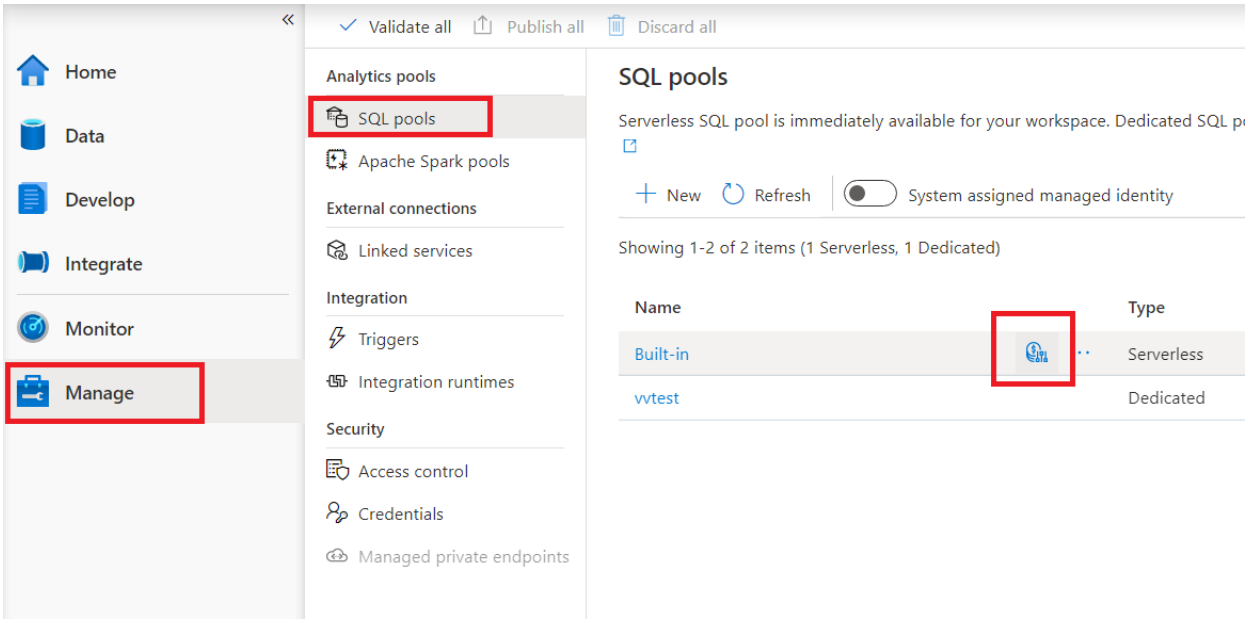
This query processes slightly more than 100 KB of data. The amount of data processed for this query is rounded up to 10 MB, as specified in the Rounding section of this article.

Cost control

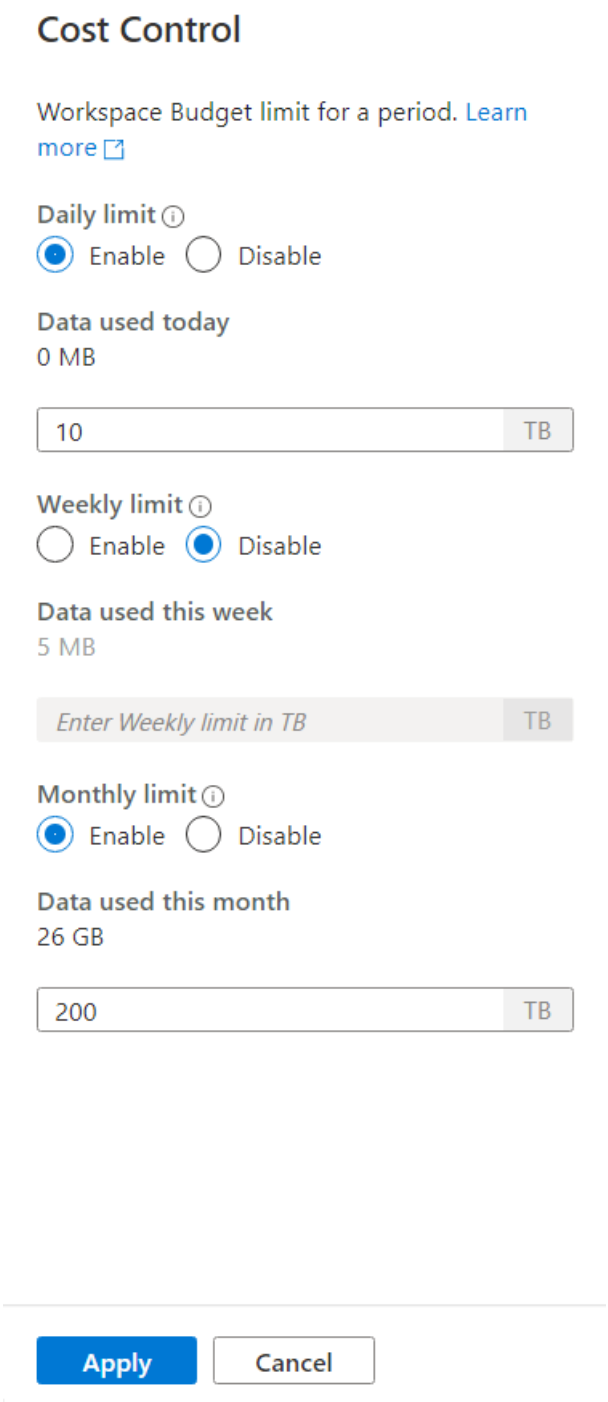
Cost control feature in serverless SQL pool enables you to set the budget for amount of data processed. You can set the budget in TB of data processed for a day, week, and month. At the same time you can have one or more budgets set. To configure cost control for serverless SQL pool, you can use Synapse Studio or T-SQL.

Configure cost control for serverless SQL pool in Synapse Studio

To configure cost control for serverless SQL pool in Synapse Studio navigate to Manage item in the menu on the left, than select SQL pool item under Analytics pools. As you hover of serverless SQL pool, you will notice an icon for cost control - click on this icon.



Once you click on the cost control icon, a side bar will appear:



To set one or more budgets, first click on Enable radio button for a budget you want to set, than enter the integer value in the text box. Unit for the value is TBs. Once you have configured the budgets you wanted click on apply button at the bottom of the side bar. That's it, you budget is now set.

Configure cost control for serverless SQL pool in T-SQL

To configure cost control for serverless SQL pool in T-SQL, you need to execute one or more of the following stored procedures.

```
sp_set_data_processed_limit
 @type = N'daily',
 @limit_tb = 1

sp_set_data_processed_limit
 @type= N'weekly',
 @limit_tb = 2

sp_set_data_processed_limit
 @type= N'monthly',
 @limit_tb = 3334
```

To see the current configuration execute the following T-SQL statement:

```
SELECT * FROM sys.configurations
WHERE name like 'Data processed %';
```

To see how much data was processed during the current day, week, or month, execute the following T-SQL statement:

```
SELECT * FROM sys.dm_external_data_processed
```

Exceeding the limits defined in the cost control

In case any limit is exceeded during the query execution, query won't be terminated.

When limit is exceeded, new query will be rejected with the error message which contains details regarding the period, defined limit for that period and data processed for that period. For example, in case new query is executed, where weekly limit is set to 1 TB and it was exceeded, the error message will be:

```
Query is rejected because SQL Serverless budget limit for a period is exceeded. (Period = Weekly: Limit = 1
TB, Data processed = 1 TB))
```

Next steps

To learn how to optimize your queries for performance, see Best practices for serverless SQL pool.

Quickstart: Create a dedicated SQL pool using Synapse Studio

8/19/2022 • 3 minutes to read • Edit Online

Azure Synapse Analytics offers various analytics engines to help you ingest, transform, model, and analyze your data. A dedicated SQL pool offers T-SQL based compute and storage capabilities. After creating a dedicated SQL pool in your Synapse workspace, data can be loaded, modeled, processed, and delivered for faster analytic insight.

This quickstart describes the steps to create a dedicated SQL pool in a Synapse workspace using Synapse Studio.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

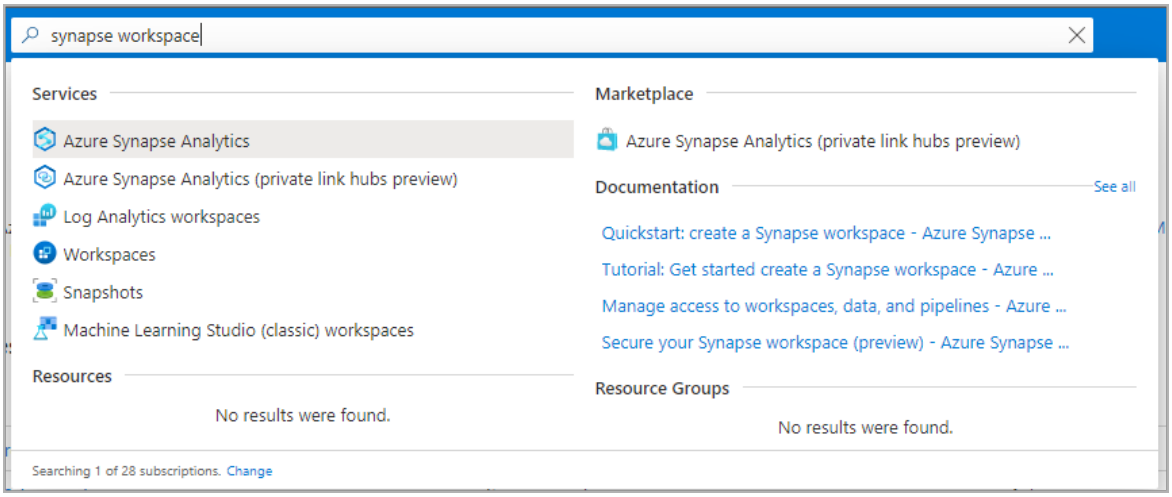
- Azure subscription - create one for free
- Synapse workspace

Sign in to the Azure portal

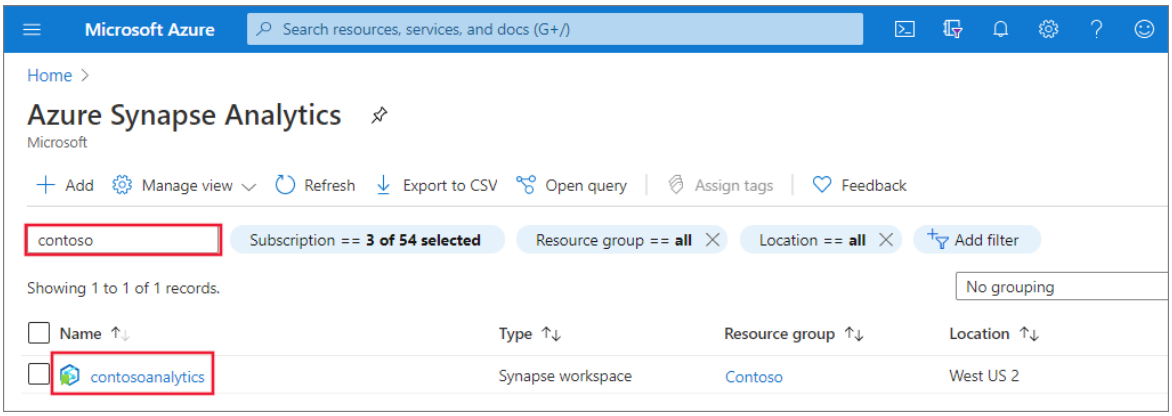
Sign in to the Azure portal

Navigate to the Synapse workspace

1. Navigate to the Synapse workspace where the dedicated SQL pool will be created by typing the service name (or resource name directly) into the search bar.

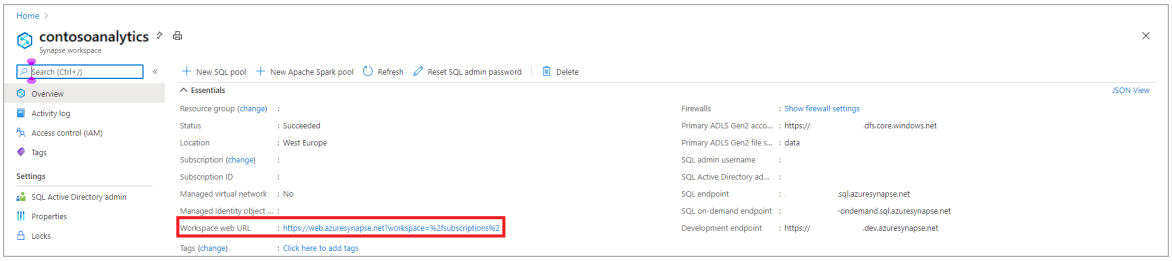


2. From the list of workspaces, type the name (or part of the name) of the workspace to open. For this example, we'll use a workspace named **contosoanalytics**.



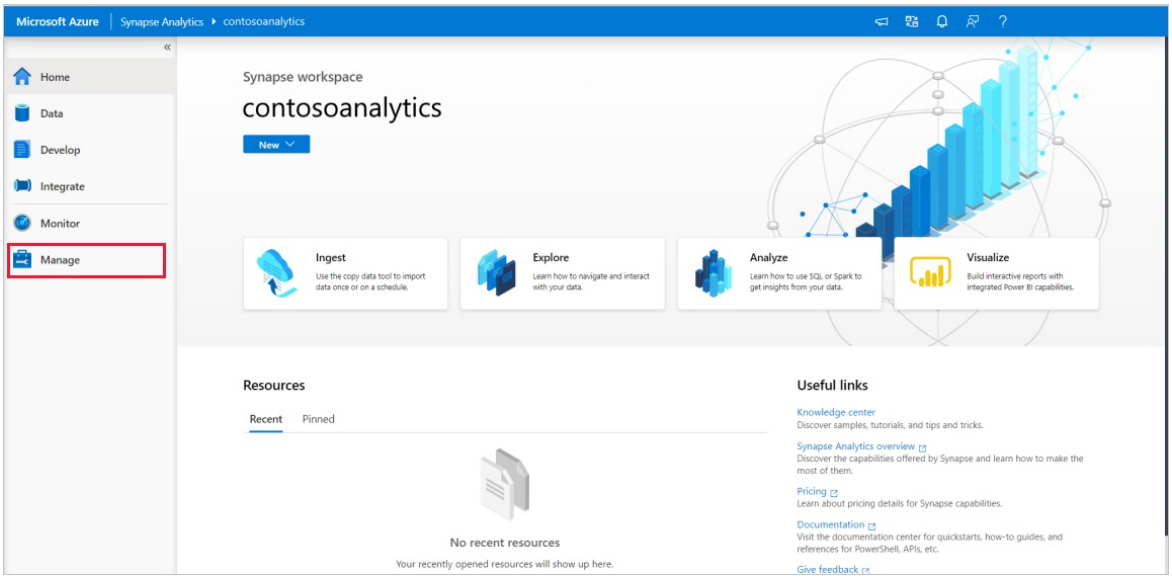
Launch Synapse Studio

1. From the workspace overview, select the **Workspace web URL** to launch Synapse Studio.

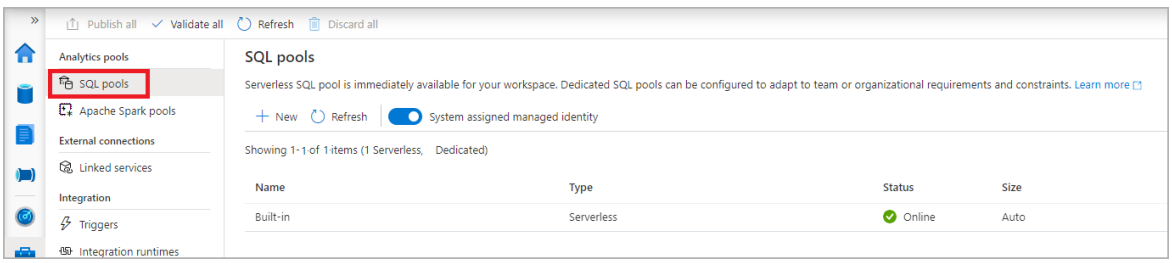


Create a dedicated SQL pool in Synapse Studio

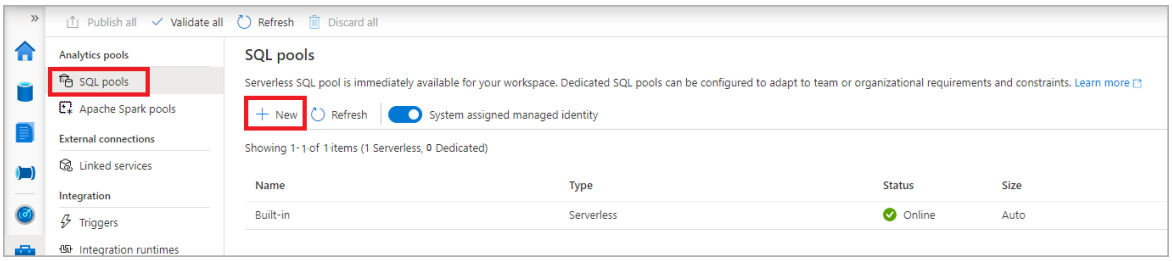
1. On the Synapse Studio home page, navigate to the **Management Hub** in the left navigation by selecting the **Manage** icon.



2. Once in the Management Hub, navigate to the **SQL pools** section to see the current list of SQL pools that are available in the workspace.

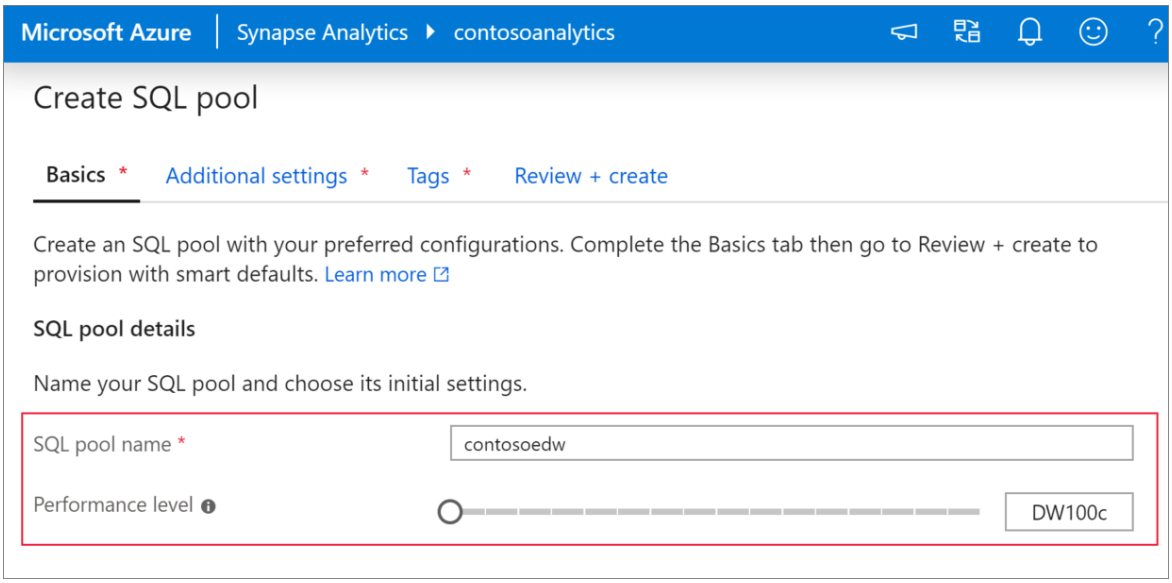


3. Select **+ New** command and the new SQL pool create wizard will appear.



4. Enter the following details in the **Basics** tab:

SETTING	SUGGESTED VALUE	DESCRIPTION
SQL pool name	contosoedw	This is the name that the dedicated SQL pool will have.
Performance level	DW100c	Set this to the smallest size to reduce costs for this quickstart

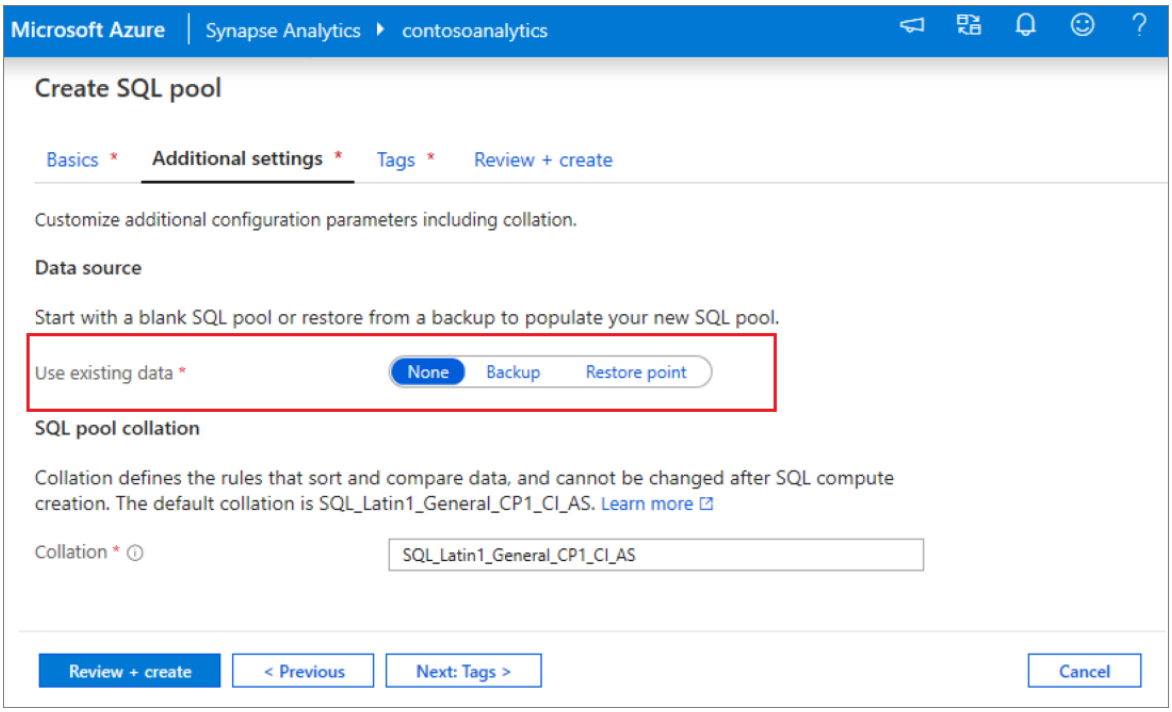


IMPORTANT

Note that there are specific limitations for the names that dedicated SQL pools can use. Names can't contain special characters, must be 60 or less characters, not contain reserved words, and be unique in the workspace.

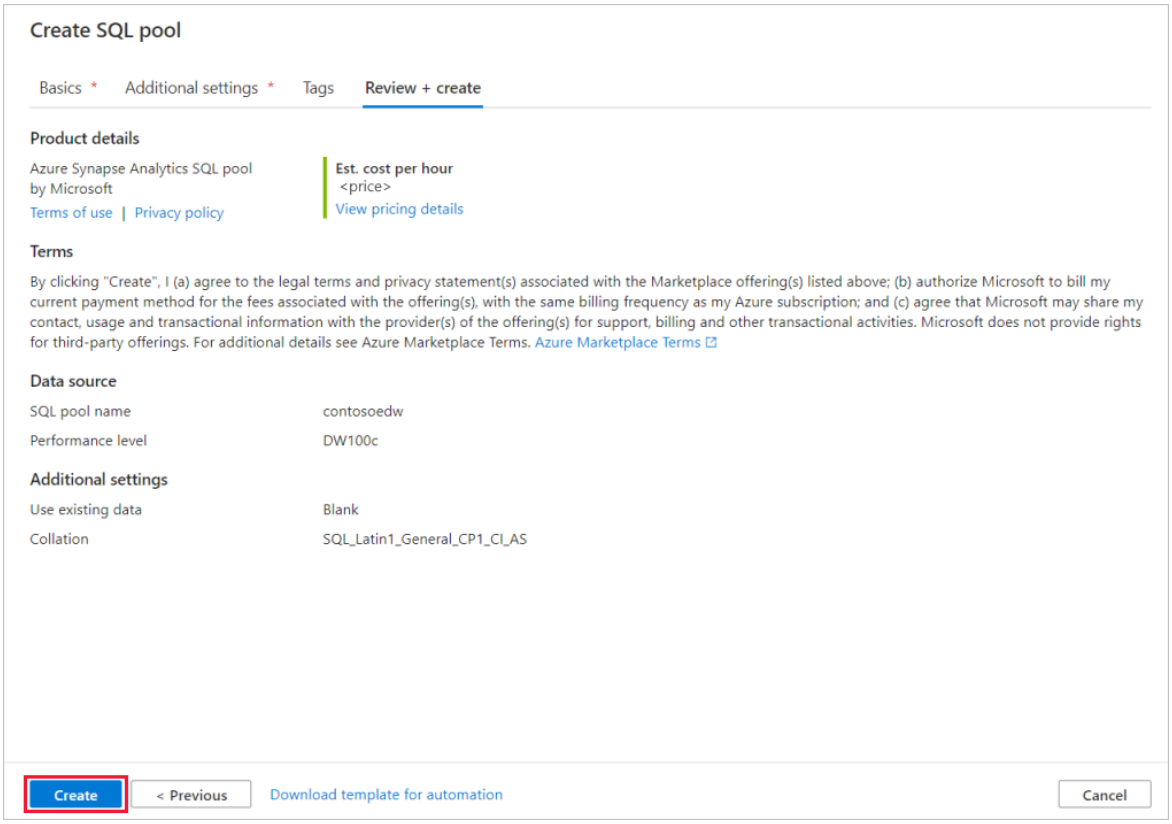
5. In the next tab, **Additional settings**, select **none** to provision the SQL pool without data. Leave the default collation as selected.

   If you want to restore your dedicated SQL pool from a restore point, select **Restore point**. For more information on how to perform a restore, see How-to: Restore an existing dedicated SQL pool



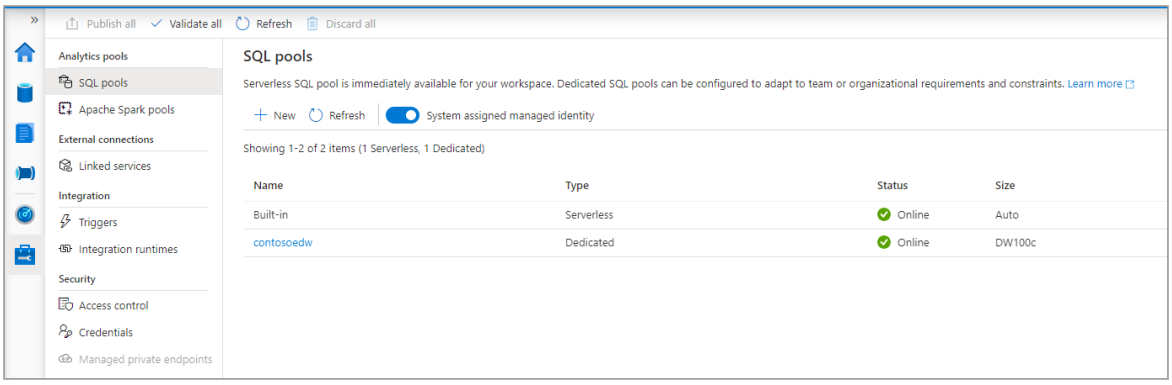
6. We won't add any tags for now, so next select **Review + create**.

7. In the **Review + create** tab, make sure that the details look correct based on what was previously entered, and press **create**.



8. At this point, the resource provisioning flow will start.

9. After the provisioning completes, navigating back to the workspace will show a new entry for the newly created SQL pool.



10. Once the dedicated SQL pool is created, it will be available in the workspace for loading data, processing streams, reading from the lake, etc.

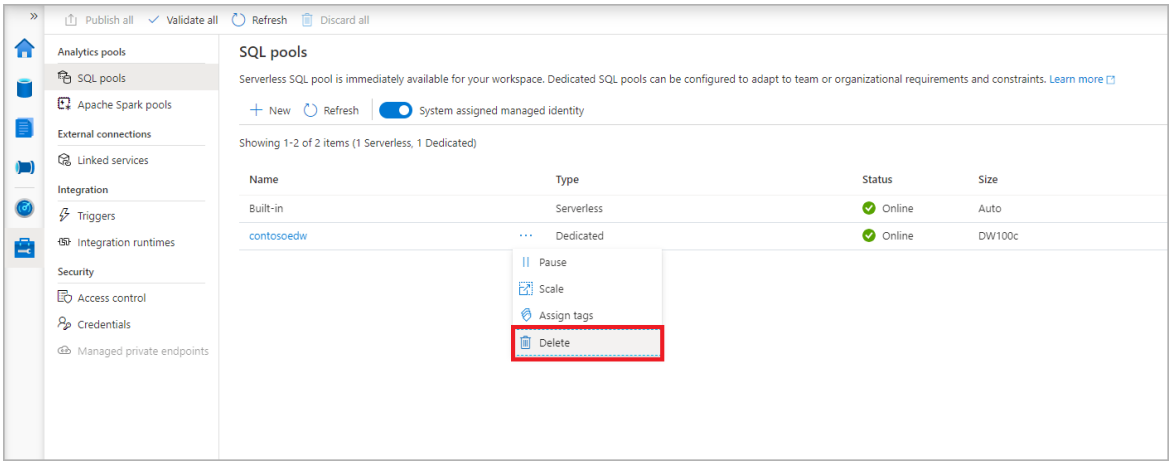
Clean up dedicated SQL pool using Synapse Studio

Follow the steps below to delete the dedicated SQL pool from the workspace using Synapse Studio.

> **WARNING**
> Deleting a dedicated SQL pool will remove the analytics engine from the workspace. It will no longer be possible to connect to the pool, and all queries, pipelines, scripts that use this dedicated SQL pool will no longer work.

If you want to delete the dedicated SQL pool, do the following:

1. Navigate to the SQL pools in the Management Hub in Synapse Studio.
2. Select the ellipsis in the dedicated SQL pool to be deleted (in this case, **contosoedw**) to show the commands for the dedicated SQL pool:



3. Press **delete**.
4. Confirm the deletion, and press **Delete** button.
5. When the process completes successfully, the dedicated SQL pool will no longer be listed in the workspace resources.

Next steps

- See Quickstart: Create an Apache Spark notebook.
- See Quickstart: Create a dedicated SQL pool using the Azure portal.

Quickstart: Create a dedicated SQL pool using the Azure portal

8/19/2022 • 2 minutes to read • Edit Online

Azure Synapse Analytics offers various analytics engines to help you ingest, transform, model, and analyze your data. A dedicated SQL pool offers T-SQL based compute and storage capabilities. After creating a dedicated SQL pool in your Synapse workspace, data can be loaded, modeled, processed, and delivered for faster analytic insight.

In this quickstart, you learn how to create a dedicated SQL pool in a Synapse workspace by using the Azure portal.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

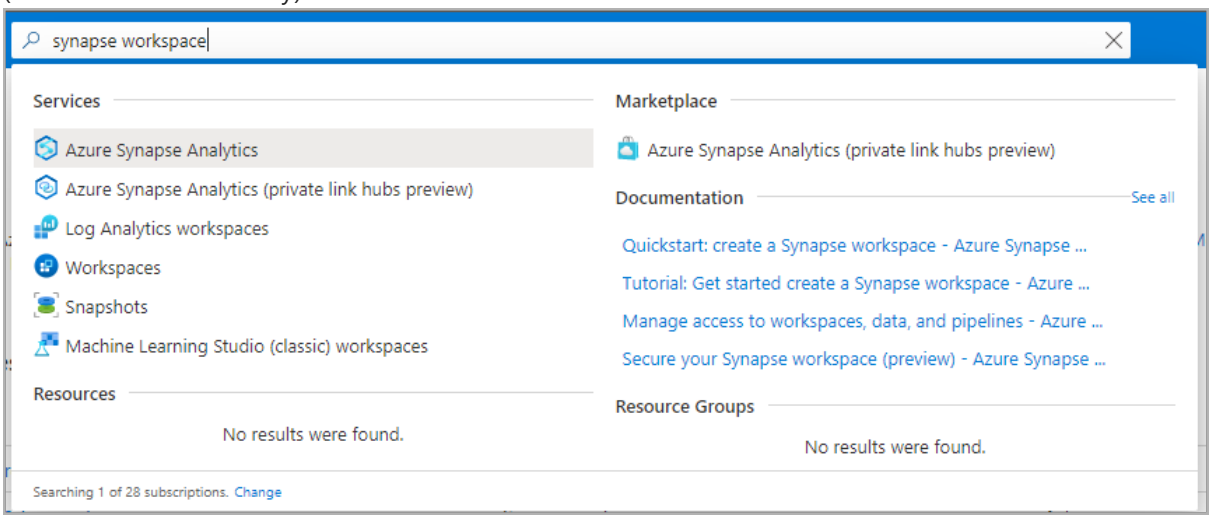
- Azure subscription - create one for free
- Synapse workspace

Sign in to the Azure portal

Sign in to the Azure portal

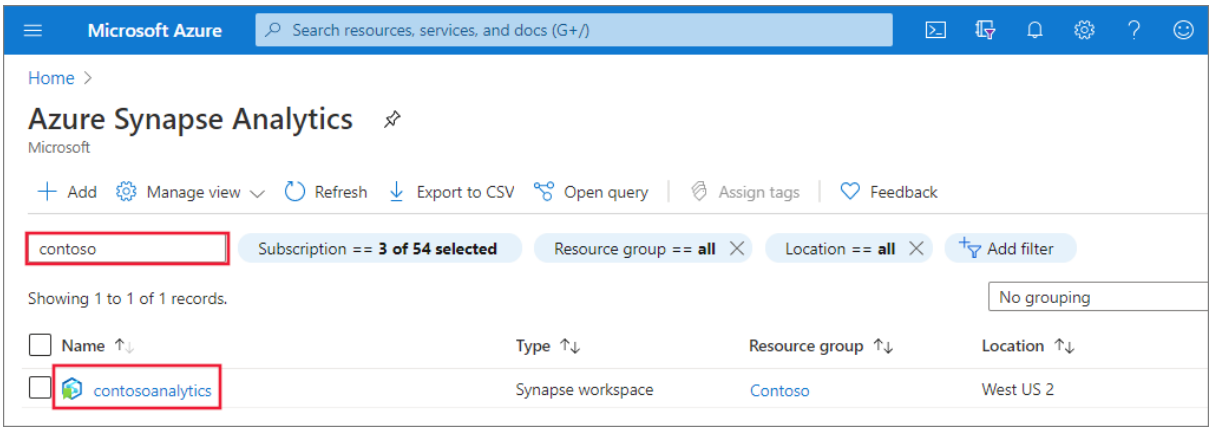
Navigate to the Synapse workspace

1. Navigate to the Synapse workspace where the dedicated SQL pool will be created by typing the service name (or resource name directly) into the search bar.



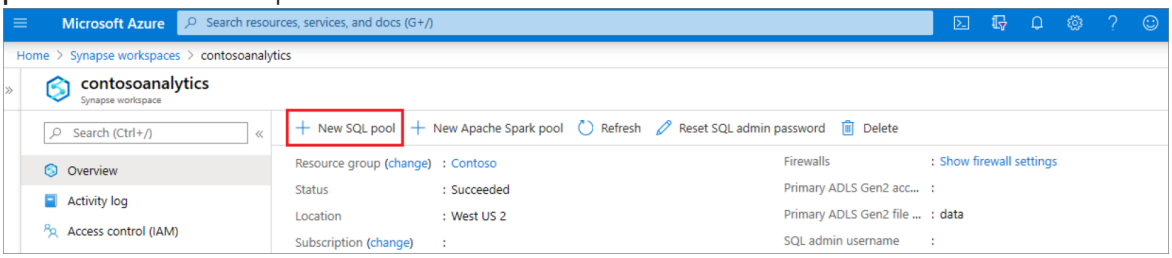
.

2. From the list of workspaces, type the name (or part of the name) of the workspace to open. For this example, we'll use a workspace named **contosoanalytics**.



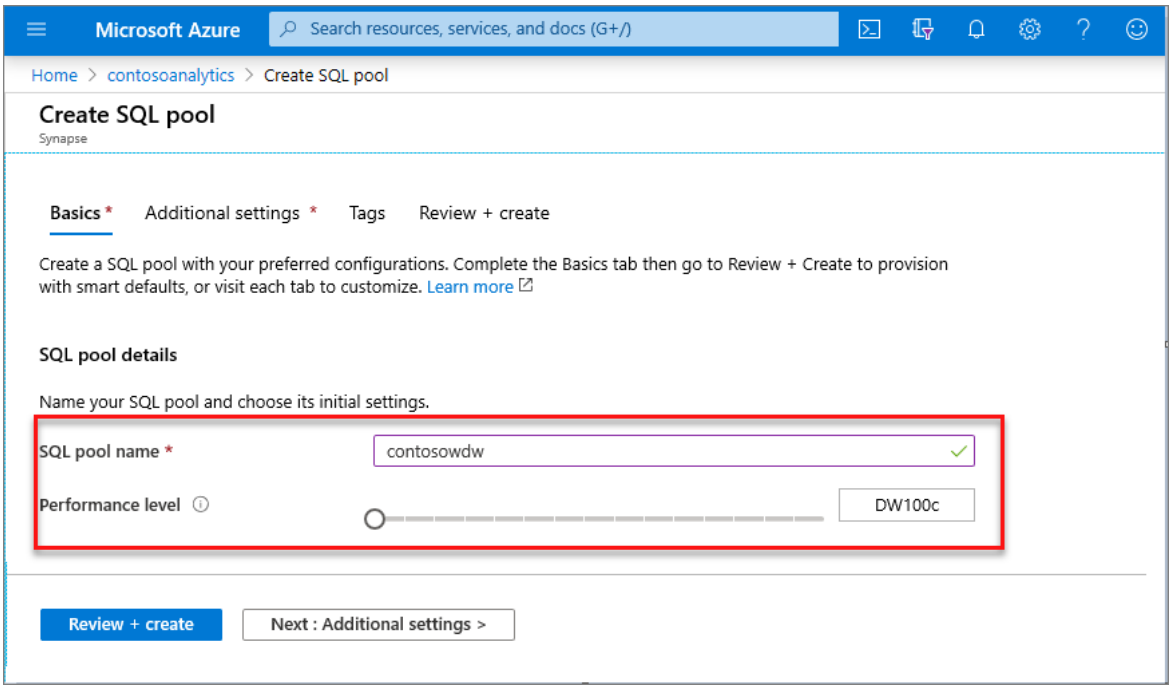
Create new dedicated SQL pool

1. In the Synapse workspace where you want to create the dedicated SQL pool, select **New dedicated SQL pool** command in the top bar.



2. Enter the following details in the **Basics** tab:

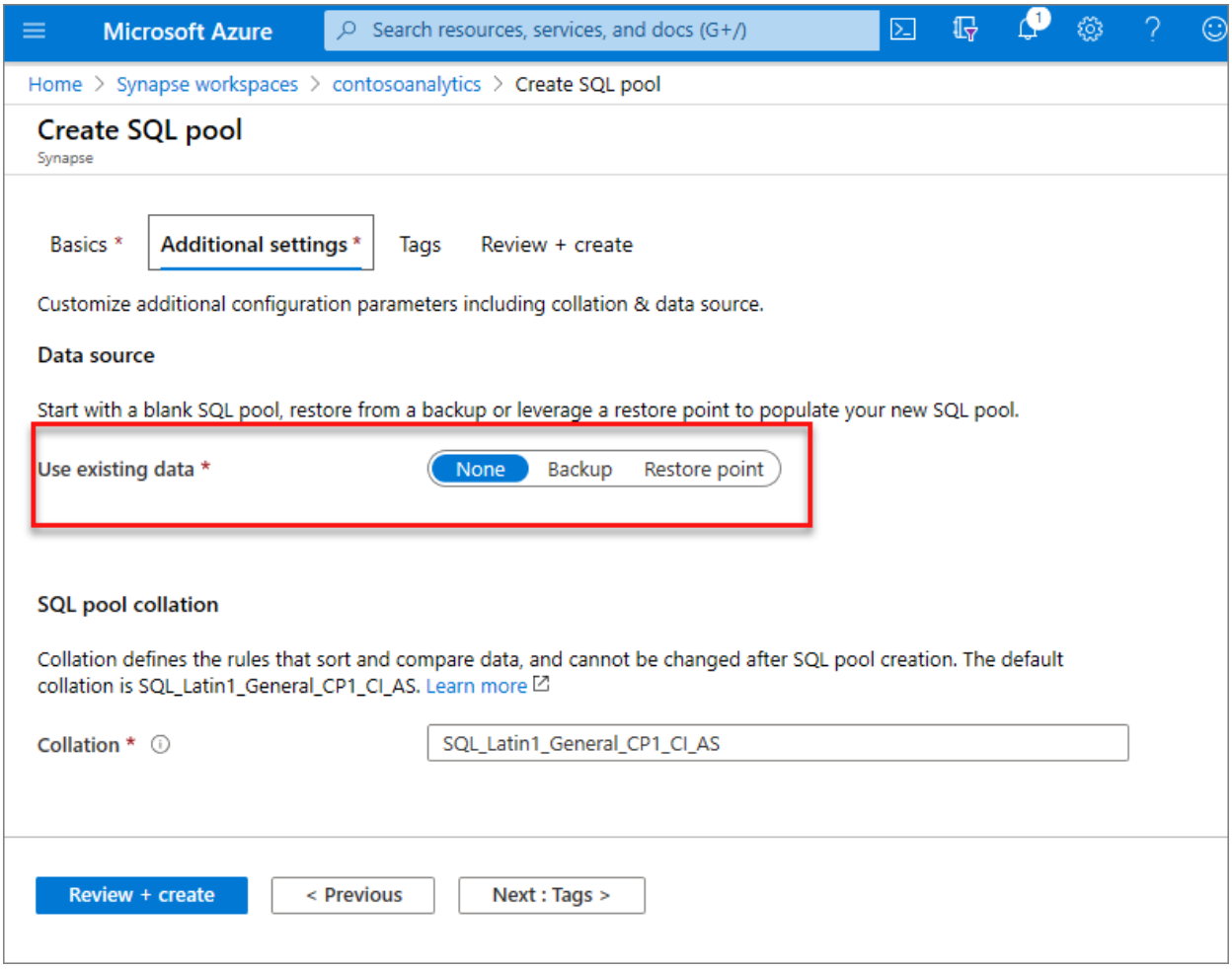
SETTING	SUGGESTED VALUE	DESCRIPTION
Dedicated SQL pool name	Any valid name	Name of the dedicated SQL pool.
Performance level	DW100c	Set to the smallest size to reduce costs for this quickstart



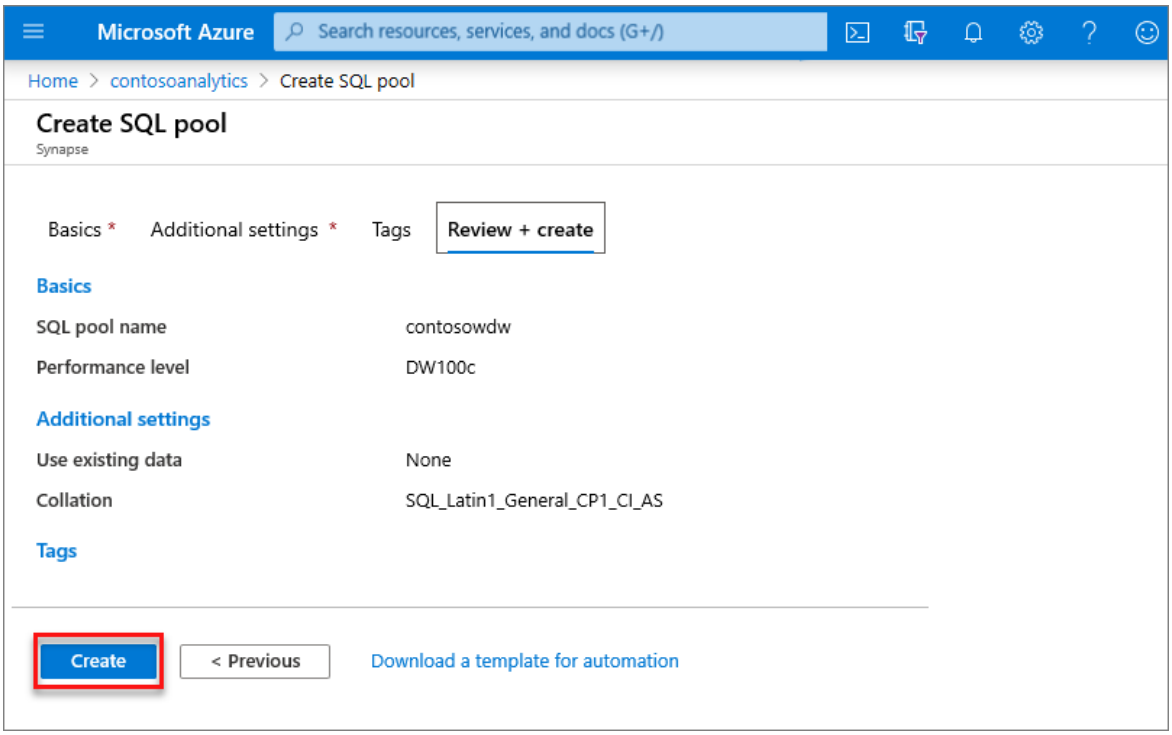
> **IMPORTANT**
>
> Note that there are specific limitations for the names that dedicated SQL pools can use. Names can't contain special characters, must be 60 or less characters, not contain reserved words, and be unique in the workspace.

3. Select **Next: Additional settings**.
4. Select **None** to provision the dedicated SQL pool without data. Leave the default collation selected.

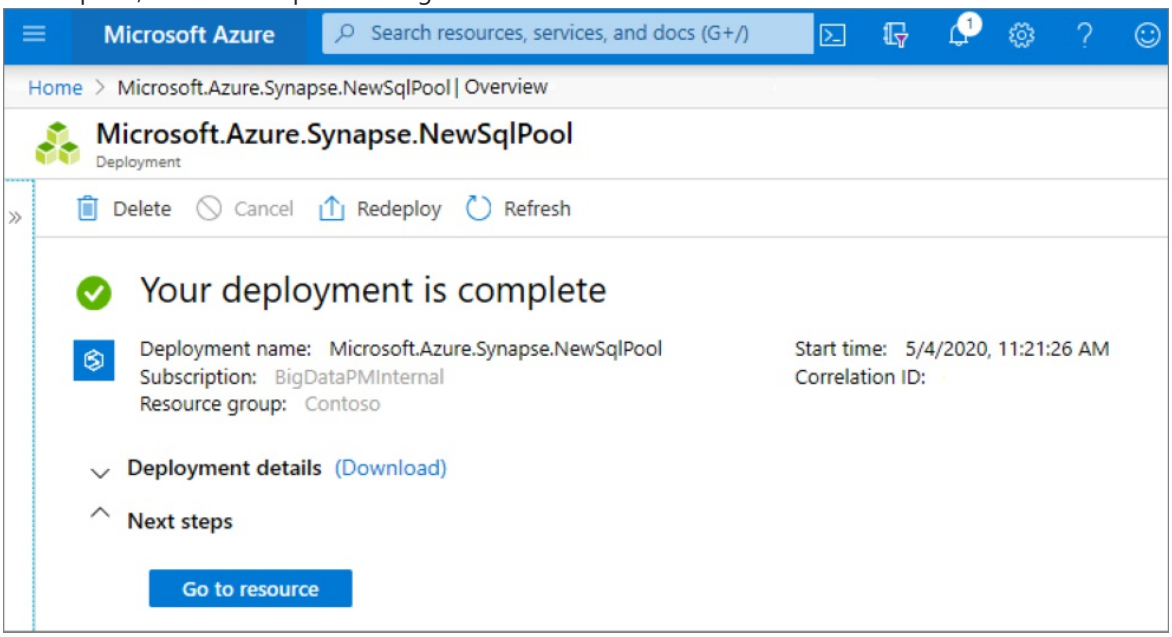
   If you want to restore your dedicated SQL pool from a restore point, select **Restore point**. For more information on how to perform a restore, see How-to: Restore an existing dedicated SQL pool



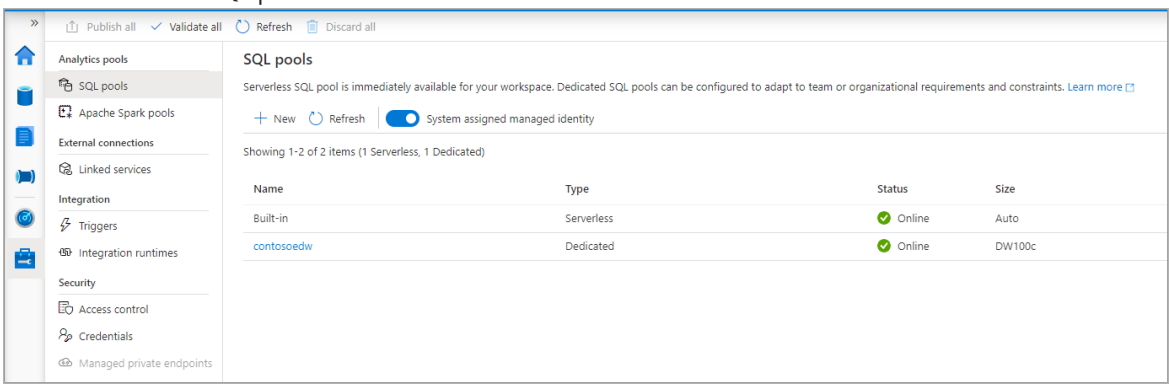
5. Select **Review + create**.
6. Make sure that the details look correct based on what was previously entered. Select **Create**.



7. At this point, the resource provisioning flow will start.



8. After the provisioning completes, navigating back to the workspace will show a new entry for the newly created dedicated SQL pool.



After the dedicated SQL pool is created, it will be available in the workspace for loading data, processing streams, reading from the lake, etc.

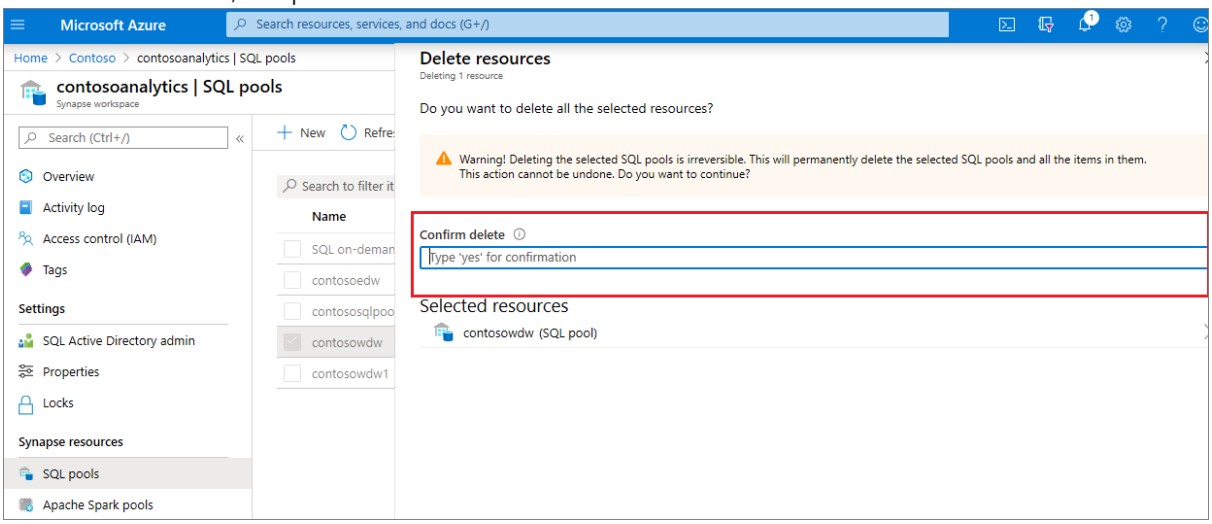
Clean up resources

Follow the steps below to delete the dedicated SQL pool from the workspace.

> **WARNING**
> Deleting a dedicated SQL pool will both remove the analytics engine and the data stored in the database of the deleted dedicated SQL pool from the workspace. It will no longer be possible to connect to the dedicated SQL pool, and all queries, pipelines, and notebooks that read or write to this dedicated SQL pool will no longer work.

If you want to delete the dedicated SQL pool, complete the following steps:

1. Navigate to the SQL pools blade in the workspace blade
2. Select the dedicated SQL pool to be deleted (in this case, **contosowdw**)
3. Once selected, press **delete**
4. Confirm the deletion, and press **Delete** button



5. When the process completes successfully, the dedicated SQL pool will no longer be listed in the workspace resources.

Next steps

- See Quickstart: Create a serverless Apache Spark pool in Synapse Studio using web tools.
- See Quickstart: Create a serverless Apache Spark pool using the Azure portal.

Quickstart: Pause and resume compute in dedicated SQL pool via the Azure portal

8/19/2022 • 2 minutes to read • Edit Online

You can use the Azure portal to pause and resume the dedicated SQL pool compute resources. If you don't have an Azure subscription, create a free account before you begin.

Sign in to the Azure portal

Sign in to the Azure portal.

Before you begin

Use Create and Connect - portal to create a dedicated SQL pool called **mySampleDataWarehouse**.

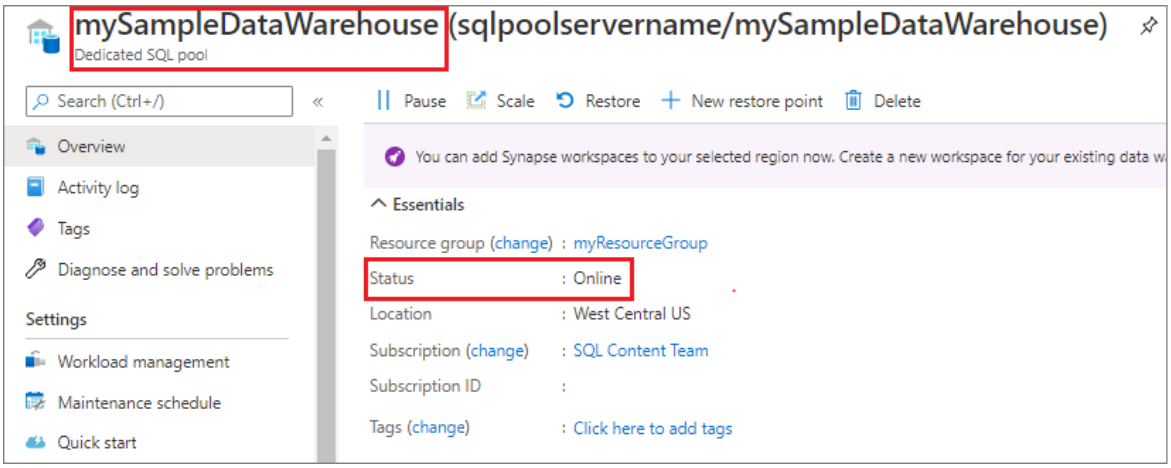
Pause compute

To reduce costs, you can pause and resume compute resources on-demand. For example, if you won't be using the database during the night and on weekends, you can pause it during those times, and resume it during the day.

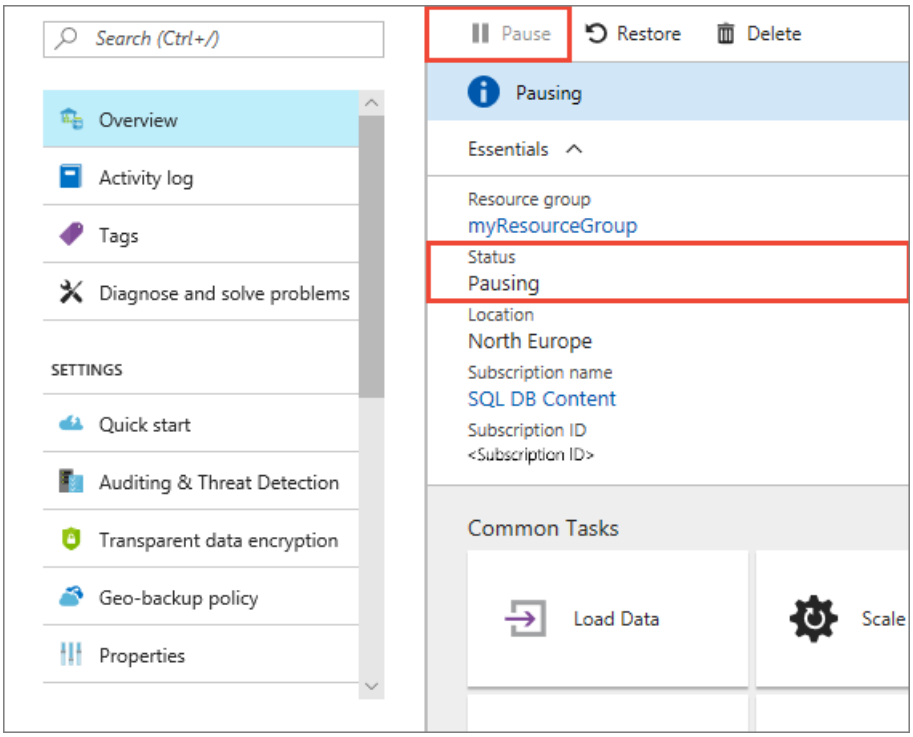
> **NOTE**
> You won't be charged for compute resources while the database is paused. However, you will continue to be charged for storage.

Follow these steps to pause a dedicated SQL pool:

1. Sign in to the Azure portal.
2. Navigate to your the **Dedicated SQL pool** page to open the SQL pool.
3. Notice **Status** is **Online**.

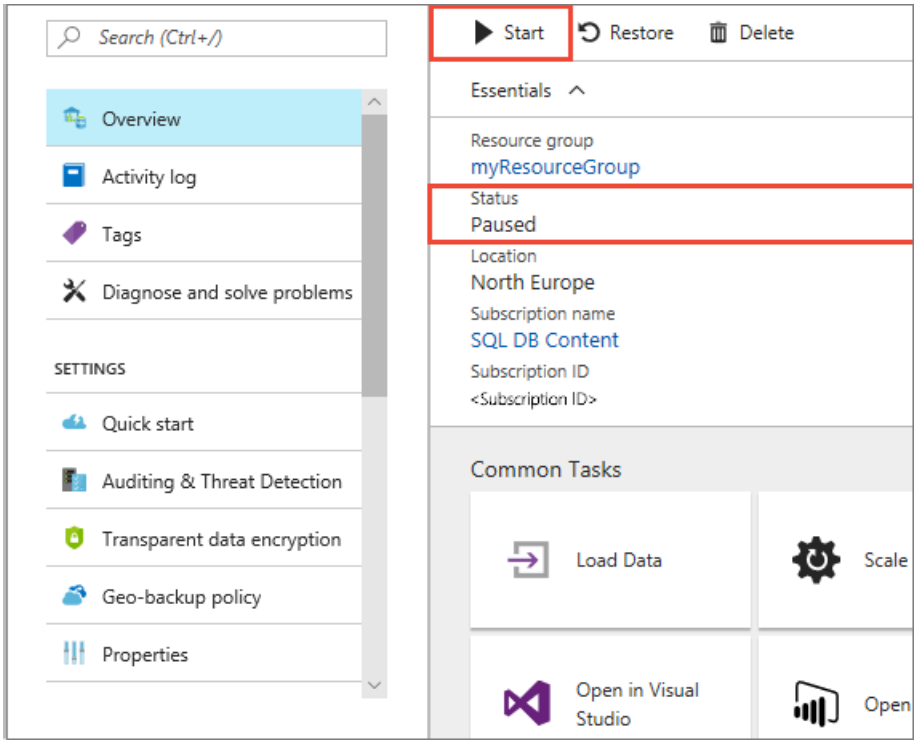


4. To pause the dedicated SQL pool, click the **Pause** button.
5. A confirmation question appears asking if you want to continue. Click **Yes**.
6. Wait a few moments, and then notice the **Status** is **Pausing**.



7. When the pause operation is complete, the status is **Paused** and the option button is **Resume**.

8. The compute resources for the dedicated SQL pool are now offline. You won't be charged for compute until you resume the service.

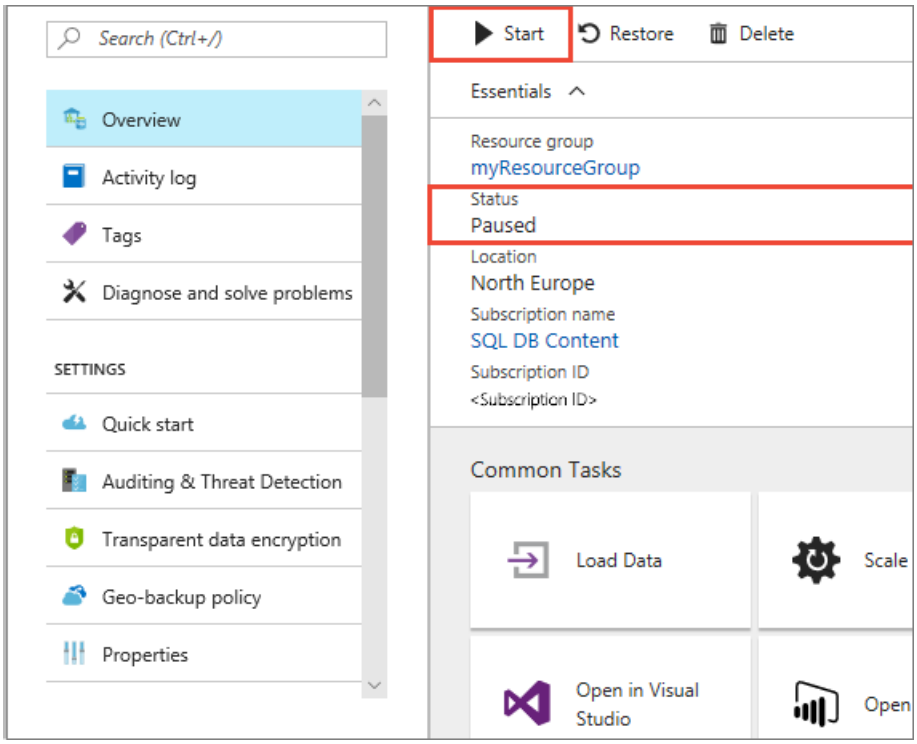


Resume compute

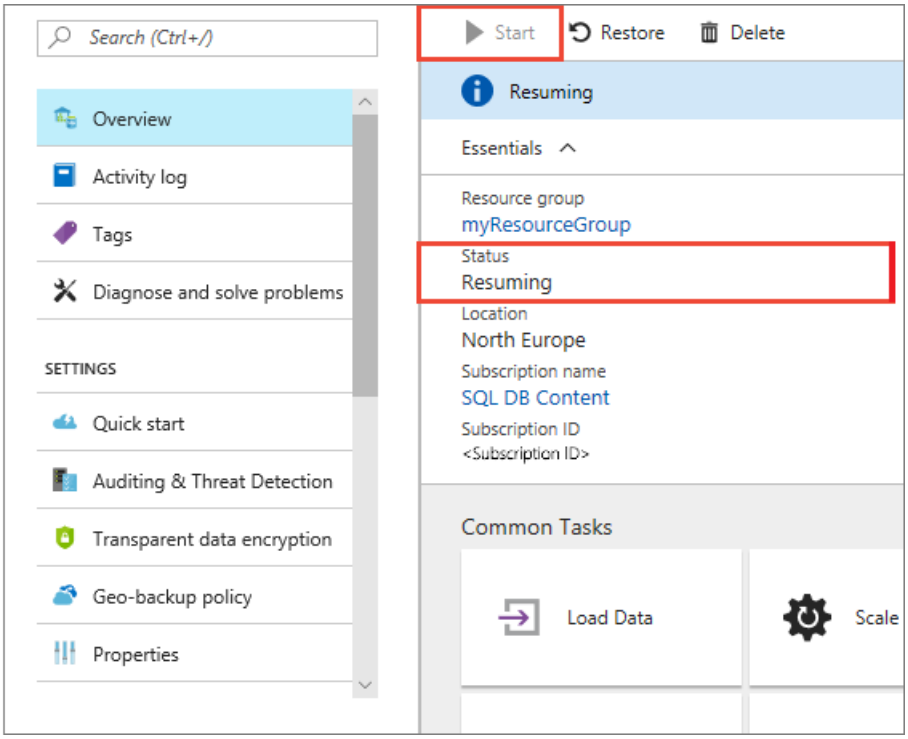
Follow these steps to resume a dedicated SQL pool.

1. Navigate to your the **Dedicated SQL pool** page to open the SQL pool.

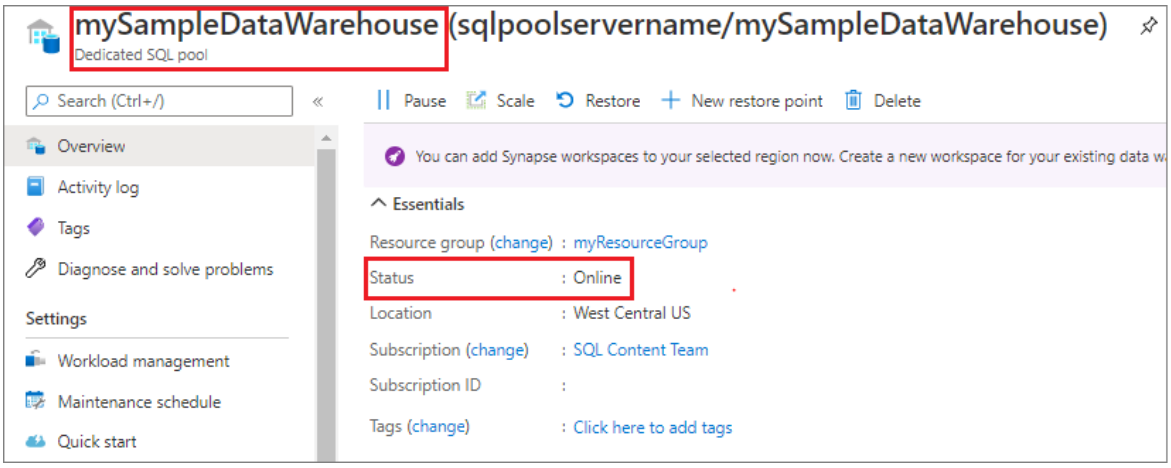
2. On the **mySampleDataWarehouse** page, notice **Status** is **Paused**.



3. To resume SQL pool, click **Resume**.
4. A confirmation question appears asking if you want to start. Click **Yes**.
5. Notice the **Status** is **Resuming**.



6. When the SQL pool is back online, the status is **Online** and the option button is **Pause**.
7. The compute resources for SQL pool are now online and you can use the service. Charges for compute have resumed.



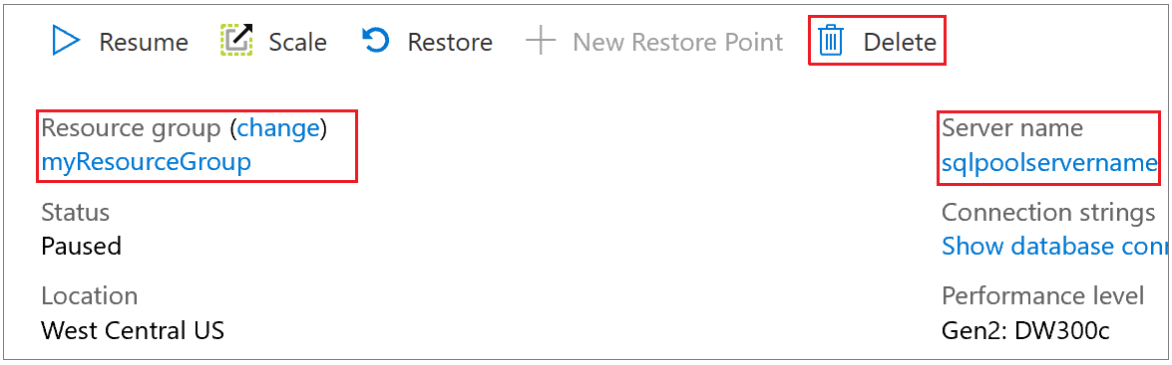
Clean up resources

You are being charged for data warehouse units and the data stored in your dedicated SQL pool. These compute and storage resources are billed separately.

- If you want to keep the data in storage, pause compute.
- If you want to remove future charges, you can delete the dedicated SQL pool.

Follow these steps to clean up resources as you desire.

1. Sign in to the Azure portal, and select your dedicated SQL pool.



2. To pause compute, click the **Pause** button.
3. To remove the dedicated SQL pool so you are not charged for compute or storage, click **Delete**.

Next steps

You have now paused and resumed compute for your dedicated SQL pool. Continue to the next article to learn more about how to Load data into a dedicated SQL pool. For additional information about managing compute capabilities, see the Manage compute overview article.

Quickstart: Bulk load data using the COPY statement

8/19/2022 • 2 minutes to read • Edit Online

In this quickstart, you'll bulk load data into your dedicated SQL pool using the simple and flexible COPY statement for high-throughput data ingestion. The COPY statement is the recommended loading utility as it enables you to seamlessly and flexibly load data by providing functionality to:

- Allow lower privileged users to load without needing strict CONTROL permissions on the data warehouse
- Leverage only a single T-SQL statement without having to create any additional database objects
- Leverage a finer permission model without exposing storage account keys using Share Access Signatures (SAS)
- Specify a different storage account for the ERRORFILE location (REJECTED_ROW_LOCATION)
- Customize default values for each target column and specify source data fields to load into specific target columns
- Specify a custom row terminator for CSV files
- Escape string, field, and row delimiters for CSV files
- Leverage SQL Server Date formats for CSV files
- Specify wildcards and multiple files in the storage location path

Prerequisites

This quickstart assumes you already have a dedicated SQL pool. If a dedicated SQL pool hasn't been created, use the Create and Connect - portal quickstart.

Set up the required permissions

```
-- List the permissions for your user
select  princ.name
,       princ.type_desc
,       perm.permission_name
,       perm.state_desc
,       perm.class_desc
,       object_name(perm.major_id)
from    sys.database_principals princ
left join
        sys.database_permissions perm
on      perm.grantee_principal_id = princ.principal_id
where name = '<yourusername>';

--Make sure your user has the permissions to CREATE tables in the [dbo] schema
GRANT CREATE TABLE TO <yourusername>;
GRANT ALTER ON SCHEMA::dbo TO <yourusername>;

--Make sure your user has ADMINISTER DATABASE BULK OPERATIONS permissions
GRANT ADMINISTER DATABASE BULK OPERATIONS TO <yourusername>

--Make sure your user has INSERT permissions on the target table
GRANT INSERT ON <yourtable> TO <yourusername>
```

Create the target table

In this example, we'll be loading data from the New York taxi dataset. We'll load a table called Trip that represents taxi trips taken within a single year. Run the following to create the table:

```
CREATE TABLE [dbo].[Trip]
(
    [DateID] int NOT NULL,
    [MedallionID] int NOT NULL,
    [HackneyLicenseID] int NOT NULL,
    [PickupTimeID] int NOT NULL,
    [DropoffTimeID] int NOT NULL,
    [PickupGeographyID] int NULL,
    [DropoffGeographyID] int NULL,
    [PickupLatitude] float NULL,
    [PickupLongitude] float NULL,
    [PickupLatLong] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DropoffLatitude] float NULL,
    [DropoffLongitude] float NULL,
    [DropoffLatLong] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [PassengerCount] int NULL,
    [TripDurationSeconds] int NULL,
    [TripDistanceMiles] float NULL,
    [PaymentType] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [FareAmount] money NULL,
    [SurchargeAmount] money NULL,
    [TaxAmount] money NULL,
    [TipAmount] money NULL,
    [TollsAmount] money NULL,
    [TotalAmount] money NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
);
```

Run the COPY statement

Run the following COPY statement that will load data from the Azure blob storage account into the Trip table.

```
COPY INTO [dbo].[Trip] FROM 'https://nytaxiblob.blob.core.windows.net/2013/Trip2013/'
WITH (
   FIELDTERMINATOR='|',
   ROWTERMINATOR='0x0A'
) OPTION (LABEL = 'COPY: dbo.trip');
```

Monitor the load

Check whether your load is making progress by periodically running the following query:

```
SELECT  r.[request_id]
,       r.[status]
,       r.resource_class
,       r.command
,       sum(bytes_processed) AS bytes_processed
,       sum(rows_processed) AS rows_processed
FROM    sys.dm_pdw_exec_requests r
              JOIN sys.dm_pdw_dms_workers w
                     ON r.[request_id] = w.request_id
WHERE [label] = 'COPY: dbo.trip' and session_id <> session_id() and type = 'WRITER'
GROUP BY r.[request_id]
,       r.[status]
,       r.resource_class
,       r.command;
```

Next steps

- For best practices on data loading, see Best Practices for Loading Data.
- For information on how to manage the resources for your data loads, see Workload Isolation.

Securely load data using Synapse SQL

8/19/2022 • 4 minutes to read • Edit Online

This article highlights and provides examples on the secure authentication mechanisms for the COPY statement. The COPY statement is the most flexible and secure way of bulk loading data in Synapse SQL.

Supported authentication mechanisms

The following matrix describes the supported authentication methods for each file type and storage account. This applies to the source storage location and the error file location.

	CSV	PARQUET	ORC
Azure blob storage	SAS/MSI/SERVICE PRINCIPAL/KEY/AAD	SAS/KEY	SAS/KEY
Azure Data Lake Gen2	SAS/MSI/SERVICE PRINCIPAL/KEY/AAD	SAS (blob[1])/MSI (dfs[2])/SERVICE PRINCIPAL/KEY/AAD	SAS (blob[1])/MSI (dfs[2])/SERVICE PRINCIPAL/KEY/AAD

1: The .blob endpoint (`.blob.core.windows.net`) in your external location path is required for this authentication method.

2: The .dfs endpoint (`.dfs.core.windows.net`) in your external location path is required for this authentication method.

A. Storage account key with LF as the row terminator (Unix-style new line)

```
--Note when specifying the column list, input field numbers start from 1
COPY INTO target_table (Col_one default 'myStringDefault' 1, Col_two default 1 3)
FROM 'https://adlsgen2account.dfs.core.windows.net/myblobcontainer/folder1/'
WITH (
    FILE_TYPE = 'CSV'
    ,CREDENTIAL=(IDENTITY= 'Storage Account Key', SECRET='<Your_Account_Key>')
    --CREDENTIAL should look something like this:
    --CREDENTIAL=(IDENTITY= 'Storage Account Key', SECRET='<secret>'),
    ,ROWTERMINATOR='0x0A' --0x0A specifies to use the Line Feed character (Unix based systems)
)
```

IMPORTANT

- Use the hexadecimal value (0x0A) to specify the Line Feed/Newline character. Note the COPY statement will interpret the `\n` string as `\r\n` (carriage return newline).

B. Shared Access Signatures (SAS) with CRLF as the row terminator (Windows style new line)

```
COPY INTO target_table
FROM 'https://adlsgen2account.dfs.core.windows.net/myblobcontainer/folder1/'
WITH (
    FILE_TYPE = 'CSV'
    ,CREDENTIAL=(IDENTITY= 'Shared Access Signature', SECRET='<Your_SAS_Token>')
    --CREDENTIAL should look something like this:
    --CREDENTIAL=(IDENTITY= 'Shared Access Signature', SECRET='?sv=2018-03-28&ss=bfqt&srt=sco&sp=rl&st=2016-
10-17T20%3A14%3A55Z&se=2021-10-18T20%3A19%3A00Z&sig=IEoOdmeYnE9%2FKiJDSFSYsz4AkN'),
    ,ROWTERMINATOR='\n'-- COPY command automatically prefixes the \r character when \n (newline) is
specified. This results in carriage return newline (\r\n) for Windows based systems.
)
```

IMPORTANT

Do not specify the `ROWTERMINATOR` as '\r\n' which will be interpreted as '\r\r\n' and can result in parsing issues. The COPY command automatically prefixes the \r character when \n (newline) is specified. This results in carriage return newline (\r\n) for Windows based systems.

C. Managed Identity

Managed Identity authentication is required when your storage account is attached to a VNet.

Prerequisites

1. Install Azure PowerShell. Refer to Install PowerShell.
2. If you have a general-purpose v1 or blob storage account, you must first upgrade to general-purpose v2. Refer to Upgrade to a general-purpose v2 storage account.
3. You must have **Allow trusted Microsoft services to access this storage account** turned on under Azure Storage account **Firewalls and Virtual networks** settings menu. Refer to Configure Azure Storage firewalls and virtual networks.

Steps

1. If you have a standalone dedicated SQL pool, register your SQL server with Azure Active Directory (Azure AD) using PowerShell:

   ```
   Connect-AzAccount
   Select-AzSubscription -SubscriptionId <subscriptionId>
   Set-AzSqlServer -ResourceGroupName your-database-server-resourceGroup -ServerName your-SQL-servername
   -AssignIdentity
   ```

   This step is not required for dedicated SQL pools within a Synapse workspace. The system assigned managed identity (SA-MI) of the workspace is a member of the Synapse Administrator role and thus has elevated privileges on the dedicated SQL pools of the workspace.

2. Create a **general-purpose v2 Storage Account**. For more information, see Create a storage account.

   NOTE

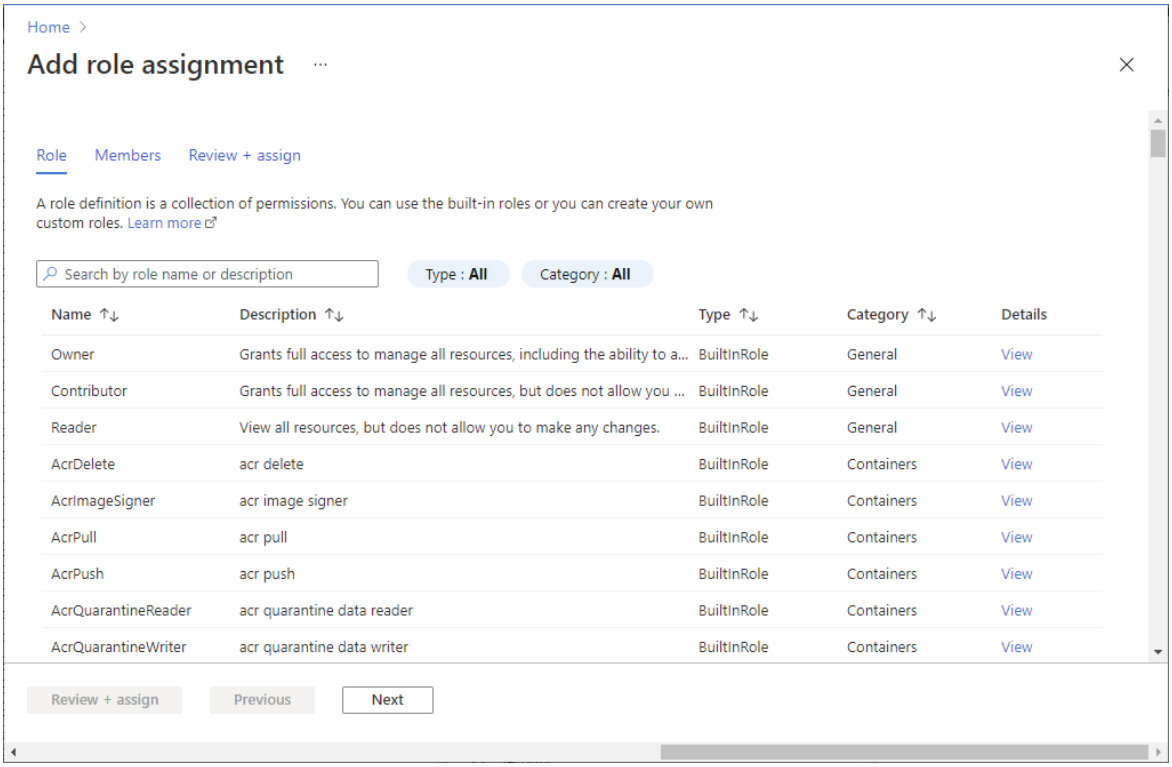
   - If you have a general-purpose v1 or blob storage account, you must **first upgrade to v2**. For more information, see Upgrade to a general-purpose v2 storage account.
   - For known issues with Azure Data Lake Storage Gen2, refer to Known issues with Azure Data Lake Storage Gen2.

3. Under your storage account, select **Access control (IAM)**.

4. Select **Add** > **Add role assignment** to open the Add role assignment page.

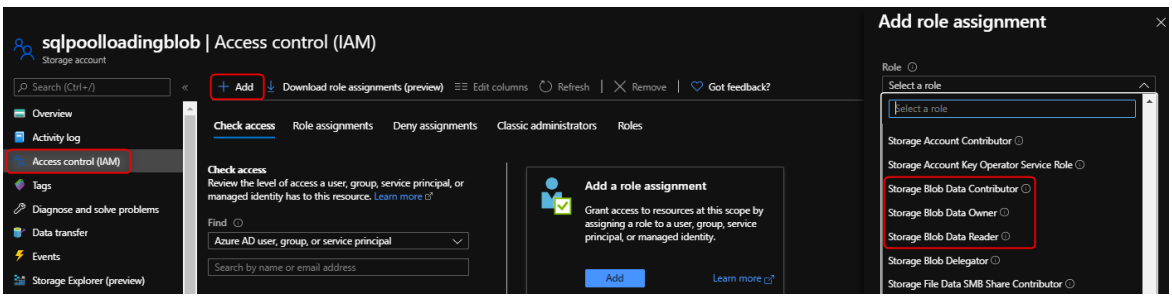
5. Assign the following role. For detailed steps, see Assign Azure roles using the Azure portal.

SETTING	VALUE
Role	Storage Blob Data Contributor
Assign access to	SERVICEPRINCIPAL
Members	server or workspace hosting your dedicated SQL pool that you've registered with Azure Active Directory (Azure AD)



> **NOTE**
>
> Only members with Owner privilege can perform this step. For various Azure built-in roles, refer to Azure built-in roles.

> **IMPORTANT**
>
> Specify the **Storage Blob Data** Owner, Contributor, or Reader Azure role. These roles are different than the Azure built-in roles of Owner, Contributor, and Reader.



6. You can now run the COPY statement specifying "Managed Identity":

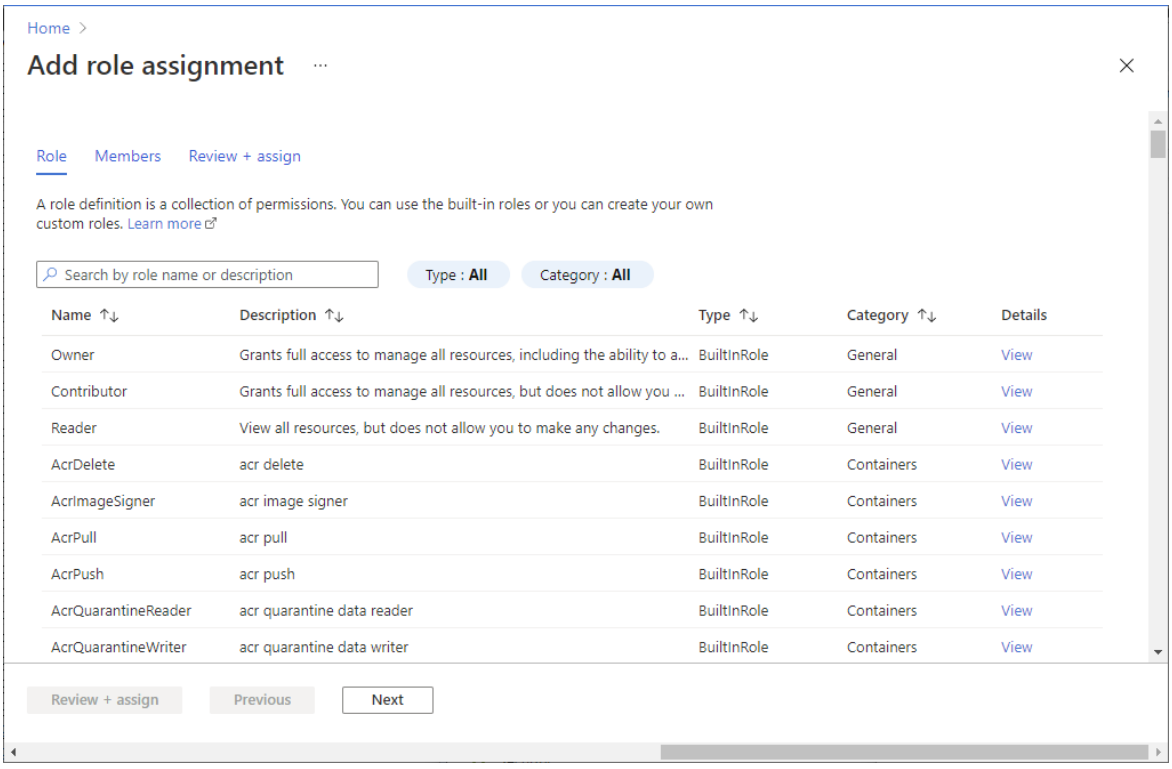
```
COPY INTO dbo.target_table
FROM 'https://myaccount.blob.core.windows.net/myblobcontainer/folder1/*.txt'
WITH (
    FILE_TYPE = 'CSV',
    CREDENTIAL = (IDENTITY = 'Managed Identity'),
)
```

D. Azure Active Directory Authentication

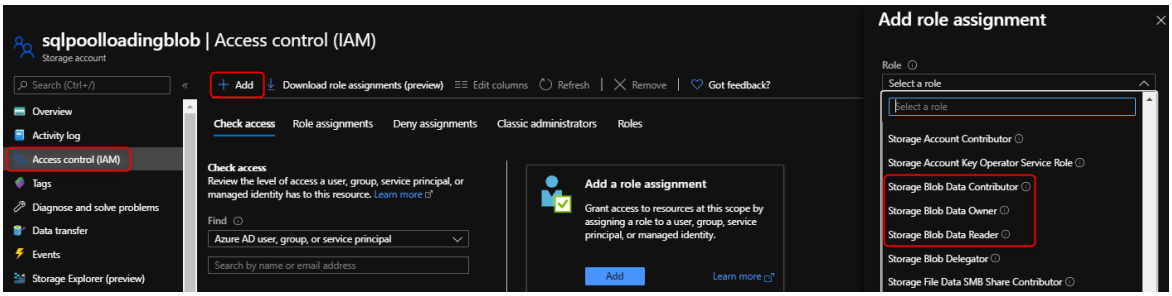
Steps

1. Under your storage account, select **Access control (IAM)**.
2. Select **Add** > **Add role assignment** to open the Add role assignment page.
3. Assign the following role. For detailed steps, see Assign Azure roles using the Azure portal.

SETTING	VALUE
Role	Storage Blob Data Owner, Contributor, or Reader
Assign access to	USER
Members	Azure AD user



> **IMPORTANT**
>
> Specify the **Storage Blob Data** Owner, Contributor, or Reader Azure role. These roles are different than the Azure built-in roles of Owner, Contributor, and Reader.



4. Configure Azure AD authentication. Refer to Configure and manage Azure AD authentication with Azure SQL.

5. Connect to your SQL pool using Active Directory where you can now run the COPY statement without specifying any credentials:

```
COPY INTO dbo.target_table
FROM 'https://myaccount.blob.core.windows.net/myblobcontainer/folder1/*.txt'
WITH (
    FILE_TYPE = 'CSV'
)
```

E. Service Principal Authentication

Steps

1. Create an Azure Active Directory application.
2. Get application ID.
3. Get the authentication key.
4. Get the V1 OAuth 2.0 token endpoint.
5. Assign read, write, and execution permissions to your Azure AD application on your storage account.
6. You can now run the COPY statement:

```
COPY INTO dbo.target_table
FROM 'https://myaccount.blob.core.windows.net/myblobcontainer/folder0/*.txt'
WITH (
    FILE_TYPE = 'CSV'
    ,CREDENTIAL=(IDENTITY= '<application_ID>@<OAuth_2.0_Token_EndPoint>' , SECRET=
'<authentication_key>')
    --CREDENTIAL should look something like this:
    --,CREDENTIAL=(IDENTITY= '92761aac-12a9-4ec3-89b8-
7149aef4c35b@https://login.microsoftonline.com/72f714bf-86f1-41af-91ab-2d7cd011db47/oauth2/token',
SECRET='juXi12sZ6gse]woKQNgqwSywYv]7A.M')
)
```

> **IMPORTANT**
>
> Use the **V1** version of the OAuth 2.0 token endpoint

Next steps

- Check the COPY statement article article for the detailed syntax
- Check the data loading overview article for loading best practices

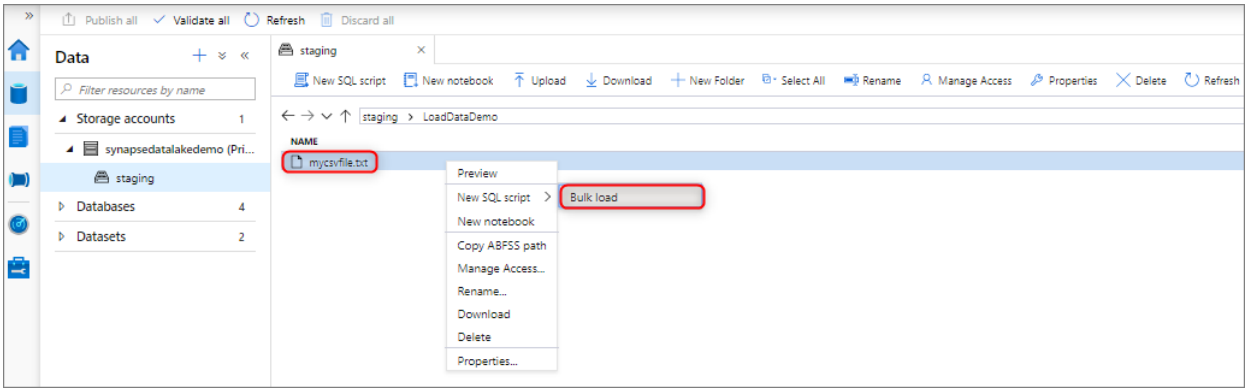
Quickstart: Bulk loading with Synapse Studio

8/19/2022 • 2 minutes to read • Edit Online

Loading data is easy with the Bulk Load wizard in Synapse Studio. Synapse Studio is a feature of Azure Synapse Analytics. The Bulk Load wizard guides you through creating a T-SQL script with the COPY statement to bulk load data into a dedicated SQL pool.

Entry points to the Bulk Load wizard

You can bulk load data by right-clicking the following area within Synapse Studio: a file or folder from an Azure storage account that's attached to your workspace.

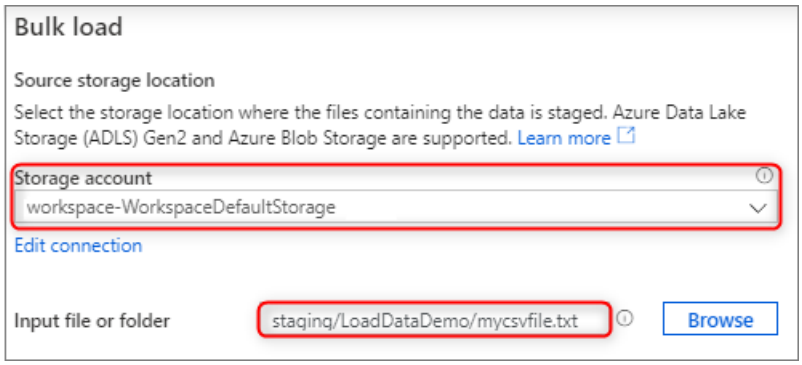


Prerequisites

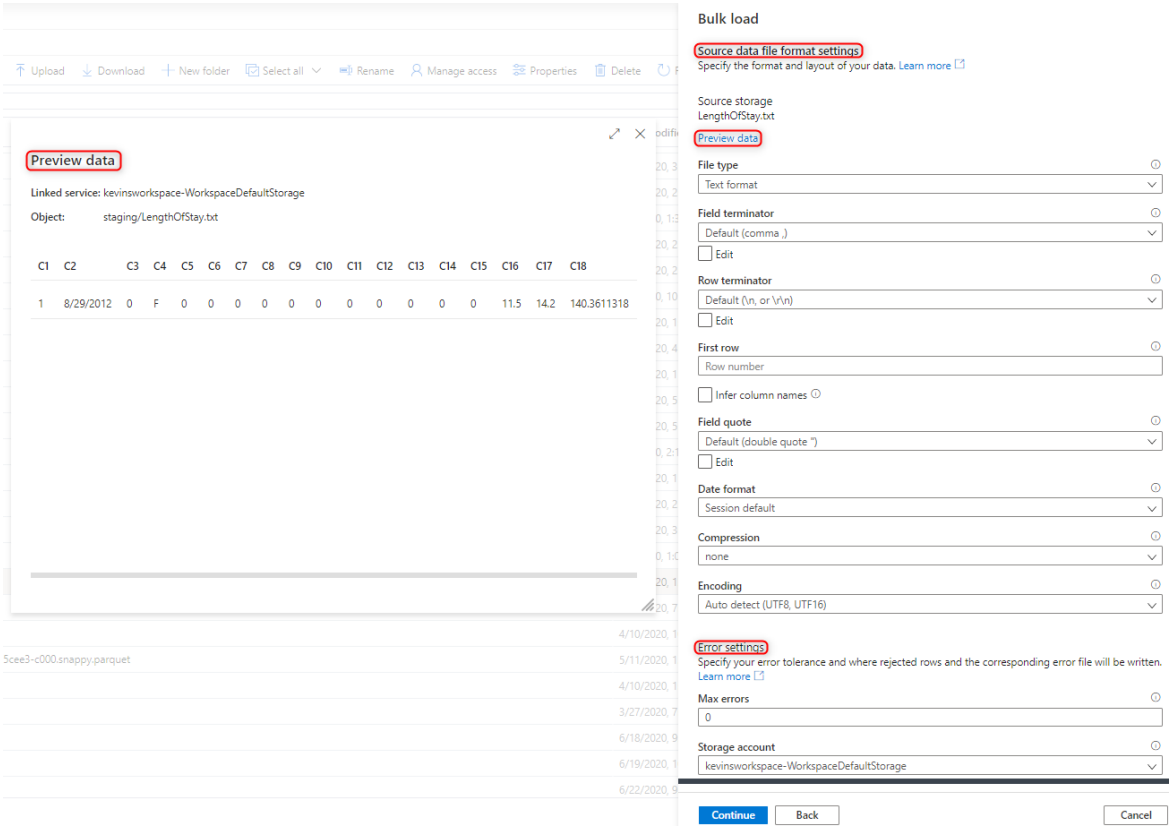
- The wizard generates a COPY statement, which uses Azure Active Directory (Azure AD) pass-through for authentication. Your Azure AD user must have access to the workspace with at least the Storage Blob Data Contributor Azure role for the Azure Data Lake Storage Gen2 account.
- You must have the required permissions to use the COPY statement and Create Table permissions if you're creating a new table to load to.
- The linked service associated with the Data Lake Storage Gen2 account *must have access to the file or folder* to load. For example, if the authentication mechanism for the linked service is a managed identity, the workspace's managed identity must have at least Storage Blob Data Reader permission on the storage account.
- If a virtual network is enabled on your workspace, make sure the integrated runtime associated with the Data Lake Storage Gen2 account's linked services for the source data and error file location has interactive authoring enabled. Interactive authoring is required for autoschema detection, previewing the source file contents, and browsing Data Lake Storage Gen2 storage accounts within the wizard.

Steps

1. On the **Source storage location** panel, select the storage account and the file or folder you're loading from. The wizard automatically tries to detect Parquet files and delimited text (CSV) files, including mapping the source fields from the file to the appropriate target SQL data types.



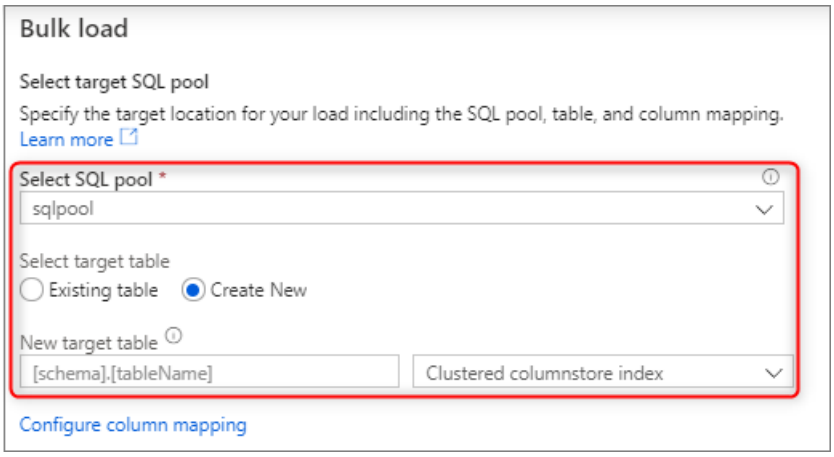
2. Select the file format settings, including your error settings for when there are rejected rows during the bulk load process. You can also select **Preview data** to see how the COPY statement will parse the file to help you configure the file format settings. Select **Preview data** every time you change a file format setting, to see how the COPY statement will parse the file with the updated setting.



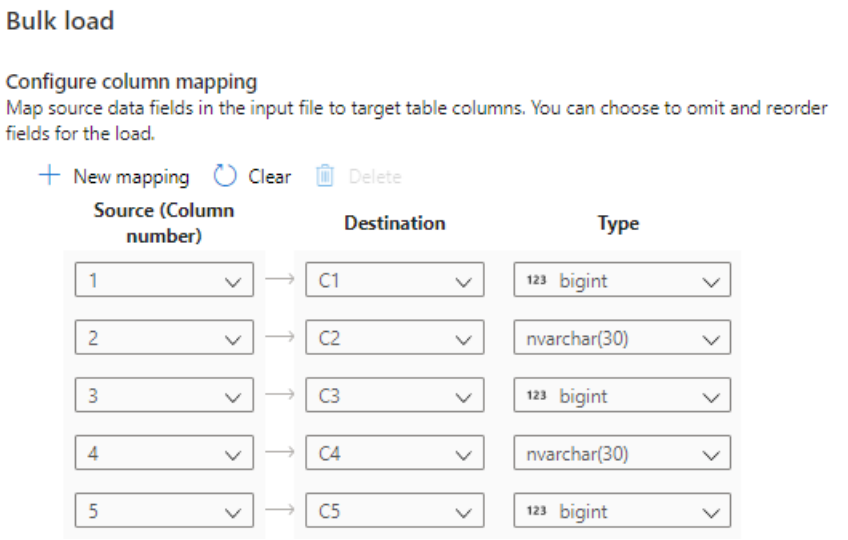
> **NOTE**
>
> - The Bulk Load wizard doesn't support previewing the data with multi-character field terminators. When you specify a multi-character field terminator, the wizard will preview the data within a single column.
> - When you select **Infer column names**, the Bulk Load wizard will parse the columns names from the first row specified by the **First row** field. The Bulk Load wizard will automatically increment the `FIRSTROW` value in the COPY statement by 1 to ignore this header row.
> - Specifying multi-character row terminators is supported in the COPY statement. However, the Bulk Load wizard doesn't support it and will throw an error.

3. Select the dedicated SQL pool that you're using to load, including whether the load will be for an existing

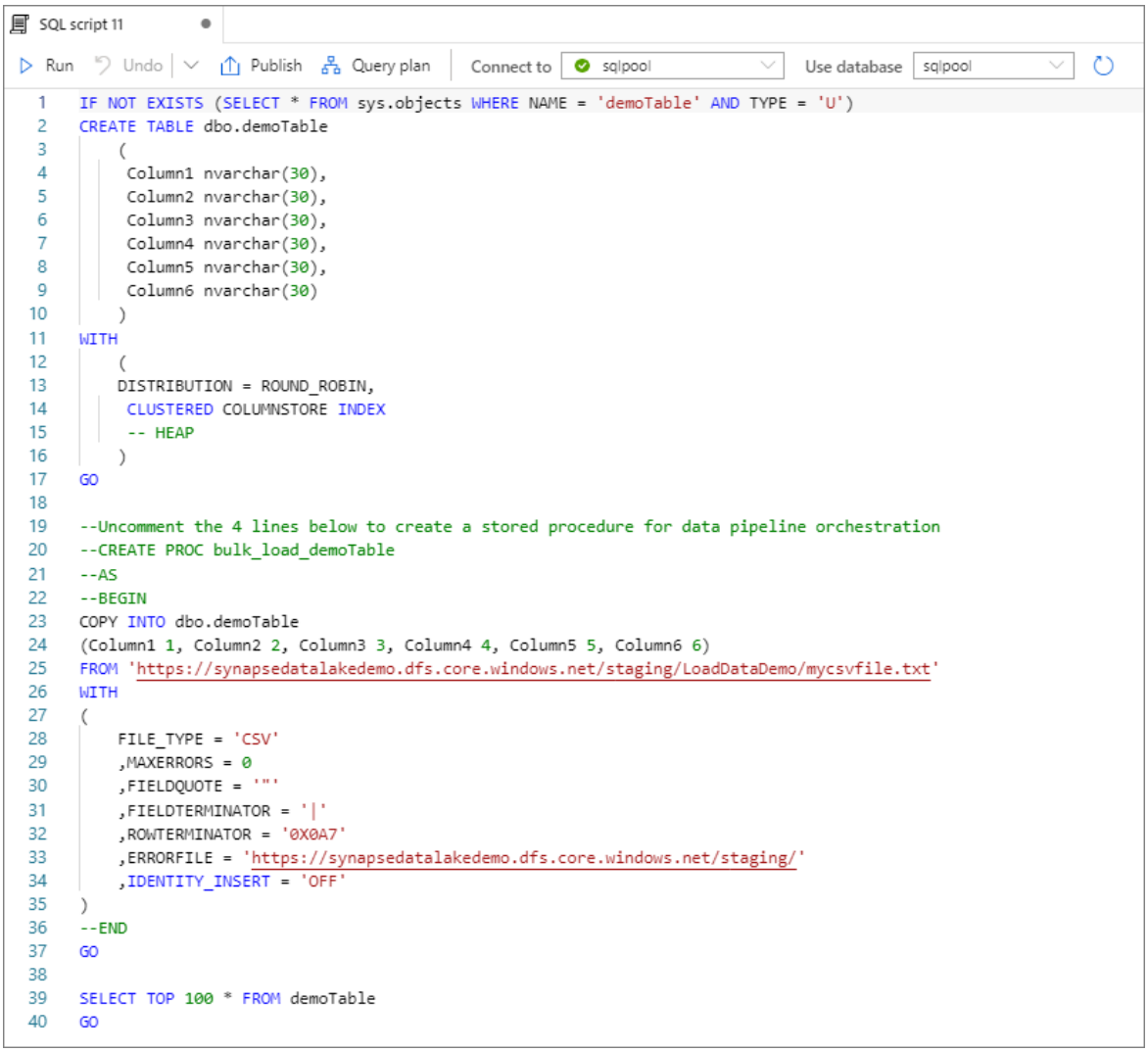
table or a new table.



4. Select **Configure column mapping** to make sure you have the appropriate column mapping. Note columns names will be detected automatically if you enabled **Infer column names**. For new tables, configuring the column mapping is critical for updating the target column data types.



5. Select **Open script**. A T-SQL script is generated with the COPY statement to load from your data lake.



Next steps

- Check the COPY statement article for more information on COPY capabilities.
- Check the data loading overview article for information about using an extract, transform, and load (ETL) process.

Quickstart: Create a dedicated SQL pool workload classifier using the Azure portal

8/19/2022 • 3 minutes to read • Edit Online

In this quickstart, you will create a workload classifier for assigning queries to a workload group. The classifier will assign requests from the `ELTLogin` SQL user to the `DataLoads` workload group. Follow the Quickstart: Configure workload isolation tutorial to create the `DataLoads` workload group. This tutorial will create a workload classifier with the WLM_LABEL option to help further classify requests correctly. The classifier will assign `HIGH` workload importance to these requests as well.

If you don't have an Azure subscription, create a free account before you begin.

Sign in to the Azure portal

Sign in to the Azure portal.

> **NOTE**
> Creating a dedicated SQL pool instance in Azure Synapse Analytics may result in a new billable service. For more information, see Azure Synapse Analytics pricing.

Prerequisites

This quickstart assumes you already have a dedicated SQL pool instance that you have CONTROL DATABASE permissions. If you need to create one, use Create and Connect - portal to create a dedicated SQL pool called **mySampleDataWarehouse**.

A workload group `DataLoads` exists. See the Quickstart: Configure workload isolation tutorial to create the workload group.

> **IMPORTANT**
> Your dedicated SQL pool must be online to configure workload management.

Create a login for ELTLogin

Create a SQL Server authentication login in the `master` database using CREATE LOGIN for `ELTLogin`.

```
IF NOT EXISTS (SELECT * FROM sys.sql_logins WHERE name = 'ELTLogin')
BEGIN
CREATE LOGIN [ELTLogin] WITH PASSWORD='<strongpassword>'
END
;
```

Create user and grant permissions

After the login is created, a user needs to be created in the database. Use CREATE USER to create the SQL user

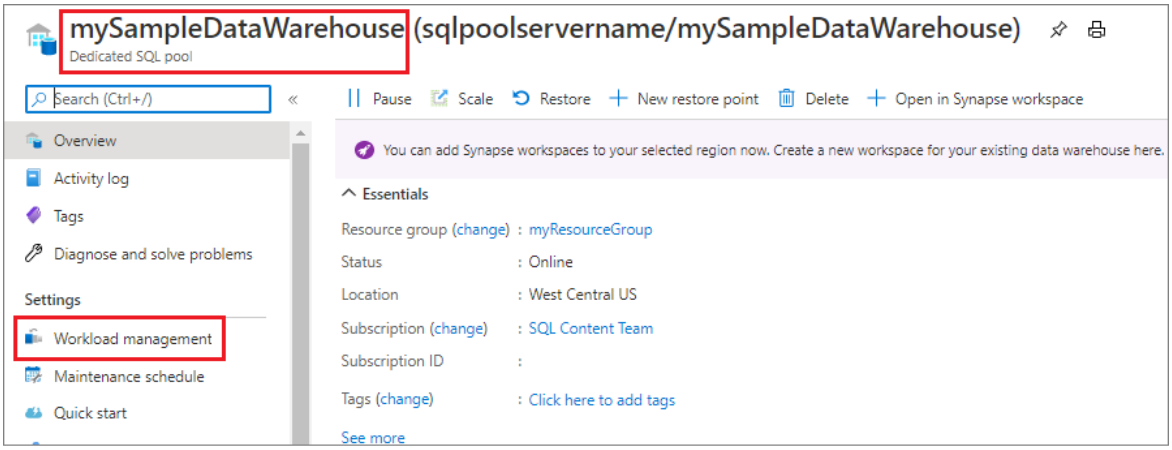
`ELTRole` in the **mySampleDataWarehouse**. Since we will test the classification during this tutorial, grant `ELTLogin` permissions to **mySampleDataWarehouse**.

```
IF NOT EXISTS (SELECT * FROM sys.database_principals WHERE name = 'ELTLogin')
BEGIN
CREATE USER [ELTLogin] FOR LOGIN [ELTLogin]
GRANT CONTROL ON DATABASE::mySampleDataWarehouse TO ELTLogin
END
;
```

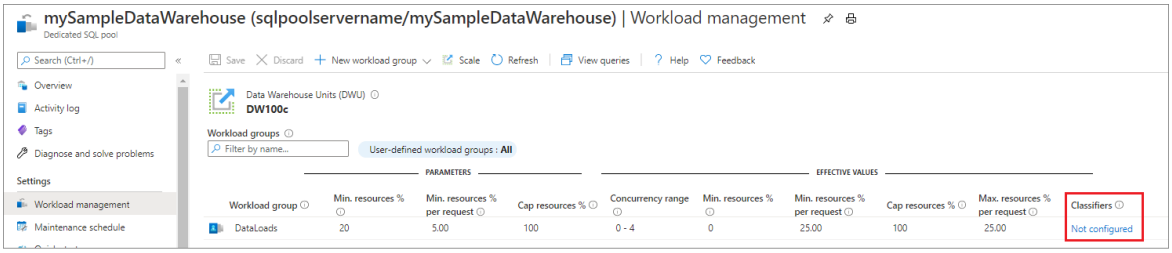
Configure workload classification

Classification allows you to route requests, based on a set of rules, to a workload group. In the Quickstart: Configure workload isolation tutorial we created the `DataLoads` workload group. Now you will create a workload classifier to route queries to the `DataLoads` workload group.

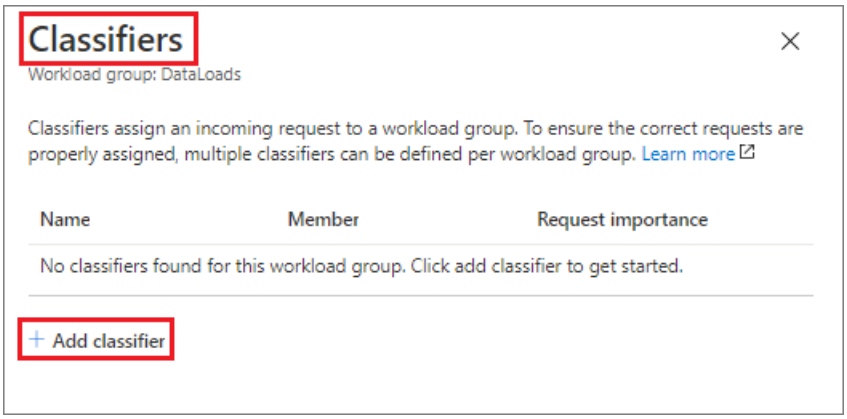
1. Navigate to your **mySampleDataWarehouse** dedicated SQL pool page.
2. Select **Workload management**.



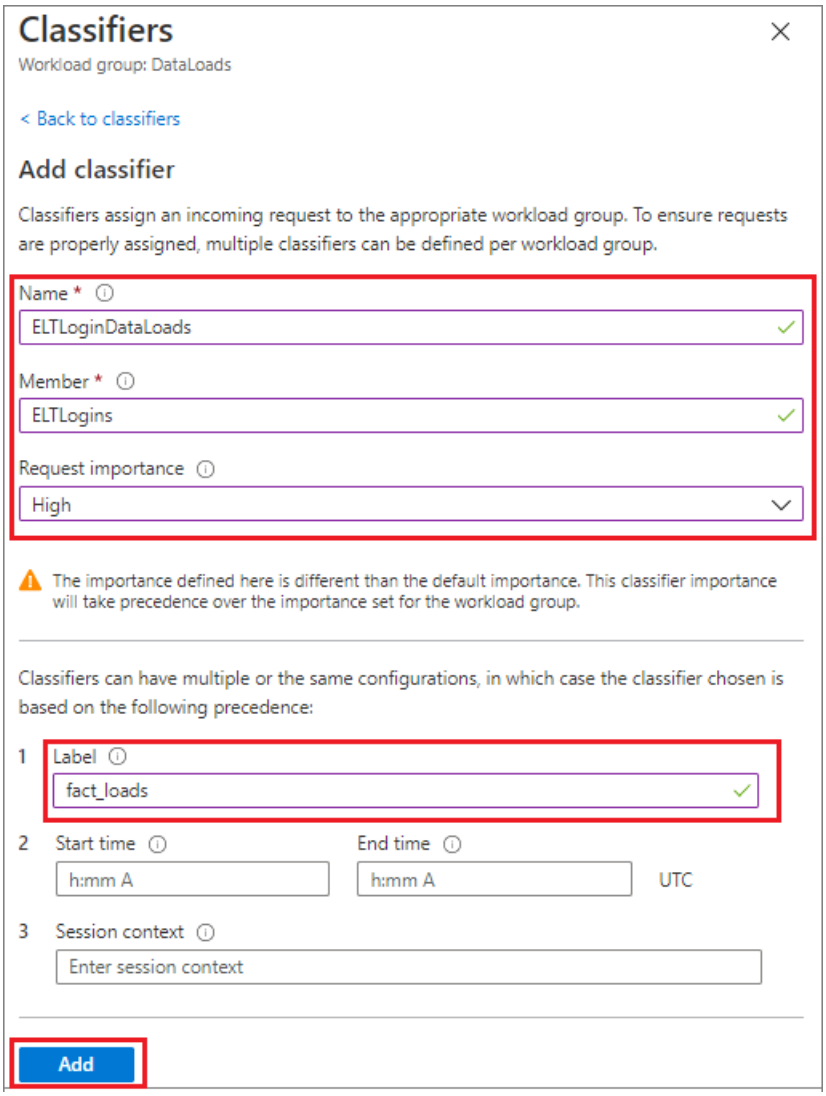
3. Select **Settings & classifiers** on the right-hand side of the `DataLoads` workload group.



4. Select **Not configured** under the Classifiers column.
5. Select **+ Add classifier**.



6. Enter `ELTLoginDataLoads` for **Name**.
7. Enter `ELTLogin` for **Member**.
8. Choose `High` for **Request Importance**. *Optional*, normal importance is default.
9. Enter `fact_loads` for **Label**.
10. Select **Add**.
11. Select **Save**.



Verify and test classification

Check the sys.workload_management_workload_classifiers catalog view to verify existence of the

`ELTLoginDataLoads` classifier.

```
SELECT * FROM sys.workload_management_workload_classifiers WHERE name = 'ELTLoginDataLoads'
```

Check the sys.workload_management_workload_classifier_details catalog view to verify classifier details.

```
SELECT c.[name], c.group_name, c.importance, cd.classifier_type, cd.classifier_value
  FROM sys.workload_management_workload_classifiers c
  JOIN sys.workload_management_workload_classifier_details cd
    ON cd.classifier_id = c.classifier_id
  WHERE c.name = 'ELTLoginDataLoads'
```

Run the following statements to test classification. Ensure you are connected as `ELTLogin` and `Label` is used in query.

```
CREATE TABLE factstaging (ColA int)
INSERT INTO factstaging VALUES(0)
INSERT INTO factstaging VALUES(1)
INSERT INTO factstaging VALUES(2)
GO

CREATE TABLE testclassifierfact WITH (DISTRIBUTION = ROUND_ROBIN)
AS
SELECT * FROM factstaging
OPTION (LABEL='fact_loads')
```

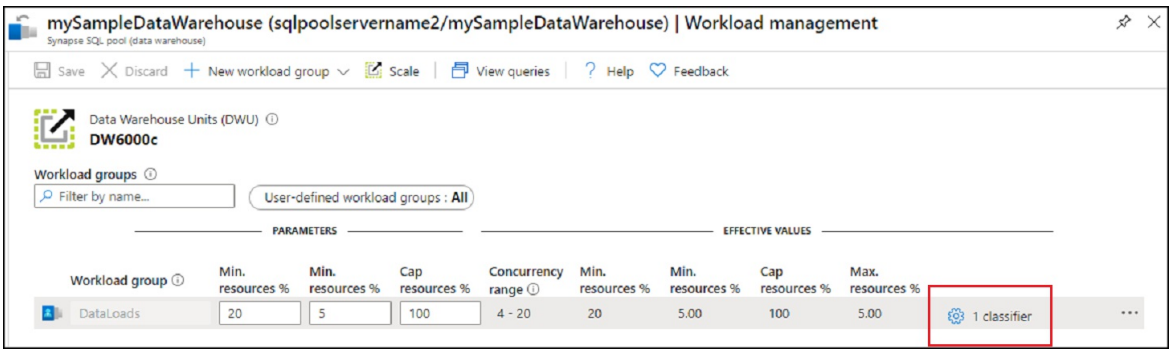
Verify the `CREATE TABLE` statement classified to the `DataLoads` workload group using the `ELTLoginDataLoads` workload classifier.

```
SELECT TOP 1 request_id, classifier_name, group_name, resource_allocation_percentage, submit_time, [status],
[label], command
FROM sys.dm_pdw_exec_requests
WHERE [label] = 'fact_loads'
ORDER BY submit_time DESC
```

Clean up resources

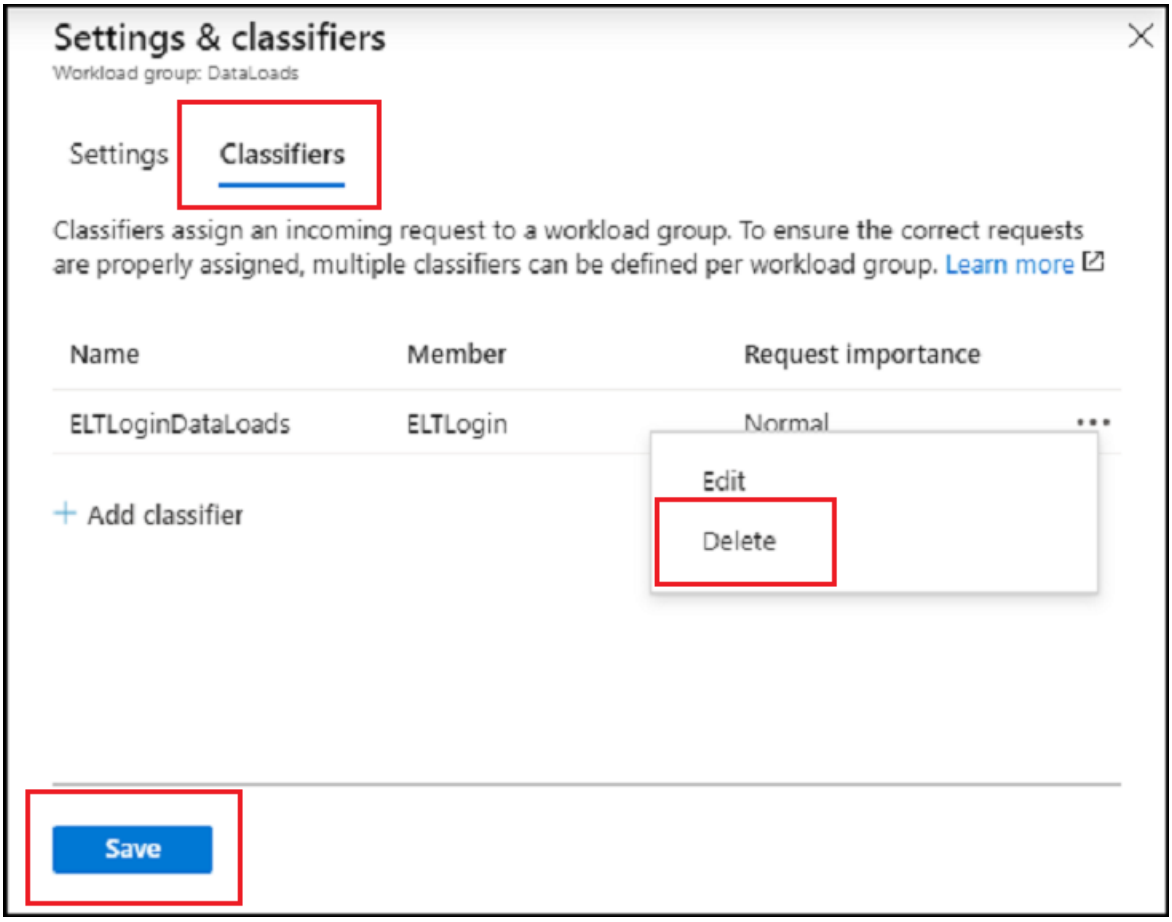
To delete the `ELTLoginDataLoads` workload classifier created in this tutorial:

1. Click on **1 Classifier** on the right-hand side of the `DataLoads` workload group.



2. Click on **Classifiers**.
3. Click on the `...` to the right of the `ELTLoginDataLoads` workload classifier.
4. Click on **Delete**.

5. Click on **Save**.

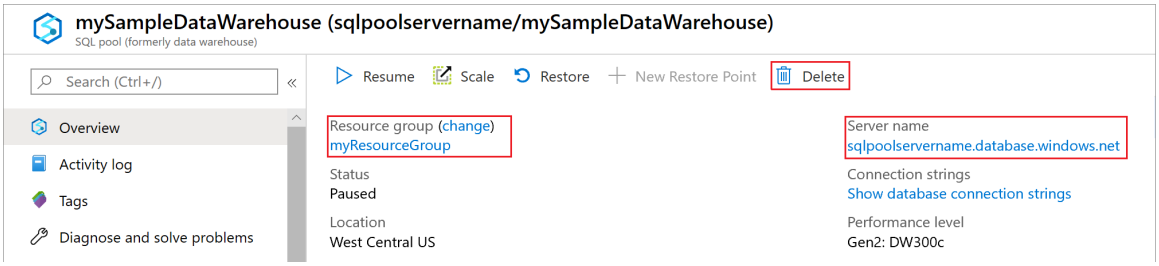


You're being charged for data warehouse units and data stored in your dedicated SQL pool. These compute and storage resources are billed separately.

- If you want to keep the data in storage, you can pause compute when you aren't using your dedicated SQL pool. By pausing compute, you're only charged for data storage. When you're ready to work with the data, resume compute.
- If you want to remove future charges, you can delete your dedicated SQL pool.

Follow these steps to clean up resources.

1. Sign in to the Azure portal, select your dedicated SQL pool.



2. To pause compute, select the **Pause** button. When the dedicated SQL pool is paused, you see a **Start** button. To resume compute, select **Start**.

3. To remove your dedicated SQL pool so you're not charged for compute or storage, select **Delete**.

Next steps

Monitor your workload using the Azure portal monitoring metrics. See Manage and monitor Workload Management for details.

Quickstart: Create a workload classifier using T-SQL

8/19/2022 • 2 minutes to read • Edit Online

In this quickstart, you'll quickly create a workload classifier with high importance for the CEO of your organization. This workload classifier will allow CEO queries to take precedence over other queries with lower importance in the queue.

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> Creating a dedicated SQL pool instance in Azure Synapse Analytics may result in a new billable service. For more information, see Azure Synapse Analytics pricing.

Prerequisites

This quickstart assumes you have already provisioned a dedicated SQL pool in Azure Synapse Analytics and that you have CONTROL DATABASE permissions. If you need to create one, use Create and Connect - portal to create a dedicated SQL pool called **mySampleDataWarehouse**.

Sign in to the Azure portal

Sign in to the Azure portal.

Create login for TheCEO

Create a SQL Server authentication login in the `master` database using CREATE LOGIN for 'TheCEO'.

```
IF NOT EXISTS (SELECT * FROM sys.sql_logins WHERE name = 'TheCEO')
BEGIN
CREATE LOGIN [TheCEO] WITH PASSWORD='<strongpassword>'
END
;
```

Create user

Create user, "TheCEO", in mySampleDataWarehouse

```
IF NOT EXISTS (SELECT * FROM sys.database_principals WHERE name = 'THECEO')
BEGIN
CREATE USER [TheCEO] FOR LOGIN [TheCEO]
END
;
```

Create a workload classifier

Create a workload classifier for "TheCEO" with high importance.

```
DROP WORKLOAD CLASSIFIER [wgcTheCEO];
CREATE WORKLOAD CLASSIFIER [wgcTheCEO]
WITH (WORKLOAD_GROUP = 'xlargerc'
      ,MEMBERNAME = 'TheCEO'
      ,IMPORTANCE = HIGH);
```

View existing classifiers

```
SELECT * FROM sys.workload_management_workload_classifiers
```

Clean up resources

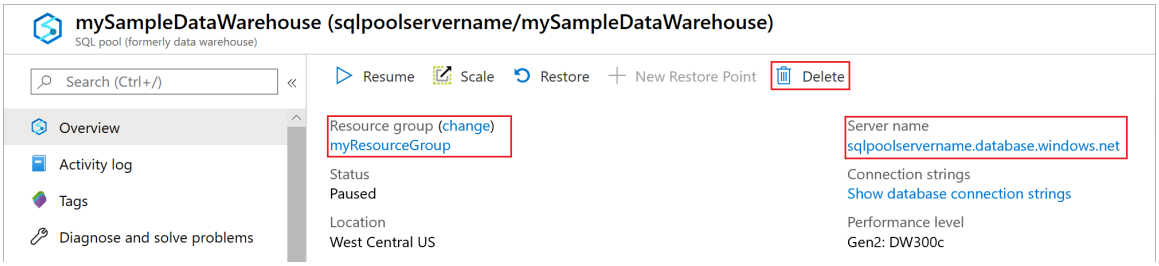
```
DROP WORKLOAD CLASSIFIER [wgcTheCEO]
DROP USER [TheCEO]
;
```

You're being charged for data warehouse units and data stored in your dedicated SQL pool. These compute and storage resources are billed separately.

- If you want to keep the data in storage, you can pause compute when you aren't using the dedicated SQL pool. By pausing compute, you're only charged for data storage. When you're ready to work with the data, resume compute.
- If you want to remove future charges, you can delete the dedicated SQL pool.

Follow these steps to clean up resources.

1. Sign in to the Azure portal, select your dedicated SQL pool.



2. To pause compute, select the **Pause** button. When the dedicated SQL pool is paused, you see a **Start** button. To resume compute, select **Start**.

3. To remove the dedicated SQL pool so you're not charged for compute or storage, select **Delete**.

Next steps

- You've now created a workload classifier. Run a few queries as TheCEO to see how they perform. See sys.dm_pdw_exec_requests to view queries and the importance assigned.
- For more information about dedicated SQL pool workload management, see Workload Importance and Workload Classification.
- See the how-to articles to Configure Workload Importance and how to Manage and monitor Workload Management.

Quickstart: Configure dedicated SQL pool workload isolation using a workload group in the Azure portal

8/19/2022 • 2 minutes to read • Edit Online

In this quickstart, you will configure workload isolation by creating a workload group for reserving resources. For purposes of this tutorial, we will create the workload group for data loading called `DataLoads`. The workload group will reserve 20% of the system resources. With 20% isolation for data loads, they are guaranteed resources that allow them to hit SLAs. After creating the workload group, create a workload classifier to assign queries to this workload group.

If you don't have an Azure subscription, create a free account before you begin.

Sign in to the Azure portal

Sign in to the Azure portal.

> **NOTE**
> Creating a dedicated SQL pool instance in Azure Synapse Analytics may result in a new billable service. For more information, see Azure Synapse Analytics pricing.

Prerequisites

This quickstart assumes you already have a dedicated SQL pool instance in Synapse SQL and that you have CONTROL DATABASE permissions. If you need to create one, use Quickstart: Create dedicated SQL pool - portal to create a data warehouse called **mySampleDataWarehouse**.

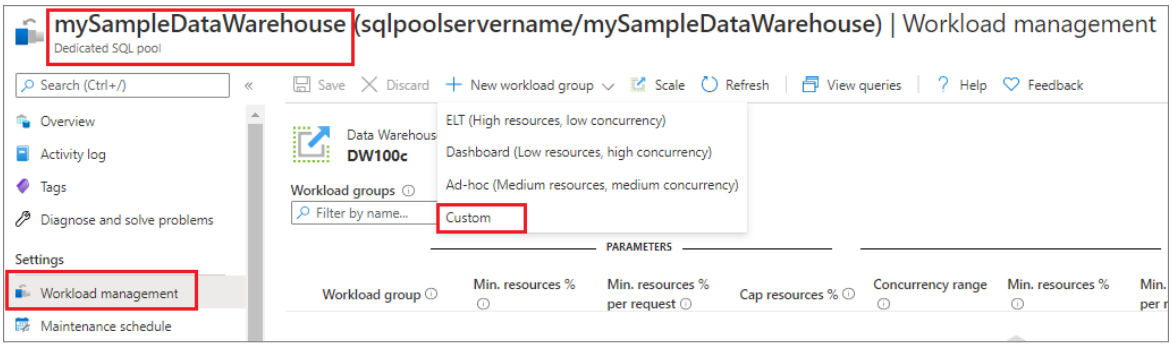
> **IMPORTANT**
> Your dedicated SQL pool must be online to configure workload management.

Configure workload isolation

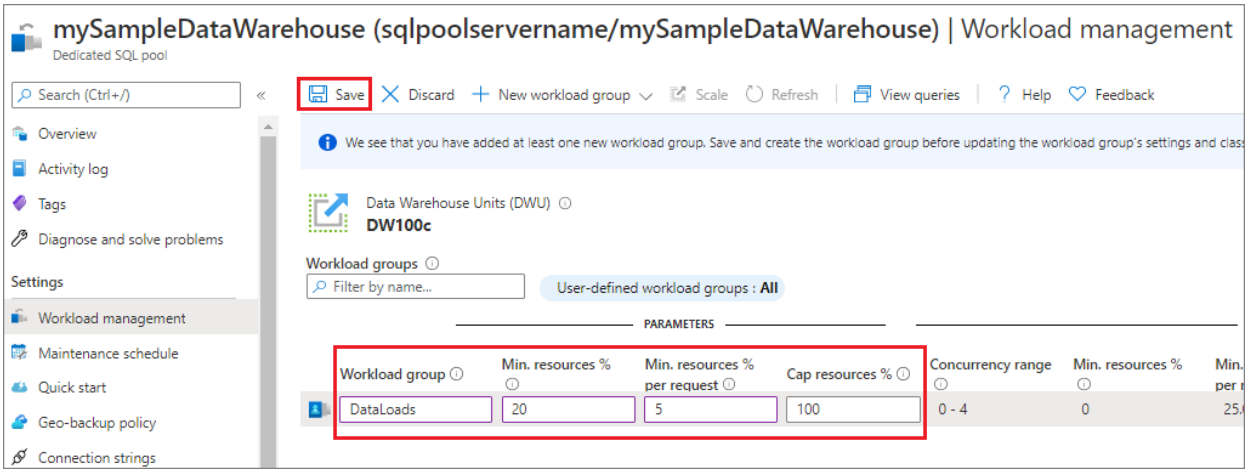
Dedicated SQL pool resources can be isolated and reserved for specific workloads by creating workload groups. See the Workload Isolation concept documentation for more details on how workload groups help you achieve managing your workload. The Create and Connect - portal quickstart created **mySampleDataWarehouse** and initialized it at DW100c. The following steps create a workload group in **mySampleDataWarehouse**.

To create a workload group with 20% isolation:

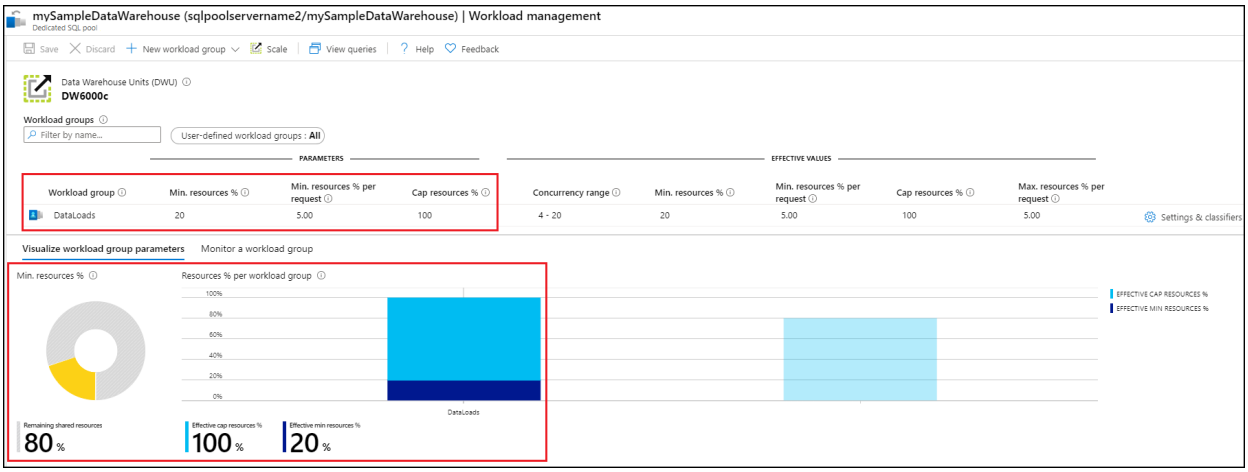
1. Navigate to your **mySampleDataWarehouse** dedicated SQL pool page..
2. Select **Workload management**.
3. Select **New workload group**.
4. Select **Custom**.



5. Enter `DataLoads` for the **Workload group**.
6. Enter `20` for **Min. resources %**.
7. Enter `5` for **Min. resources % per request**.
8. Enter `100` for **Cap resources %**.
9. Enter **Save**.



A portal notification appears when the workload group is created. The workload group resources are displayed in the charts below the configured values.



Clean up resources

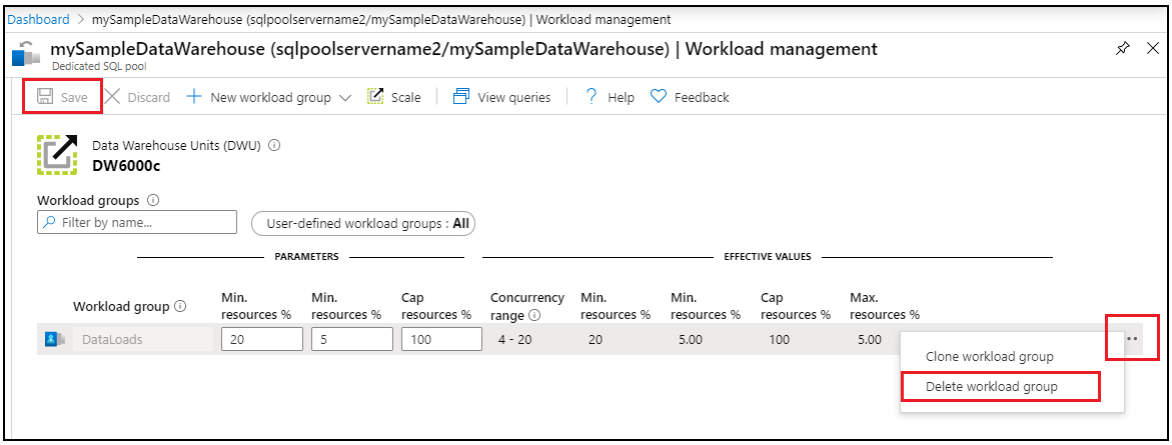
To delete the `DataLoads` workload group created in this tutorial:

1. Click on the `...` to the right of the `DataLoads` workload group.

2. Click on **Delete workload group**.

3. Click on **Yes** when prompted to confirm deletion of the workload group.

4. Click on **Save**.

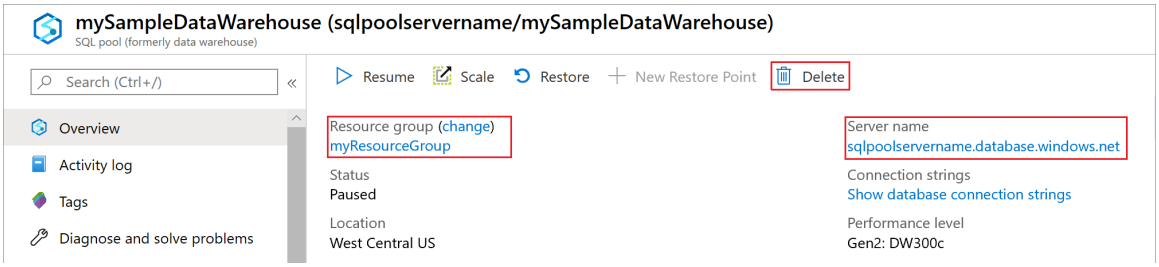


You're being charged for data warehouse units and data stored in your data warehouse. These compute and storage resources are billed separately.

- If you want to keep the data in storage, you can pause compute when you aren't using the data warehouse. By pausing compute, you're only charged for data storage. When you're ready to work with the data, resume compute.
- If you want to remove future charges, you can delete the data warehouse.

Follow these steps to clean up resources.

1. Sign in to the Azure portal, select your dedicated SQL pool.



2. To pause compute, select the **Pause** button. When the data warehouse is paused, you see a **Start** button. To resume compute, select **Start**.

3. To remove the data warehouse so you're not charged for compute or storage, select **Delete**.

Next steps

To use the `DataLoads` workload group, a workload classifier needs to be created to route requests to the workload group. Continue to the create workload classifier tutorial to create a workload classifier for `DataLoads`.

See also

See the how-to article Manage and monitor Workload Management for details on how to monitor workloads for workload management.

Quickstart: Configure workload isolation in a dedicated SQL pool using T-SQL

8/19/2022 • 2 minutes to read • Edit Online

In this quickstart, you'll quickly create a workload group and classifier for reserving resources for data loading. The workload group will allocate 20% of the system resources to the data loads. The workload classifier will assign requests to the data loads workload group. With 20% isolation for data loads, they are guaranteed resources to hit SLAs.

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> Creating a Synapse SQL instance in Azure Synapse Analytics may result in a new billable service. For more information, see Azure Synapse Analytics pricing.

Prerequisites

This quickstart assumes you already have a Synapse SQL instance in Azure Synapse and that you have CONTROL DATABASE permissions. If you need to create one, use Create and Connect - portal to create a dedicated SQL pool called **mySampleDataWarehouse**.

Create login for DataLoads

Create a SQL Server authentication login in the `master` database using CREATE LOGIN for 'ELTLogin'.

```
IF NOT EXISTS (SELECT * FROM sys.sql_logins WHERE name = 'ELTLogin')
BEGIN
CREATE LOGIN [ELTLogin] WITH PASSWORD='<strongpassword>'
END
;
```

Create user

Create user, "ELTLogin", in mySampleDataWarehouse

```
IF NOT EXISTS (SELECT * FROM sys.database_principals WHERE name = 'ELTLogin')
BEGIN
CREATE USER [ELTLogin] FOR LOGIN [ELTLogin]
END
;
```

Create a workload group

Create a workload group for DataLoads with 20% isolation.

```
CREATE WORKLOAD GROUP DataLoads
WITH ( MIN_PERCENTAGE_RESOURCE = 20
      ,CAP_PERCENTAGE_RESOURCE = 100
      ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 5)
;
```

Create a workload classifier

Create a workload classifier to map ELTLogin to the DataLoads workload group.

```
CREATE WORKLOAD CLASSIFIER [wgcELTLogin]
WITH (WORKLOAD_GROUP = 'DataLoads'
      ,MEMBERNAME = 'ELTLogin')
;
```

View existing workload groups and classifiers and run-time values

```
--Workload groups
SELECT * FROM
sys.workload_management_workload_groups

--Workload classifiers
SELECT * FROM
sys.workload_management_workload_classifiers

--Run-time values
SELECT * FROM
sys.dm_workload_management_workload_groups_stats
```

Clean up resources

```
DROP WORKLOAD CLASSIFIER [wgcELTLogin]
DROP WORKLOAD GROUP [DataLoads]
DROP USER [ELTLogin]
;
```

You're being charged for data warehouse units and data stored in your dedicated SQL pool. These compute and storage resources are billed separately.

- If you want to keep the data in storage, you can pause compute when you aren't using the dedicated SQL pool. By pausing compute, you're only charged for data storage. When you're ready to work with the data, resume compute.
- If you want to remove future charges, you can delete the dedicated SQL pool.

Next steps

- You've now created a workload group. Run a few queries as ELTLogin to see how they perform. See sys.dm_pdw_exec_requests to view queries and the workload group assigned.
- For more information about Synapse SQL workload management, see Workload Management and Workload Isolation.

Tutorial: Explore and Analyze data lakes with serverless SQL pool

8/19/2022 • 4 minutes to read • Edit Online

In this tutorial, you learn how to perform exploratory data analysis. You combine different Azure Open Datasets using serverless SQL pool. You then visualize the results in Synapse Studio for Azure Synapse Analytics.

The `OPENROWSET(BULK...)` function allows you to access files in Azure Storage. `[OPENROWSET](develop-openrowset.md)` reads content of a remote data source, such as a file, and returns the content as a set of rows.

Automatic schema inference

Since data is stored in the Parquet file format, automatic schema inference is available. You can query the data without listing the data types of all columns in the files. You also can use the virtual column mechanism and the `filepath` function to filter out a certain subset of files.

> **NOTE**
>
> The default collation is `SQL_Latin1_General_CP1_CI_ASIf`. For a non-default collation, take into account case sensitivity.
>
> If you create a database with case sensitive collation when you specify columns, make sure to use correct name of the column.
>
> A column name `tpepPickupDateTime` would be correct while `tpeppickupdatetime` wouldn't work in a non-default collation.

This tutorial uses a dataset about New York City (NYC) Taxi:

- Pick-up and drop-off dates and times
- Pick-up and drop-off locations
- Trip distances
- Itemized fares
- Rate types
- Payment types
- Driver-reported passenger counts

To get familiar with the NYC Taxi data, run the following query:

```
SELECT TOP 100 * FROM
    OPENROWSET(
        BULK
'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/puYear=*/puMonth=*/*.parquet',
        FORMAT='PARQUET'
    ) AS [nyc]
```

Similarly, you can query the Public Holidays dataset by using the following query:

```
SELECT TOP 100 * FROM
    OPENROWSET(
        BULK 'https://azureopendatastorage.blob.core.windows.net/holidaydatacontainer/Processed/*.parquet',
        FORMAT='PARQUET'
    ) AS [holidays]
```

You can also query the Weather Data dataset by using the following query:

```
SELECT
    TOP 100 *
FROM
    OPENROWSET(
        BULK
'https://azureopendatastorage.blob.core.windows.net/isdweatherdatacontainer/ISDWeather/year=*/month=*/*.parq
uet',
        FORMAT='PARQUET'
    ) AS [weather]
```

You can learn more about the meaning of the individual columns in the descriptions of the data sets:

- NYC Taxi
- Public Holidays
- Weather Data

Time series, seasonality, and outlier analysis

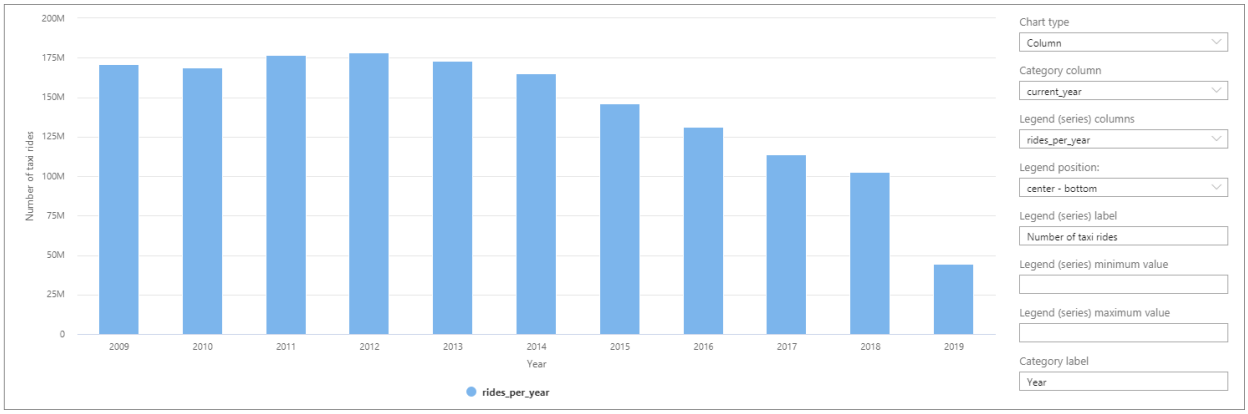
You can summarize the yearly number of taxi rides by using the following query:

```
SELECT
    YEAR(tpepPickupDateTime) AS current_year,
    COUNT(*) AS rides_per_year
FROM
    OPENROWSET(
        BULK
'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/puYear=*/puMonth=*/*.parquet',
        FORMAT='PARQUET'
    ) AS [nyc]
WHERE nyc.filepath(1) >= '2009' AND nyc.filepath(1) <= '2019'
GROUP BY YEAR(tpepPickupDateTime)
ORDER BY 1 ASC
```

The following snippet shows the result for the yearly number of taxi rides:

CURRENT_YEAR	RIDES_PER_YEAR
2009	170896844
2010	169001153
2011	176897208
2012	178544324
2013	173179759
2014	165114361
2015	146112989
2016	131165043
2017	113496933
2018	102803387
2019	44458570

The data can be visualized in Synapse Studio by switching from the **Table** to the **Chart** view. You can choose among different chart types, such as **Area**, **Bar**, **Column**, **Line**, **Pie**, and **Scatter**. In this case, plot the **Column** chart with the **Category** column set to **current_year**:



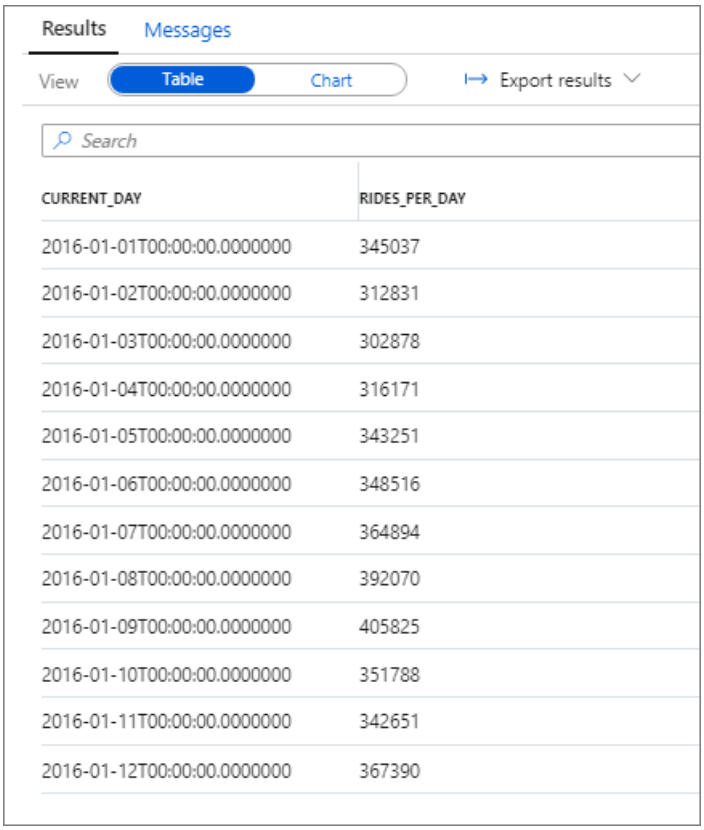
From this visualization, you can see a trend of decreasing ride numbers over the years. Presumably, this decrease is due to the recent increased popularity of ride-sharing companies.

> **NOTE**
>
> At the time of writing this tutorial, data for 2019 is incomplete. As a result, there's a huge drop in the number of rides for that year.

You can focus the analysis on a single year, for example, 2016. The following query returns the daily number of rides during that year:

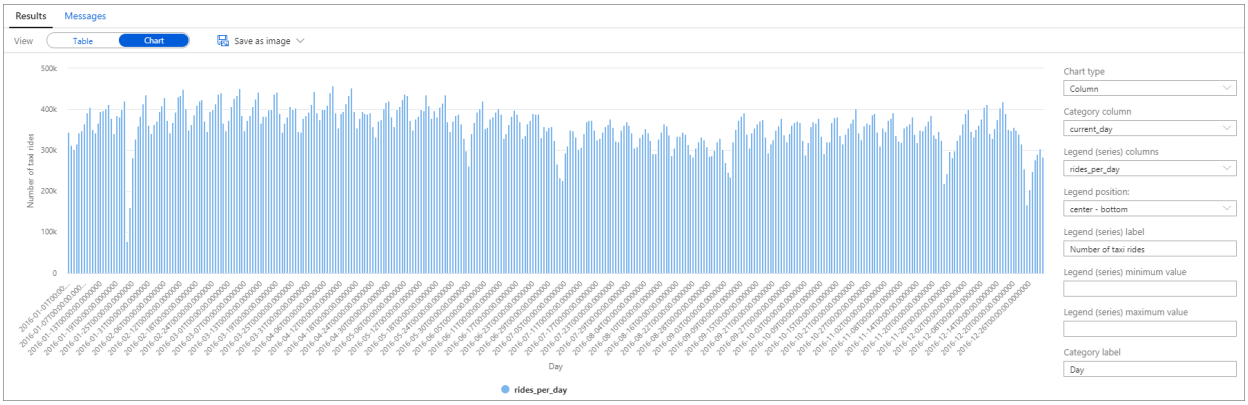
```
SELECT
    CAST([tpepPickupDateTime] AS DATE) AS [current_day],
    COUNT(*) as rides_per_day
FROM
    OPENROWSET(
        BULK
'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/puYear=*/puMonth=*/*.parquet',
        FORMAT='PARQUET'
    ) AS [nyc]
WHERE nyc.filepath(1) = '2016'
GROUP BY CAST([tpepPickupDateTime] AS DATE)
ORDER BY 1 ASC
```

The following snippet shows the result for this query:



CURRENT_DAY	RIDES_PER_DAY
2016-01-01T00:00:00.0000000	345037
2016-01-02T00:00:00.0000000	312831
2016-01-03T00:00:00.0000000	302878
2016-01-04T00:00:00.0000000	316171
2016-01-05T00:00:00.0000000	343251
2016-01-06T00:00:00.0000000	348516
2016-01-07T00:00:00.0000000	364894
2016-01-08T00:00:00.0000000	392070
2016-01-09T00:00:00.0000000	405825
2016-01-10T00:00:00.0000000	351788
2016-01-11T00:00:00.0000000	342651
2016-01-12T00:00:00.0000000	367390

Again, you can visualize data by plotting the **Column** chart with the **Category** column set to **current_day** and the **Legend (series)** column set to **rides_per_day**.



From the plot chart, you can see there's a weekly pattern, with Saturdays as the peak day. During Summer months, there are fewer taxi rides because of vacations. Also, notice some significant drops in the number of taxi rides without a clear pattern of when and why they occur.

Next, see if the drop in rides correlates with public holidays. Check if there's a correlation by joining the NYC Taxi

rides dataset with the Public Holidays dataset:

```
WITH taxi_rides AS (
SELECT
    CAST([tpepPickupDateTime] AS DATE) AS [current_day],
    COUNT(*) as rides_per_day
FROM
    OPENROWSET(
        BULK
'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/puYear=*/puMonth=*/*.parquet',
        FORMAT='PARQUET'
    ) AS [nyc]
WHERE nyc.filepath(1) = '2016'
GROUP BY CAST([tpepPickupDateTime] AS DATE)
),
public_holidays AS (
SELECT
    holidayname as holiday,
    date
FROM
    OPENROWSET(
        BULK 'https://azureopendatastorage.blob.core.windows.net/holidaydatacontainer/Processed/*.parquet',
        FORMAT='PARQUET'
    ) AS [holidays]
WHERE countryorregion = 'United States' AND YEAR(date) = 2016
),
joined_data AS (
SELECT
    *
FROM taxi_rides t
LEFT OUTER JOIN public_holidays p on t.current_day = p.date
)

SELECT
    *,
    holiday_rides =
    CASE
      WHEN holiday is null THEN 0
      WHEN holiday is not null THEN rides_per_day
    END
FROM joined_data
ORDER BY current_day ASC
```

Results Messages

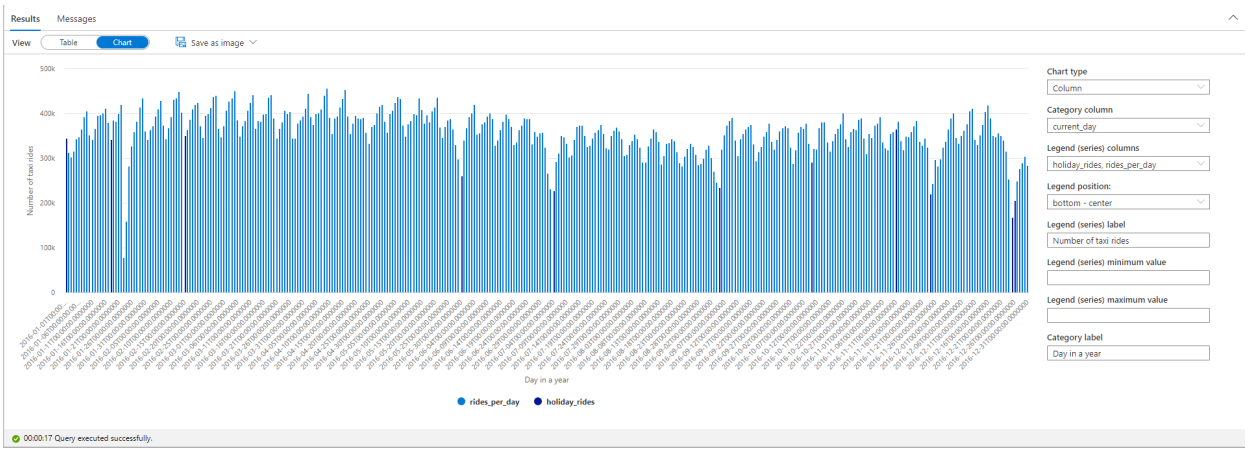
View Table Chart Export results

Search

current_day	rides_per_day	holiday	date	holiday_rides
2016-01-01T00:00:00.0000000	345037	New Year's Day	2016-01-01T00:00:00.0000000	345037
2016-01-02T00:00:00.0000000	312831	(NULL)	(NULL)	0
2016-01-03T00:00:00.0000000	302878	(NULL)	(NULL)	0
2016-01-04T00:00:00.0000000	316171	(NULL)	(NULL)	0
2016-01-05T00:00:00.0000000	343251	(NULL)	(NULL)	0
2016-01-06T00:00:00.0000000	348516	(NULL)	(NULL)	0
2016-01-07T00:00:00.0000000	364894	(NULL)	(NULL)	0
2016-01-08T00:00:00.0000000	392070	(NULL)	(NULL)	0
2016-01-09T00:00:00.0000000	405825	(NULL)	(NULL)	0
2016-01-10T00:00:00.0000000	351788	(NULL)	(NULL)	0
2016-01-11T00:00:00.0000000	342651	(NULL)	(NULL)	0
2016-01-12T00:00:00.0000000	367390	(NULL)	(NULL)	0
2016-01-13T00:00:00.0000000	395090	(NULL)	(NULL)	0
2016-01-14T00:00:00.0000000	396473	(NULL)	(NULL)	0
2016-01-15T00:00:00.0000000	401289	(NULL)	(NULL)	0

00:00:17 Query executed successfully.

Highlight the number of taxi rides during public holidays. For that purpose, choose **current_day** for the **Category** column and **rides_per_day** and **holiday_rides** as the **Legend (series)** columns.



From the plot chart, you can see that during public holidays the number of taxi rides is lower. There's still one unexplained large drop on January 23. Let's check the weather in NYC on that day by querying the Weather Data dataset:

```
SELECT
    AVG(windspeed) AS avg_windspeed,
    MIN(windspeed) AS min_windspeed,
    MAX(windspeed) AS max_windspeed,
    AVG(temperature) AS avg_temperature,
    MIN(temperature) AS min_temperature,
    MAX(temperature) AS max_temperature,
    AVG(sealvlpressure) AS avg_sealvlpressure,
    MIN(sealvlpressure) AS min_sealvlpressure,
    MAX(sealvlpressure) AS max_sealvlpressure,
    AVG(precipdepth) AS avg_precipdepth,
    MIN(precipdepth) AS min_precipdepth,
    MAX(precipdepth) AS max_precipdepth,
    AVG(snowdepth) AS avg_snowdepth,
    MIN(snowdepth) AS min_snowdepth,
    MAX(snowdepth) AS max_snowdepth
FROM
    OPENROWSET(
        BULK
'https://azureopendatastorage.blob.core.windows.net/isdweatherdatacontainer/ISDWeather/year=*/month=*/*.parq
uet',
        FORMAT='PARQUET'
    ) AS [weather]
WHERE countryorregion = 'US' AND CAST([datetime] AS DATE) = '2016-01-23' AND stationname = 'JOHN F KENNEDY
INTERNATIONAL AIRPORT'
```

The results of the query indicate that the drop in the number of taxi rides occurred because:

- There was a blizzard on that day in NYC with heavy snow (~30 cm).
- It was cold (temperature was below zero degrees Celsius).
- It was windy (~10 m/s).

This tutorial has shown how a data analyst can quickly perform exploratory data analysis. You can combine different datasets by using serverless SQL pool and visualize the results by using Azure Synapse Studio.

Next steps

To learn how to connect serverless SQL pool to Power BI Desktop and create reports, see Connect serverless SQL pool to Power BI Desktop and create reports.

To learn how to use External tables in serverless SQL pool see Use external tables with Synapse SQL

Tutorial: Create Logical Data Warehouse with serverless SQL pool

8/19/2022 • 6 minutes to read • Edit Online

In this tutorial, you will learn how to create a Logical Data Warehouse (LDW) on top of Azure storage and Azure Cosmos DB.

LDW is a relational layer built on top of Azure data sources such as Azure Data Lake storage (ADLS), Azure Cosmos DB analytical storage, or Azure Blob storage.

Create an LDW database

You need to create a custom database where you will store your external tables and views that are referencing external data sources.

```
CREATE DATABASE Ldw
      COLLATE Latin1_General_100_BIN2_UTF8;
```

This collation will provide the optimal performance while reading Parquet and Cosmos DB. If you don't want to specify the database collation, make sure that you specify this collation in the column definition.

Configure data sources and formats

As a first step, you need to configure data source and specify file format of remotely stored data.

Create data source

Data sources represent connection string information that describes where your data is placed and how to authenticate to your data source.

One example of data source definition that references public ECDC COVID 19 Azure Open Data Set is shown in the following example:

```
CREATE EXTERNAL DATA SOURCE ecdc_cases WITH (
    LOCATION = 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases/'
);
```

A caller may access data source without credential if an owner of data source allowed anonymous access or give explicit access to Azure AD identity of the caller.

You can explicitly define a custom credential that will be used while accessing data on external data source.

- Managed Identity of the Synapse workspace
- Shared Access Signature of the Azure storage
- Custom Service Principal Name or Azure Application identity.
- Read-only Cosmos Db account key that enables you to read Cosmos DB analytical storage.

As a prerequisite, you will need to create a master key in the database:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Setup you password - you need to create master key only once';
```

In the following external data source, Synapse SQL pool should use a managed identity of the workspace to access data in the storage.

```
CREATE DATABASE SCOPED CREDENTIAL WorkspaceIdentity
WITH IDENTITY = 'Managed Identity';
GO
CREATE EXTERNAL DATA SOURCE ecdc_cases WITH (
    LOCATION = 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases/',
    CREDENTIAL = WorkspaceIdentity
);
```

In order to access Cosmos DB analytical storage, you need to define a credential containing a read-only Cosmos DB account key.

```
CREATE DATABASE SCOPED CREDENTIAL MyCosmosDbAccountCredential
WITH IDENTITY = 'SHARED ACCESS SIGNATURE',
     SECRET = 's5zarR2pT0JWH9k8roipnWxUYBegOuFGjJpSjGlR36y86cW0GQ6RaaG8kGjsRAQoWMw1QKTkkX8HQtFpJjC8Hg==';
```

Any user with the Synapse Administrator role can use these credentials to access Azure Data Lake storage or Cosmos DB analytical storage. If you have low privileged users that do not have Synapse Administrator role, you would need to give them an explicit permission to reference these database scoped credentials:

```
GRANT REFERENCES ON DATABASE SCOPED CREDENTIAL::WorkspaceIdentity TO <user>
GO
GRANT REFERENCES ON DATABASE SCOPED CREDENTIAL::MyCosmosDbAccountCredential TO <user>
GO
```

Find more details in grant DATABASE SCOPED CREDENTIAL permissions page.

Define external file formats

External file formats define the structure of the files stored on external data source. You can define Parquet and CSV external file formats:

```
CREATE EXTERNAL FILE FORMAT ParquetFormat WITH (  FORMAT_TYPE = PARQUET );
GO
CREATE EXTERNAL FILE FORMAT CsvFormat WITH (  FORMAT_TYPE = DELIMITEDTEXT );
```

Find more information in this article

Explore your data

Once you set up your data sources, you can use the `OPENROWSET` function to explore your data. The OPENROWSET function reads content of a remote data source (for example file) and returns the content as a set of rows.

```
select top 10  *
from openrowset(bulk 'latest/ecdc_cases.parquet',
                data_source = 'ecdc_cases',
                format='parquet') as a
```

The `OPENROWSET` function will give you information about the columns in the external files or containers and enable you to define a schema of your external tables and views.

Create external tables on Azure storage

Once you discover the schema, you can create external tables and views on top of your external data sources. The good practice is to organize your tables and views in databases schemas. In the following query you can create a schema where you will place all objects that are accessing ECDC COVID data set in Azure data Lake storage:

```
create schema ecdc_adls;
```

The database schemas are useful for grouping the objects and defining permissions per schema.

Once you define the schemas, you can create external tables that are referencing the files. The following external table is referencing the ECDC COVID parquet file placed in the Azure storage:

```
create external table ecdc_adls.cases (
    date_rep                  date,
    day                       smallint,
    month                     smallint,
    year                      smallint,
    cases                     smallint,
    deaths                    smallint,
    countries_and_territories  varchar(256),
    geo_id                    varchar(60),
    country_territory_code    varchar(16),
    pop_data_2018             int,
    continent_exp             varchar(32),
    load_date                 datetime2(7),
    iso_country               varchar(16)
) with (
    data_source= ecdc_cases,
    location = 'latest/ecdc_cases.parquet',
    file_format = ParquetFormat
);
```

Make sure that you use the smallest possible types for string and number columns to optimize performance of your queries.

Create views on Azure Cosmos DB

As an alternative to external tables, you can create views on top of your external data.

Similar to the tables shown in the previous example, you should place the views in separate schemas:

```
create schema ecdc_cosmosdb;
```

Now you are able to create a view in the schema that is referencing a Cosmos DB container:

```
CREATE OR ALTER VIEW ecdc_cosmosdb.Ecdc
AS SELECT *
FROM OPENROWSET(
      PROVIDER = 'CosmosDB',
      CONNECTION = 'Account=synapselink-cosmosdb-sqlsample;Database=covid',
      OBJECT = 'Ecdc',
      CREDENTIAL = 'MyCosmosDbAccountCredential'
    ) WITH
     ( date_rep varchar(20),
       cases bigint,
       geo_id varchar(6)
     ) as rows
```

To optimize performance, you should use the smallest possible types in the `WITH` schema definition.

> **NOTE**
> You should place your Azure Cosmos DB account key in a separate credential and reference this credential from the `OPENROWSET` function. Do not keep your account key in the view definition.

Access and permissions

As a final step, you should create database users that should be able to access your LDW, and give them permissions to select data from the external tables and views. In the following script you can see how to add a new user that will be authenticated using Azure AD identity:

```
CREATE USER [jovan@contoso.com] FROM EXTERNAL PROVIDER;
GO
```

Instead of Azure AD principals, you can create SQL principals that authenticate with the login name and password.

```
CREATE LOGIN [jovan] WITH PASSWORD = 'My Very strong Password ! 1234';
CREATE USER [jovan] FROM LOGIN [jovan];
```

In both cases, you can assign permissions to the users.

```
DENY ADMINISTER DATABASE BULK OPERATIONS TO [jovan@contoso.com]
GO
GRANT SELECT ON SCHEMA::ecdc_adls TO [jovan@contoso.com]
GO
GRANT SELECT ON OBJECT::ecdc_cosmosDB.cases TO [jovan@contoso.com]
GO
GRANT REFERENCES ON DATABASE SCOPED CREDENTIAL::MyCosmosDbAccountCredential TO [jovan@contoso.com]
GO
```

The security rules depend on your security policies. Some generic guidelines are:

- You should deny `ADMINISTER DATABASE BULK OPERATIONS` permission to the new users because they should be able to read data only using the external tables and views that you prepared.
- You should provide `SELECT` permission only to the tables that some user should be able to use.
- If you are providing access to data using the views, you should grant `REFERENCES` permission to the credential that will be used to access external data source.

This user has minimal permissions needed to query external data. If you want to create a power-user who can set up permissions, external tables and views, you can give `CONTROL` permission to the user:

```
GRANT CONTROL TO [jovan@contoso.com]
```

Role-based security

Instead of assigning permissions to the individual uses, a good practice it to organize the users into roles and manage permission at role-level. The following code sample creates a new role representing the people who can analyze COVID-19 cases, and adds three users to this role:

```
CREATE ROLE CovidAnalyst;

ALTER ROLE CovidAnalyst ADD MEMBER [jovan@contoso.com];
ALTER ROLE CovidAnalyst ADD MEMBER [milan@contoso.com];
ALTER ROLE CovidAnalyst ADD MEMBER [petar@contoso.com];
```

You can assign the permissions to all users that belong to the group:

```
GRANT SELECT ON SCHEMA::ecdc_cosmosdb TO [CovidAnalyst];
GO
DENY SELECT ON SCHEMA::ecdc_adls TO [CovidAnalyst];
GO
DENY ADMINISTER DATABASE BULK OPERATIONS TO [CovidAnalyst];
```

This role-based security access control might simplify management of your security rules.

Next steps

- To learn how to connect serverless SQL pool to Power BI Desktop and create reports, see Connect serverless SQL pool to Power BI Desktop and create reports.
- To learn how to use External tables in serverless SQL pool see Use external tables with Synapse SQL

Tutorial: Use serverless SQL pool with Power BI Desktop & create a report

8/19/2022 • 3 minutes to read • Edit Online

In this tutorial, you'll learn how to:

- Create demo database
- Create view used for report
- Connect Power BI Desktop to serverless SQL pool
- Create report based on view

Prerequisites

To complete this tutorial, you need the following prerequisites:

- Power BI Desktop - needed to visualize the data and create a report.
- Azure Synapse workspace - needed to create database, external data source, and view.

Optional:

- A SQL query tool, such as Azure Data Studio, or SQL Server Management Studio (SSMS).

Values for the following parameters:

PARAMETER	DESCRIPTION
Serverless SQL pool service endpoint address	Used as server name
Serverless SQL pool service endpoint region	Used to determine the storage used in the samples
Username and password for endpoint access	Used to access endpoint
Database you'll use to create views	The database used as starting point in the samples

1 - Create database

For the demo environment, create your own demo database. You use this database to view metadata, not to store actual data.

Create the demo database (and drop an existing database if necessary) by running the following Transact-SQL (T-SQL) script:

```
-- Drop database if it exists
DROP DATABASE IF EXISTS Demo
GO

-- Create new database
CREATE DATABASE [Demo];
GO
```

2 - Create data source

A data source is necessary for the serverless SQL pool service to access files in storage. Create the data source for a storage account that is located in the same region as your endpoint. Although serverless SQL pool can access storage accounts from different regions, having the storage and endpoint in the same region provides better performance.

Create the data source by running the following Transact-SQL (T-SQL) script:

```
-- There is no credential in data surce. We are using public storage account which doesn't need a secret.
CREATE EXTERNAL DATA SOURCE AzureOpenData
WITH ( LOCATION = 'https://azureopendatastorage.blob.core.windows.net/')
```

3 - Prepare view

Create the view based on the external demo data for Power BI to consume by running the following Transact-SQL (T-SQL) script:

Create the view `usPopulationView` inside the database `Demo` with the following query:

```
DROP VIEW IF EXISTS usPopulationView;
GO

CREATE VIEW usPopulationView AS
SELECT
    *
FROM
    OPENROWSET(
        BULK 'censusdatacontainer/release/us_population_county/year=20*/*.parquet',
        DATA_SOURCE = 'AzureOpenData',
        FORMAT='PARQUET'
    ) AS uspv;
```

The demo data contains the following data sets:

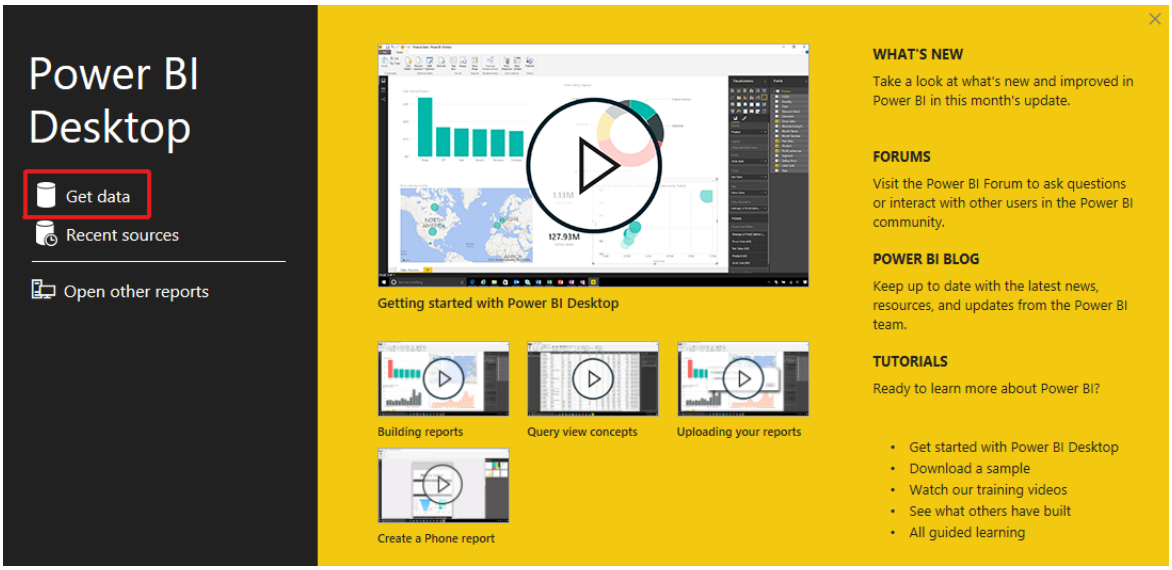
US population by gender and race for each US county sourced from 2000 and 2010 Decennial Census in parquet format.

FOLDER PATH	DESCRIPTION
/release/	Parent folder for data in demo storage account
/release/us_population_county/	US population data files in Parquet format, partitioned by year using Hive/Hadoop partitioning scheme.

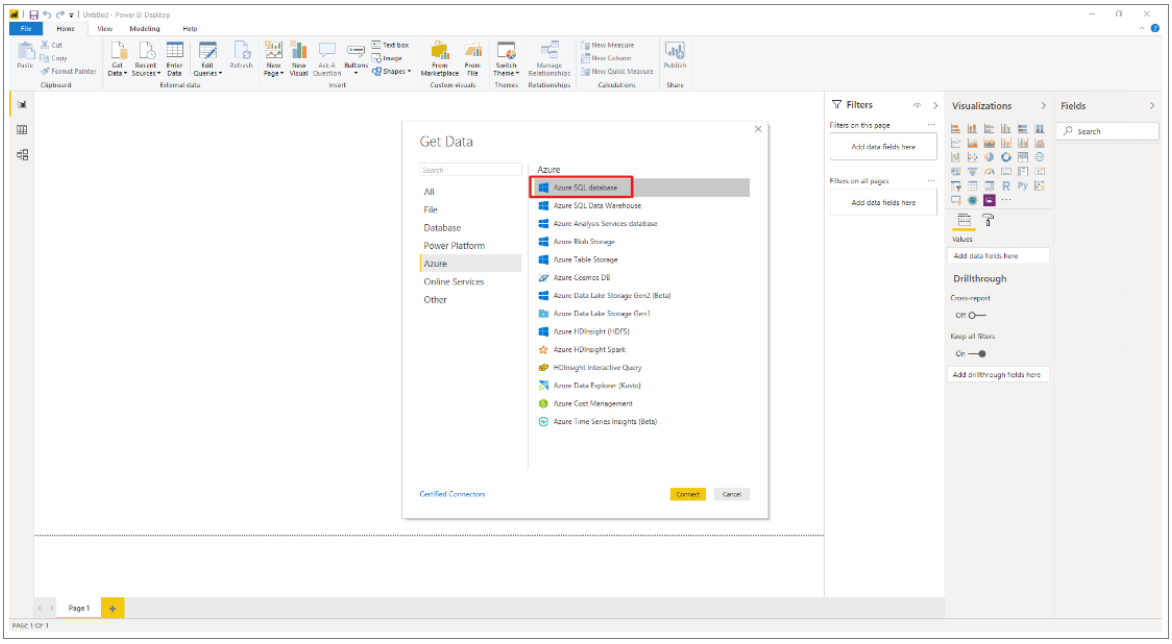
4 - Create Power BI report

Create the report for Power BI Desktop using the following steps:

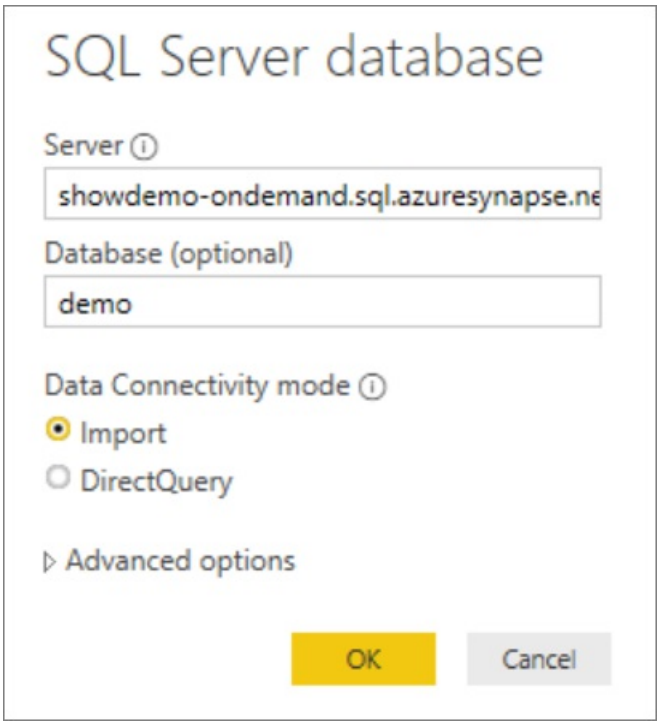
1. Open the Power BI Desktop application and select **Get data**.



2. Select **Azure** > **Azure SQL Database**.

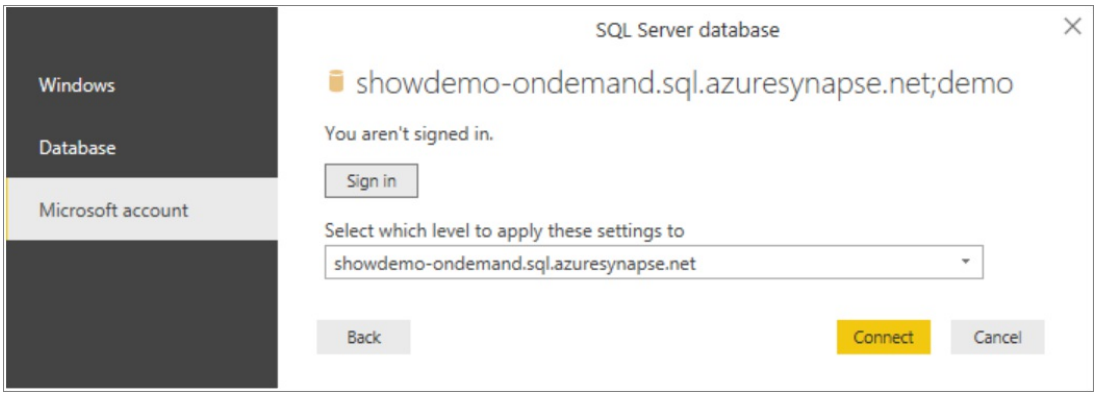


3. Type the name of the server where the database is located in the **Server** field, and then type `Demo` in the database name. Select the **Import** option and then select **OK**.

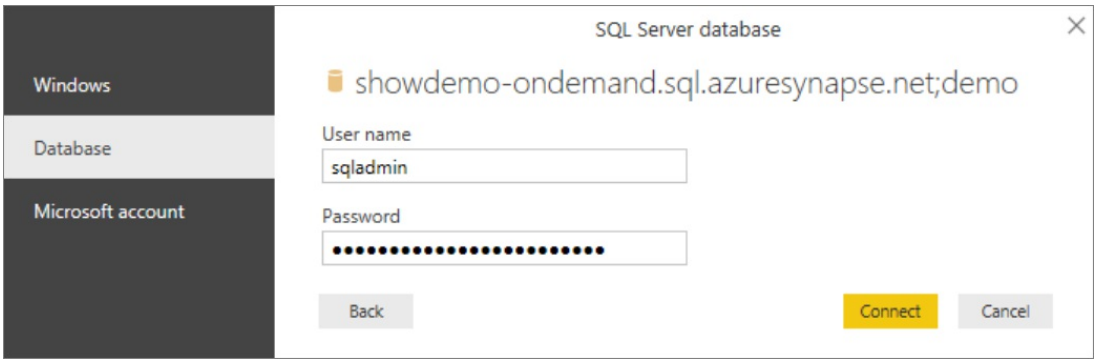


4. Select preferred authentication method:

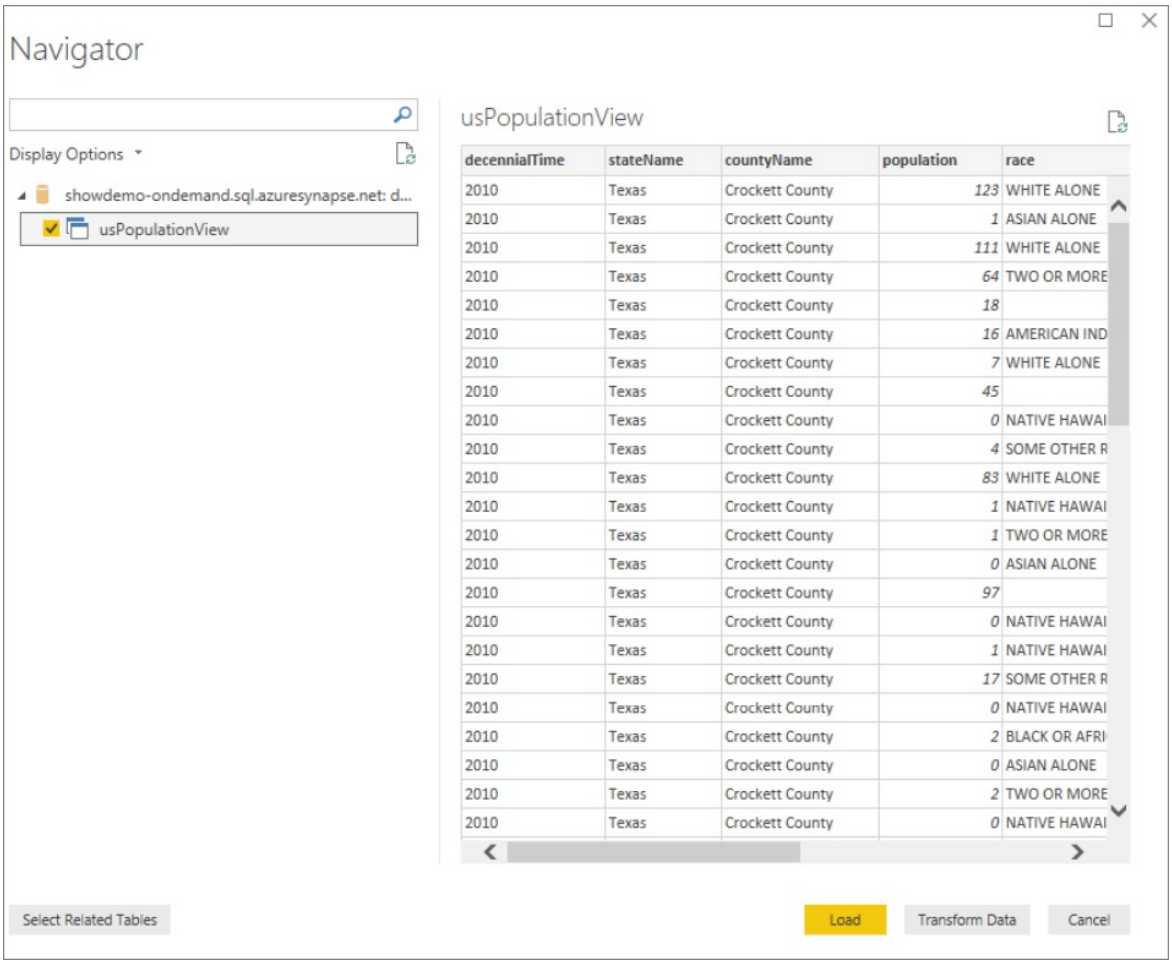
   - Example for AAD



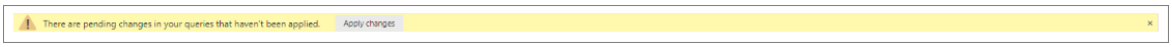
   - Example for SQL Login - Type your User name and password.



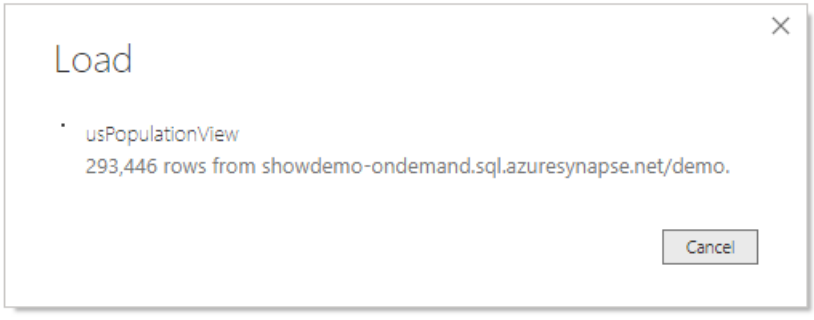
5. Select the view `usPopulationView`, and then select **Load**.



6. Wait for the operation to complete, and then a pop-up will appear stating `There are pending changes in your queries that haven't been applied`. Select **Apply changes**.

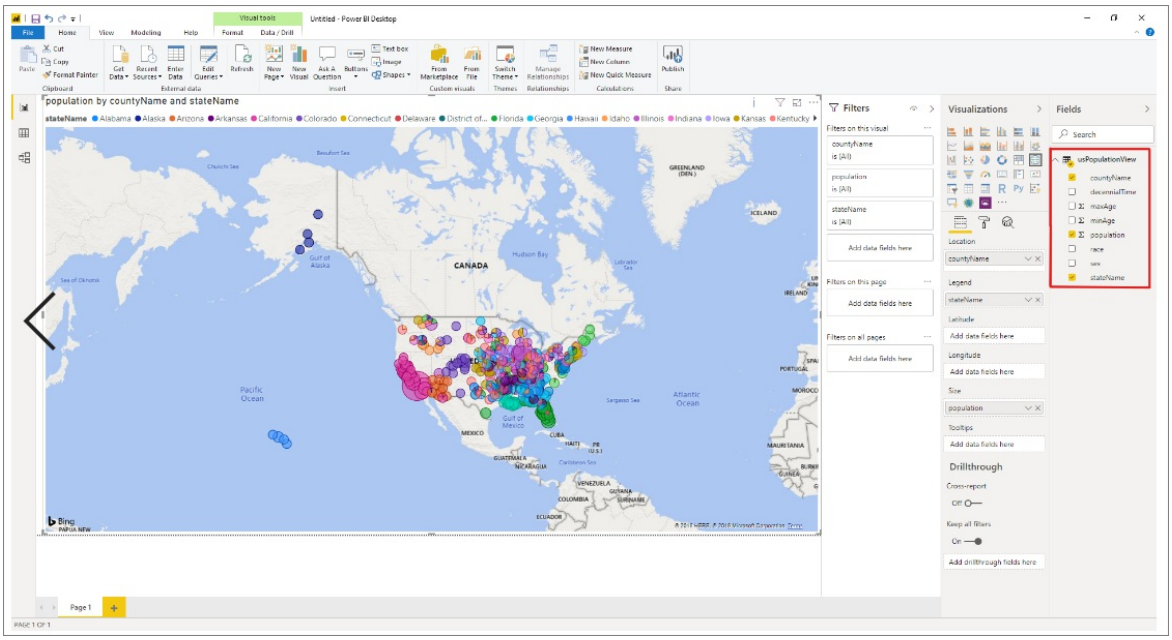


7. Wait for the **Apply query changes** dialog box to disappear, which may take a few minutes.



8. Once the load completes, select the following columns in this order to create the report:

- countyName
- population
- stateName



Clean up resources

Once you're done using this report, delete the resources with the following steps:

1. Delete the credential for the storage account

```
DROP EXTERNAL DATA SOURCE AzureOpenData
```

2. Delete the view

```
DROP VIEW usPopulationView;
```

3. Drop the database

```
DROP DATABASE Demo;
```

Next steps

Advance to the Query storage files to learn how to query storage files using Synapse SQL.

Backup and restore in Azure Synapse Dedicated SQL pool

8/19/2022 • 8 minutes to read • Edit Online

Learn how to use backup and restore in Azure Synapse Dedicated SQL pool. Use dedicated SQL pool restore points to recover or copy your data warehouse to a previous state in the primary region. Use data warehouse geo-redundant backups to restore to a different geographical region.

What is a data warehouse snapshot

A *data warehouse snapshot* creates a restore point you can leverage to recover or copy your data warehouse to a previous state. Since dedicated SQL pool is a distributed system, a data warehouse snapshot consists of many files that are located in Azure storage. Snapshots capture incremental changes from the data stored in your data warehouse.

A *data warehouse restore* is a new data warehouse that is created from a restore point of an existing or deleted data warehouse. Restoring your data warehouse is an essential part of any business continuity and disaster recovery strategy because it re-creates your data after accidental corruption or deletion. Data warehouse snapshot is also a powerful mechanism to create copies of your data warehouse for test or development purposes. Dedicated SQL pool restore rates can vary depending on the database size and location of the source and target data warehouse.

Automatic Restore Points

Snapshots are a built-in feature that creates restore points. You do not have to enable this capability. However, the dedicated SQL pool should be in an active state for restore point creation. If it is paused frequently, automatic restore points may not be created so make sure to create user-defined restore point before pausing the dedicated SQL pool. Automatic restore points currently cannot be deleted by users as the service uses these restore points to maintain SLAs for recovery.

Snapshots of your data warehouse are taken throughout the day creating restore points that are available for seven days. This retention period cannot be changed. Dedicated SQL pool supports an eight-hour recovery point objective (RPO). You can restore your data warehouse in the primary region from any one of the snapshots taken in the past seven days.

To see when the last snapshot started, run this query on your online dedicated SQL pool.

```
select   top 1 *
from     sys.pdw_loader_backup_runs
order by run_id desc
;
```

User-Defined Restore Points

This feature enables you to manually trigger snapshots to create restore points of your data warehouse before and after large modifications. This capability ensures that restore points are logically consistent, which provides additional data protection in case of any workload interruptions or user errors for quick recovery time. User-defined restore points are available for seven days and are automatically deleted on your behalf. You cannot change the retention period of user-defined restore points. **42 user-defined restore points** are guaranteed at

any point in time so they must be deleted before creating another restore point. You can trigger snapshots to create user-defined restore points through PowerShell or the Azure portal.

> **NOTE**
>
> If you require restore points longer than 7 days, please vote for this capability here. You can also create a user-defined restore point and restore from the newly created restore point to a new data warehouse. Once you have restored, you have the dedicated SQL pool online and can pause it indefinitely to save compute costs. The paused database incurs storage charges at the Azure Synapse storage rate. If you need an active copy of the restored data warehouse, you can resume which should take only a few minutes.

Restore point retention

The following lists details for restore point retention periods:

1. Dedicated SQL pool deletes a restore point when it hits the 7-day retention period **and** when there are at least 42 total restore points (including both user-defined and automatic).
2. Snapshots are not taken when a dedicated SQL pool is paused.
3. The age of a restore point is measured by the absolute calendar days from the time the restore point is taken including when the SQL pool is paused.
4. At any point in time, a dedicated SQL pool is guaranteed to be able to store up to 42 user-defined restore points or 42 automatic restore points as long as these restore points have not reached the 7-day retention period
5. If a snapshot is taken, the dedicated SQL pool is then paused for greater than 7 days, and then resumed, the restore point will persist until there are 42 total restore points (including both user-defined and automatic)

Snapshot retention when a SQL pool is dropped

When you drop a dedicated SQL pool, a final snapshot is created and saved for seven days. You can restore the dedicated SQL pool to the final restore point created at deletion. If the dedicated SQL pool is dropped in a paused state, no snapshot is taken. In that scenario, make sure to create a user-defined restore point before dropping the dedicated SQL pool.

Geo-backups and disaster recovery

A geo-backup is created once per day to a paired data center. The RPO for a geo-restore is 24 hours. A geo-restore is always a data movement operation and the RTO will depend on the data size. Only the latest geo-backup is retained. You can restore the geo-backup to a server in any other region where dedicated SQL pool is supported. A geo-backup ensures you can restore data warehouse in case you cannot access the restore points in your primary region.

If you do not require geo-backups for your dedicated SQL pool, you can disable them and save on disaster recovery storage costs. To do so, refer to How to guide: Disable geo-backups for a dedicated SQL pool (formerly SQL DW). Note that if you disable geo-backups, you will not be able to recover your dedicated SQL pool to your paired Azure region if your primary Azure data center is unavailable.

NOTE

If you require a shorter RPO for geo-backups, vote for this capability here. You can also create a user-defined restore point and restore from the newly created restore point to a new data warehouse in a different region. After you have restored, you have the data warehouse online and can pause it indefinitely to save compute costs. The paused database incurs storage charges at the Azure Premium Storage rate. Another common pattern for a shorter recovery point is to ingest data into primary and secondary instances of a data warehouse in parallel. In this scenario, data is ingested from a source (or sources) and persisted to two separate instances of the data warehouse (primary and secondary). To save on compute costs, you can pause the secondary instance of the warehouse. If you need an active copy of the data warehouse, you can resume, which should take only a few minutes.

Data residency

If your paired data center is located outside of your country, you can ensure that your data stays within your region by provisioning your database on locally redundant storage (LRS). If your database has already been provisioned on RA-GRS (Read Only Geographically Redundant Storage, the current default) then you can opt out of geo-backups, however your database will continue to reside on storage that is replicated to a regional pair. To ensure that customer data stays within your region, you can provision or restore your dedicated SQL pool to locally redundant storage. For more information on how to provision or restore to local redundant storage, see How-to guide for configuring single region residency for a dedicated SQL pool (formerly SQL DW) in Azure Synapse Analytics

To confirm that your paired data center is in a different country, refer to Azure Paired Regions.

Backup and restore costs

You will notice the Azure bill has a line item for Storage and a line item for Disaster Recovery Storage. The storage charge is the total cost for storing your data in the primary region along with the incremental changes captured by snapshots. For a more detailed explanation of how snapshots are charged, refer to Understanding how Snapshots Accrue Charges. The geo-redundant charge covers the cost for storing the geo-backups.

The total cost for your primary data warehouse and seven days of snapshot changes is rounded to the nearest TB. For example, if your data warehouse is 1.5 TB and the snapshots captures 100 GB, you are billed for 2 TB of data at Azure standard storage rates.

If you are using geo-redundant storage, you receive a separate storage charge. The geo-redundant storage is billed at the standard Read-Access Geographically Redundant Storage (RA-GRS) rate.

For more information about Azure Synapse pricing, see Azure Synapse pricing. You are not charged for data egress when restoring across regions.

Restoring from restore points

Each snapshot creates a restore point that represents the time the snapshot started. To restore a data warehouse, you choose a restore point and issue a restore command.

You can either keep the restored data warehouse and the current one, or delete one of them. If you want to replace the current data warehouse with the restored data warehouse, you can rename it using ALTER DATABASE with the MODIFY NAME option.

To restore a data warehouse, see Restore a dedicated SQL pool.

To restore a deleted data warehouse, see Restore a deleted database, or if the entire server was deleted, see Restore a data warehouse from a deleted server.

Cross subscription restore

You can perform a cross-subscription restore by follow the guidance here.

Geo-redundant restore

You can restore your dedicated SQL pool to any region supporting dedicated SQL pool at your chosen performance level.

> **NOTE**
> To perform a geo-redundant restore you must not have opted out of this feature.

Support Process

You can submit a support ticket through the Azure portal for Azure Synapse Analytics.

Next steps

For more information about restore points, see User-defined restore points

Data loading strategies for dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

Traditional SMP dedicated SQL pools use an Extract, Transform, and Load (ETL) process for loading data. Synapse SQL, within Azure Synapse Analytics, uses distributed query processing architecture that takes advantage of the scalability and flexibility of compute and storage resources.

Using an Extract, Load, and Transform (ELT) process leverages built-in distributed query processing capabilities and eliminates the resources needed for data transformation prior to loading.

While dedicated SQL pools support many loading methods, including popular SQL Server options such as bcp and the SqlBulkCopy API, the fastest and most scalable way to load data is through PolyBase external tables and the COPY statement.

With PolyBase and the COPY statement, you can access external data stored in Azure Blob storage or Azure Data Lake Store via the T-SQL language. For the most flexibility when loading, we recommend using the COPY statement.

What is ELT?

Extract, Load, and Transform (ELT) is a process by which data is extracted from a source system, loaded into a dedicated SQL pool, and then transformed.

The basic steps for implementing ELT are:

1. Extract the source data into text files.
2. Land the data into Azure Blob storage or Azure Data Lake Store.
3. Prepare the data for loading.
4. Load the data into staging tables with PolyBase or the COPY command.
5. Transform the data.
6. Insert the data into production tables.

For a loading tutorial, see loading data from Azure blob storage.

1. Extract the source data into text files

Getting data out of your source system depends on the storage location. The goal is to move the data into supported delimited text or CSV files.

Supported file formats

With PolyBase and the COPY statement, you can load data from UTF-8 and UTF-16 encoded delimited text or CSV files. In addition to delimited text or CSV files, it loads from the Hadoop file formats such as ORC and Parquet. PolyBase and the COPY statement can also load data from Gzip and Snappy compressed files.

Extended ASCII, fixed-width format, and nested formats such as WinZip or XML aren't supported. If you're exporting from SQL Server, you can use the bcp command-line tool to export the data into delimited text files.

2. Land the data into Azure Blob storage or Azure Data Lake Store

To land the data in Azure storage, you can move it to Azure Blob storage or Azure Data Lake Store Gen2. In

either location, the data should be stored in text files. PolyBase and the COPY statement can load from either location.

Tools and services you can use to move data to Azure Storage:

- Azure ExpressRoute service enhances network throughput, performance, and predictability. ExpressRoute is a service that routes your data through a dedicated private connection to Azure. ExpressRoute connections do not route data through the public internet. The connections offer more reliability, faster speeds, lower latencies, and higher security than typical connections over the public internet.
- AzCopy utility moves data to Azure Storage over the public internet. This works if your data sizes are less than 10 TB. To perform loads on a regular basis with AzCopy, test the network speed to see if it is acceptable.
- Azure Data Factory (ADF) has a gateway that you can install on your local server. Then you can create a pipeline to move data from your local server up to Azure Storage. To use Data Factory with dedicated SQL pools, see Loading data for dedicated SQL pools.

3. Prepare the data for loading

You might need to prepare and clean the data in your storage account before loading. Data preparation can be performed while your data is in the source, as you export the data to text files, or after the data is in Azure Storage. It is easiest to work with the data as early in the process as possible.

Define the tables

You must first define the table(s) you are loading to in your dedicated SQL pool when using the COPY statement.

If you are using PolyBase, you need to define external tables in your dedicated SQL pool before loading. PolyBase uses external tables to define and access the data in Azure Storage. An external table is similar to a database view. The external table contains the table schema and points to data that is stored outside the dedicated SQL pool.

Defining external tables involves specifying the data source, the format of the text files, and the table definitions. T-SQL syntax reference articles that you will need are:

- CREATE EXTERNAL DATA SOURCE
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE

Use the following SQL data type mapping when loading Parquet files:

PARQUET TYPE	PARQUET LOGICAL TYPE (ANNOTATION)	SQL DATA TYPE
BOOLEAN		bit
BINARY / BYTE_ARRAY		varbinary
DOUBLE		float
FLOAT		real
INT32		int
INT64		bigint
INT96		datetime2

PARQUET TYPE	PARQUET LOGICAL TYPE (ANNOTATION)	SQL DATA TYPE
FIXED_LEN_BYTE_ARRAY		binary
BINARY	UTF8	nvarchar
BINARY	STRING	nvarchar
BINARY	ENUM	nvarchar
BINARY	UUID	uniqueidentifier
BINARY	DECIMAL	decimal
BINARY	JSON	nvarchar(MAX)
BINARY	BSON	varbinary(max)
FIXED_LEN_BYTE_ARRAY	DECIMAL	decimal
BYTE_ARRAY	INTERVAL	varchar(max),
INT32	INT(8, true)	smallint
INT32	INT(16, true)	smallint
INT32	INT(32, true)	int
INT32	INT(8, false)	tinyint
INT32	INT(16, false)	int
INT32	INT(32, false)	bigint
INT32	DATE	date
INT32	DECIMAL	decimal
INT32	TIME (MILLIS)	time
INT64	INT(64, true)	bigint
INT64	INT(64, false)	decimal(20,0)
INT64	DECIMAL	decimal
INT64	TIME (MILLIS)	time
INT64	TIMESTAMP (MILLIS)	datetime2
Complex type	LIST	varchar(max)

PARQUET TYPE	PARQUET LOGICAL TYPE (ANNOTATION)	SQL DATA TYPE
[Complex type] (https://github.com/apache/parquet-format/blob/master/LogicalTypes.md	MAP	varchar(max)

IMPORTANT

- SQL dedicated pools do not currently support Parquet data types with MICROS and NANOS precision.
- You may experience the following error if types are mismatched between Parquet and SQL or if you have unsupported Parquet data types: **"HdfsBridge::recordReaderFillBuffer - Unexpected error encountered filling record reader buffer: ClassCastException: ..."**
- Loading a value outside the range of 0-127 into a tinyint column for Parquet and ORC file format is not supported.

For an example of creating external objects, see Create external tables.

Format text files

If you are using PolyBase, the external objects defined need to align the rows of the text files with the external table and file format definition. The data in each row of the text file must align with the table definition. To format the text files:

- If your data is coming from a non-relational source, you need to transform it into rows and columns. Whether the data is from a relational or non-relational source, the data must be transformed to align with the column definitions for the table into which you plan to load the data.
- Format data in the text file to align with the columns and data types in the destination table. Misalignment between data types in the external text files and the dedicated SQL pool table causes rows to be rejected during the load.
- Separate fields in the text file with a terminator. Be sure to use a character or a character sequence that isn't found in your source data. Use the terminator you specified with CREATE EXTERNAL FILE FORMAT.

4. Load the data using PolyBase or the COPY statement

It is best practice to load data into a staging table. Staging tables allow you to handle errors without interfering with the production tables. A staging table also gives you the opportunity to use the dedicated SQL pool parallel processing architecture for data transformations before inserting the data into production tables.

Options for loading

To load data, you can use any of these loading options:

- The COPY statement is the recommended loading utility as it enables you to seamlessly and flexibly load data. The statement has many additional loading capabilities that PolyBase does not provide. See the NY taxi cab COPY tutorial to run through a sample tutorial.
- PolyBase with T-SQL requires you to define external data objects.
- PolyBase and COPY statement with Azure Data Factory (ADF) is another orchestration tool. It defines a pipeline and schedules jobs.
- PolyBase with SSIS works well when your source data is in SQL Server. SSIS defines the source to destination table mappings, and also orchestrates the load. If you already have SSIS packages, you can modify the packages to work with the new data warehouse destination.
- PolyBase with Azure Databricks transfers data from a table to a Databricks dataframe and/or writes data from a Databricks dataframe to a table using PolyBase.

Other loading options

In addition to PolyBase and the COPY statement, you can use bcp or the SqlBulkCopy API. bcp loads directly to the database without going through Azure Blob storage, and is intended only for small loads.

> **NOTE**
> The load performance of these options is slower than PolyBase and the COPY statement.

5. Transform the data

While data is in the staging table, perform transformations that your workload requires. Then move the data into a production table.

6. Insert the data into production tables

The INSERT INTO ... SELECT statement moves the data from the staging table to the permanent table.

As you design an ETL process, try running the process on a small test sample. Try extracting 1000 rows from the table to a file, move it to Azure, and then try loading it into a staging table.

Partner loading solutions

Many of our partners have loading solutions. To find out more, see a list of our solution partners.

Next steps

For loading guidance, see Data loading best practices.

Best practices for loading data into a dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

In this article, you'll find recommendations and performance optimizations for loading data.

Prepare data in Azure Storage

To minimize latency, colocate your storage layer and your dedicated SQL pool.

When exporting data into an ORC File Format, you might get Java out-of-memory errors when there are large text columns. To work around this limitation, export only a subset of the columns.

PolyBase can't load rows that have more than 1,000,000 bytes of data. When you put data into the text files in Azure Blob storage or Azure Data Lake Store, they must have fewer than 1,000,000 bytes of data. This byte limitation is true regardless of the table schema.

All file formats have different performance characteristics. For the fastest load, use compressed delimited text files. The difference between UTF-8 and UTF-16 performance is minimal.

Split large compressed files into smaller compressed files.

Run loads with enough compute

For fastest loading speed, run only one load job at a time. If that is not feasible, run a minimal number of loads concurrently. If you expect a large loading job, consider scaling up your dedicated SQL pool before the load.

To run loads with appropriate compute resources, create loading users designated for running loads. Assign each loading user to a specific resource class or workload group. To run a load, sign in as one of the loading users, and then run the load. The load runs with the user's resource class. This method is simpler than trying to change a user's resource class to fit the current resource class need.

Create a loading user

This example creates a loading user classified to a specific workload group. The first step is to **connect to master** and create a login.

```
-- Connect to master
CREATE LOGIN loader WITH PASSWORD = 'a123STRONGpassword!';
```

Connect to the dedicated SQL pool and create a user. The following code assumes you're connected to the database called mySampleDataWarehouse. It shows how to create a user called loader and gives the user permissions to create tables and load using the COPY statement. Then it classifies the user to the DataLoads workload group with maximum resources.

```
-- Connect to the dedicated SQL pool
CREATE USER loader FOR LOGIN loader;
GRANT ADMINISTER DATABASE BULK OPERATIONS TO loader;
GRANT INSERT ON <yourtablename> TO loader;
GRANT SELECT ON <yourtablename> TO loader;
GRANT CREATE TABLE TO loader;
GRANT ALTER ON SCHEMA::dbo TO loader;

CREATE WORKLOAD GROUP DataLoads
WITH (
    MIN_PERCENTAGE_RESOURCE = 0
    ,CAP_PERCENTAGE_RESOURCE = 100
    ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 100
);

CREATE WORKLOAD CLASSIFIER [wgcELTLogin]
WITH (
   WORKLOAD_GROUP = 'DataLoads'
    ,MEMBERNAME = 'loader'
);
```

IMPORTANT

This is an extreme example of allocating 100% resources of the SQL pool to a single load. This will give you a maximum concurrency of 1. Be aware that this should be used only for the initial load where you will need to create additional workload groups with their own configurations to balance resources across your workloads.

To run a load with resources for the loading workload group, sign in as loader and run the load.

Allow multiple users to load

There is often a need to have multiple users load data into a data warehouse. Loading with the CREATE TABLE AS SELECT (Transact-SQL) requires CONTROL permissions of the database. The CONTROL permission gives control access to all schemas. You might not want all loading users to have control access on all schemas. To limit permissions, use the DENY CONTROL statement.

For example, consider database schemas, schema_A for dept A, and schema_B for dept B. Let database users user_A and user_B be users for PolyBase loading in dept A and B, respectively. They both have been granted CONTROL database permissions. The creators of schema A and B now lock down their schemas using DENY:

```
DENY CONTROL ON SCHEMA :: schema_A TO user_B;
DENY CONTROL ON SCHEMA :: schema_B TO user_A;
```

User_A and user_B are now locked out from the other dept's schema.

Load to a staging table

To achieve the fastest loading speed for moving data into a data warehouse table, load data into a staging table. Define the staging table as a heap and use round-robin for the distribution option.

Consider that loading is usually a two-step process in which you first load to a staging table and then insert the data into a production data warehouse table. If the production table uses a hash distribution, the total time to load and insert might be faster if you define the staging table with the hash distribution. Loading to the staging table takes longer, but the second step of inserting the rows to the production table does not incur data

movement across the distributions.

Load to a columnstore index

Columnstore indexes require large amounts of memory to compress data into high-quality rowgroups. For best compression and index efficiency, the columnstore index needs to compress the maximum of 1,048,576 rows into each rowgroup. When there is memory pressure, the columnstore index might not be able to achieve maximum compression rates. This effects query performance. For a deep dive, see Columnstore memory optimizations.

- To ensure the loading user has enough memory to achieve maximum compression rates, use loading users that are a member of a medium or large resource class.
- Load enough rows to completely fill new rowgroups. During a bulk-load, every 1,048,576 rows get compressed directly into the columnstore as a full rowgroup. Loads with fewer than 102,400 rows send the rows to the deltastore where rows are held in a b-tree index. If you load too few rows, they might all go to the deltastore and not get compressed immediately into columnstore format.

Increase batch size when using SQLBulkCopy API or BCP

Loading with the COPY statement will provide the highest throughput with dedicated SQL pools. If you cannot use the COPY to load and must use the SqLBulkCopy API or bcp, you should consider increasing batch size for better throughput.

> **TIP**
> A batch size between 100 K to 1M rows is the recommended baseline for determining optimal batch size capacity.

Manage loading failures

A load using an external table can fail with the error *"Query aborted-- the maximum reject threshold was reached while reading from an external source"*. This message indicates that your external data contains dirty records. A data record is considered dirty if the data types and number of columns do not match the column definitions of the external table, or if the data doesn't conform to the specified external file format.

To fix the dirty records, ensure that your external table and external file format definitions are correct and your external data conforms to these definitions. In case a subset of external data records are dirty, you can choose to reject these records for your queries by using the reject options in 'CREATE EXTERNAL TABLE' .

Insert data into a production table

A one-time load to a small table with an INSERT statement, or even a periodic reload of a look-up might perform good enough with a statement like `INSERT INTO MyLookup VALUES (1, 'Type 1')`. However, singleton inserts are not as efficient as performing a bulk-load.

If you have thousands or more single inserts throughout the day, batch the inserts so you can bulk load them. Develop your processes to append the single inserts to a file, and then create another process that periodically loads the file.

Create statistics after the load

To improve query performance, it's important to create statistics on all columns of all tables after the first load, or major changes occur in the data. Create statistics can be done manually or you can enable auto-create statistics.

For a detailed explanation of statistics, see Statistics. The following example shows how to manually create statistics on five columns of the Customer_Speed table.

```
create statistics [SensorKey] on [Customer_Speed] ([SensorKey]);
create statistics [CustomerKey] on [Customer_Speed] ([CustomerKey]);
create statistics [GeographyKey] on [Customer_Speed] ([GeographyKey]);
create statistics [Speed] on [Customer_Speed] ([Speed]);
create statistics [YearMeasured] on [Customer_Speed] ([YearMeasured]);
```

Rotate storage keys

It is good security practice to change the access key to your blob storage on a regular basis. You have two storage keys for your blob storage account, which enables you to transition the keys.

To rotate Azure Storage account keys:

For each storage account whose key has changed, issue ALTER DATABASE SCOPED CREDENTIAL.

Example:

Original key is created

```
CREATE DATABASE SCOPED CREDENTIAL my_credential WITH IDENTITY = 'my_identity', SECRET = 'key1'
```

Rotate key from key 1 to key 2

```
ALTER DATABASE SCOPED CREDENTIAL my_credential WITH IDENTITY = 'my_identity', SECRET = 'key2'
```

No other changes to underlying external data sources are needed.

Next steps

- To learn more about PolyBase and designing an Extract, Load, and Transform (ELT) process, see Design ELT for Azure Synapse Analytics.
- For a loading tutorial, Use PolyBase to load data from Azure blob storage to Azure Synapse Analytics.
- To monitor data loads, see Monitor your workload using DMVs.

Maximize rowgroup quality for columnstore index performance

8/19/2022 • 6 minutes to read • Edit Online

Rowgroup quality is determined by the number of rows in a rowgroup. Increasing the available memory can maximize the number of rows a columnstore index compresses into each rowgroup. Use these methods to improve compression rates and query performance for columnstore indexes.

Why the rowgroup size matters

Since a columnstore index scans a table by scanning column segments of individual rowgroups, maximizing the number of rows in each rowgroup enhances query performance. When rowgroups have a high number of rows, data compression improves which means there is less data to read from disk.

For more information about rowgroups, see Columnstore Indexes Guide.

Target size for rowgroups

For best query performance, the goal is to maximize the number of rows per rowgroup in a columnstore index. A rowgroup can have a maximum of 1,048,576 rows. It's okay to not have the maximum number of rows per rowgroup. Columnstore indexes achieve good performance when rowgroups have at least 100,000 rows.

Rowgroups can get trimmed during compression

During a bulk load or columnstore index rebuild, sometimes there's not enough memory available to compress all the rows designated for each rowgroup. When there is memory pressure, columnstore indexes trim the rowgroup sizes so compression into the columnstore can succeed.

When there is insufficient memory to compress at least 10,000 rows into each rowgroup, an error will be generated.

For more information on bulk loading, see Bulk load into a clustered columnstore index.

How to monitor rowgroup quality

The dynamic management view (DMV) (sys.dm_db_column_store_row_group_physical_stats contains the view definition matching SQL DB) that exposes useful information such as number of rows in rowgroups and the reason for trimming if there was trimming. You can create the following view as a handy way to query this DMV to get information on rowgroup trimming.

```
CREATE VIEW dbo.vCS_rg_physical_stats
AS
WITH cte
AS
(
select   tb.[name]                    AS [logical_table_name]
,        rg.[row_group_id]            AS [row_group_id]
,        rg.[state]                   AS [state]
,        rg.[state_desc]              AS [state_desc]
,        rg.[total_rows]              AS [total_rows]
,        rg.[trim_reason_desc]        AS trim_reason_desc
,        mp.[physical_name]           AS physical_name
FROM    sys.[schemas] sm
JOIN    sys.[tables] tb               ON  sm.[schema_id]          = tb.[schema_id]
JOIN    sys.[pdw_table_mappings] mp   ON  tb.[object_id]          = mp.[object_id]
JOIN    sys.[pdw_nodes_tables] nt     ON  nt.[name]               = mp.[physical_name]
JOIN    sys.[dm_pdw_nodes_db_column_store_row_group_physical_stats] rg      ON  rg.[object_id]     = nt.
[object_id]
                                                                            AND rg.[pdw_node_id]   = nt.
[pdw_node_id]
                                      AND rg.[distribution_id]    = nt.[distribution_id]
)
SELECT *
FROM cte;
```

The `trim_reason_desc` column indicates whether the rowgroup was trimmed (trim_reason_desc = NO_TRIM implies there was no trimming and row group is of optimal quality). The following trim reasons indicate premature trimming of the rowgroup:

- BULKLOAD: This trim reason is used when the incoming batch of rows for the load had less than 1 million rows. The engine will create compressed row groups if there are greater than 100,000 rows being inserted (as opposed to inserting into the delta store) but sets the trim reason to BULKLOAD. In this scenario, consider increasing your batch load to include more rows. Also, reevaluate your partitioning scheme to ensure it is not too granular as row groups cannot span partition boundaries.
- MEMORY_LIMITATION: To create row groups with 1 million rows, a certain amount of working memory is required by the engine. When available memory of the loading session is less than the required working memory, row groups get prematurely trimmed. The following sections explain how to estimate memory required and allocate more memory.
- DICTIONARY_SIZE: This trim reason indicates that rowgroup trimming occurred because there was at least one string column with wide and/or high cardinality strings. The dictionary size is limited to 16 MB in memory and once this limit is reached the row group is compressed. If you do run into this situation, consider isolating the problematic column into a separate table.

How to estimate memory requirements

The maximum required memory to compress one rowgroup is, approximately, as follows:

- 72 MB +
- #rows * #columns * 8 bytes +
- #rows * #short-string-columns * 32 bytes +
- #long-string-columns * 16 MB for compression dictionary

> **NOTE**
> Where short-string-columns use string data types of <= 32 bytes and long-string-columns use string data types of > 32 bytes.

Long strings are compressed with a compression method designed for compressing text. This compression method uses a *dictionary* to store text patterns. The maximum size of a dictionary is 16 MB. There is only one dictionary for each long string column in the rowgroup.

Ways to reduce memory requirements

Use the following techniques to reduce the memory requirements for compressing rowgroups into columnstore indexes.

Use fewer columns

If possible, design the table with fewer columns. When a rowgroup is compressed into the columnstore, the columnstore index compresses each column segment separately. Therefore the memory requirements to compress a rowgroup increase as the number of columns increases.

Use fewer string columns

Columns of string data types require more memory than numeric and date data types. To reduce memory requirements, consider removing string columns from fact tables and putting them in smaller dimension tables.

Additional memory requirements for string compression:

- String data types up to 32 characters can require 32 additional bytes per value.
- String data types with more than 32 characters are compressed using dictionary methods. Each column in the rowgroup can require up to an additional 16 MB to build the dictionary.

Avoid over-partitioning

Columnstore indexes create one or more rowgroups per partition. For data warehousing in Azure Synapse Analytics, the number of partitions grows quickly because the data is distributed and each distribution is partitioned. If the table has too many partitions, there might not be enough rows to fill the rowgroups. The lack of rows does not create memory pressure during compression, but it leads to rowgroups that do not achieve the best columnstore query performance.

Another reason to avoid over-partitioning is there is a memory overhead for loading rows into a columnstore index on a partitioned table. During a load, many partitions could receive the incoming rows, which are held in memory until each partition has enough rows to be compressed. Having too many partitions creates additional memory pressure.

Simplify the load query

The database shares the memory grant for a query among all the operators in the query. When a load query has complex sorts and joins, the memory available for compression is reduced.

Design the load query to focus only on loading the query. If you need to run transformations on the data, run them separate from the load query. For example, stage the data in a heap table, run the transformations, and then load the staging table into the columnstore index.

Adjust MAXDOP

Each distribution compresses rowgroups into the columnstore in parallel when there's more than one CPU core available per distribution. The parallelism requires additional memory resources, which can lead to memory pressure and rowgroup trimming.

To reduce memory pressure, you can use the MAXDOP query hint to force the load operation to run in serial mode within each distribution.

```
CREATE TABLE MyFactSalesQuota
WITH (DISTRIBUTION = ROUND_ROBIN)
AS SELECT * FROM FactSalesQuota
OPTION (MAXDOP 1);
```

Ways to allocate more memory

DWU size and the user resource class together determine how much memory is available for a user query. To increase the memory grant for a load query, you can either increase the number of DWUs or increase the resource class.

- To increase the DWUs, see How do I scale performance?
- To change the resource class for a query, see Change a user resource class example.

Next steps

To find more ways to improve performance in Synapse SQL, see the Performance overview.

Design a PolyBase data loading strategy for dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

Traditional SMP data warehouses use an Extract, Transform, and Load (ETL) process for loading data. Azure SQL pool is a massively parallel processing (MPP) architecture that takes advantage of the scalability and flexibility of compute and storage resources. An Extract, Load, and Transform (ELT) process can take advantage of built-in distributed query processing capabilities and eliminate resources needed to transform the data before loading.

While SQL pool supports many loading methods including non-Polybase options such as BCP and SQL BulkCopy API, the fastest and most scalable way to load data is through PolyBase. PolyBase is a technology that accesses external data stored in Azure Blob storage or Azure Data Lake Store via the T-SQL language.

Extract, Load, and Transform (ELT)

Extract, Load, and Transform (ELT) is a process by which data is extracted from a source system, loaded into a data warehouse, and then transformed.

The basic steps for implementing a PolyBase ELT for dedicated SQL pool are:

1. Extract the source data into text files.
2. Land the data into Azure Blob storage or Azure Data Lake Store.
3. Prepare the data for loading.
4. Load the data into dedicated SQL pool staging tables using PolyBase.
5. Transform the data.
6. Insert the data into production tables.

For a loading tutorial, see Use PolyBase to load data from Azure blob storage to Azure Synapse Analytics.

For more information, see Loading patterns blog.

1. Extract the source data into text files

Getting data out of your source system depends on the storage location. The goal is to move the data into PolyBase supported delimited text files.

PolyBase external file formats

PolyBase loads data from UTF-8 and UTF-16 encoded delimited text files. PolyBase also loads from the Hadoop file formats RC File, ORC, and Parquet. PolyBase can also load data from Gzip and Snappy compressed files. PolyBase currently does not support extended ASCII, fixed-width format, and nested formats such as WinZip, JSON, and XML.

If you're exporting from SQL Server, you can use bcp command-line tool to export the data into delimited text files. The Parquet to Azure Synapse Analytics data type mapping is as follows:

PARQUET DATA TYPE	SQL DATA TYPE
tinyint	tinyint

PARQUET DATA TYPE	SQL DATA TYPE
smallint	smallint
int	int
bigint	bigint
boolean	bit
double	float
float	real
double	money
double	smallmoney
string	nchar
string	nvarchar
string	char
string	varchar
binary	binary
binary	varbinary
timestamp	date
timestamp	smalldatetime
timestamp	datetime2
timestamp	datetime
timestamp	time
date	date
decimal	decimal

2. Land the data into Azure Blob storage or Azure Data Lake Store

To land the data in Azure storage, you can move it to Azure Blob storage or Azure Data Lake Store. In either location, the data should be stored in text files. PolyBase can load from either location.

Tools and services you can use to move data to Azure Storage:

- Azure ExpressRoute service enhances network throughput, performance, and predictability. ExpressRoute is a service that routes your data through a dedicated private connection to Azure. ExpressRoute connections do

not route data through the public internet. The connections offer more reliability, faster speeds, lower latencies, and higher security than typical connections over the public internet.

- AzCopy utility moves data to Azure Storage over the public internet. This works if your data sizes are less than 10 TB. To perform loads on a regular basis with AzCopy, test the network speed to see if it is acceptable.
- Azure Data Factory (ADF) has a gateway that you can install on your local server. Then you can create a pipeline to move data from your local server up to Azure Storage. To use Data Factory with dedicated SQL pool, see Load data into dedicated SQL pool.

3. Prepare the data for loading

You might need to prepare and clean the data in your storage account before loading it into dedicated SQL pool. Data preparation can be performed while your data is in the source, as you export the data to text files, or after the data is in Azure Storage. It is easiest to work with the data as early in the process as possible.

Define external tables

Before you can load data, you need to define external tables in your data warehouse. PolyBase uses external tables to define and access the data in Azure Storage. An external table is similar to a database view. The external table contains the table schema and points to data that is stored outside the data warehouse.

Defining external tables involves specifying the data source, the format of the text files, and the table definitions. What follows are the T-SQL syntax topics that you'll need:

- CREATE EXTERNAL DATA SOURCE
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE

Format text files

Once the external objects are defined, you need to align the rows of the text files with the external table and file format definition. The data in each row of the text file must align with the table definition. To format the text files:

- If your data is coming from a non-relational source, you need to transform it into rows and columns. Whether the data is from a relational or non-relational source, the data must be transformed to align with the column definitions for the table into which you plan to load the data.
- Format data in the text file to align with the columns and data types in the SQL pool destination table. Misalignment between data types in the external text files and the data warehouse table causes rows to be rejected during the load.
- Separate fields in the text file with a terminator. Be sure to use a character or a character sequence that is not found in your source data. Use the terminator you specified with CREATE EXTERNAL FILE FORMAT.

4. Load the data into dedicated SQL pool staging tables using PolyBase

It is best practice to load data into a staging table. Staging tables allow you to handle errors without interfering with the production tables. A staging table also gives you the opportunity to use SQL pool built-in distributed query processing capabilities for data transformations before inserting the data into production tables.

Options for loading with PolyBase

To load data with PolyBase, you can use any of these loading options:

- PolyBase with T-SQL works well when your data is in Azure Blob storage or Azure Data Lake Store. It gives you the most control over the loading process, but also requires you to define external data objects. The other methods define these objects behind the scenes as you map source tables to destination tables. To orchestrate T-SQL loads, you can use Azure Data Factory, SSIS, or Azure functions.
- PolyBase with SSIS works well when your source data is in SQL Server. SSIS defines the source to destination

table mappings, and also orchestrates the load. If you already have SSIS packages, you can modify the packages to work with the new data warehouse destination.

- PolyBase with Azure Data Factory (ADF) is another orchestration tool. It defines a pipeline and schedules jobs.
- PolyBase with Azure Databricks transfers data from an Azure Synapse Analytics table to a Databricks dataframe and/or writes data from a Databricks dataframe to an Azure Synapse Analytics table using PolyBase.

Non-PolyBase loading options

If your data is not compatible with PolyBase, you can use bcp or the SQLBulkCopy API. BCP loads directly to dedicated SQL pool without going through Azure Blob storage, and is intended only for small loads. Note, the load performance of these options is slower than PolyBase.

5. Transform the data

While data is in the staging table, perform transformations that your workload requires. Then move the data into a production table.

6. Insert the data into production tables

The INSERT INTO ... SELECT statement moves the data from the staging table to the permanent table.

As you design an ETL process, try running the process on a small test sample. Try extracting 1000 rows from the table to a file, move it to Azure, and then try loading it into a staging table.

Partner loading solutions

Many of our partners have loading solutions. To find out more, see a list of our solution partners.

Next steps

For loading guidance, see Guidance for load data.

Design decisions and coding techniques for Synapse SQL features in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

In this article, you'll find a list of resources for dedicated SQL pool and serverless SQL pool functions of Synapse SQL. The recommended articles are split up into two sections: Key design decisions and development and coding techniques.

The goal of these articles is to help you develop the optimal technical approach for the Synapse SQL components within Azure Synapse Analytics.

Key design decisions

The articles below highlight concepts and design decisions for Synapse SQL development:

ARTICLE	DEDICATED SQL POOL	SERVERLESS SQL POOL
Connections	Yes	Yes
Resource classes and concurrency	Yes	No
Transactions	Yes	No
User-defined schemas	Yes	Yes
Table distribution	Yes	No
Table indexes	Yes	No
Table partitions	Yes	No
Statistics	Yes	Yes
CTAS	Yes	No
External tables	Yes	Yes
CETAS	Yes	Yes

Recommendations

Below you'll find essential articles that emphasize specific coding techniques, tips, and recommendations for development:

ARTICLE	DEDICATED SQL POOL	SERVERLESS SQL POOL
Stored procedures	Yes	Yes

ARTICLE	DEDICATED SQL POOL	SERVERLESS SQL POOL
Labels	Yes	No
Views	Yes	Yes
Temporary tables	Yes	Yes
Dynamic SQL	Yes	Yes
Looping	Yes	Yes
Group by options	Yes	No
Variable assignment	Yes	Yes

Benefits & best practices

- To learn more on which scenarios are suited for Serverless SQL pool, see Serverless SQL pool benefits article.
- Best practices for using serverless SQL pool
- Best practices for optimal performance using dedicated SQL pools

T-SQL feature support

Transact-SQL language is used in serverless SQL pool and dedicated model can reference different objects and has some differences in the set of supported features. For more information, see Transact-SQL features supported in Azure Synapse SQL article.

Next steps

For more reference information, see SQL pool T-SQL statements.

Best practices for serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 9 minutes to read • Edit Online

In this article, you'll find a collection of best practices for using serverless SQL pool. Serverless SQL pool is a resource in Azure Synapse Analytics. If you're working with a dedicated SQL pool, see Best practices for dedicated SQL pools for specific guidance.

Serverless SQL pool allows you to query files in your Azure Storage accounts. It doesn't have local storage or ingestion capabilities. All files that the queries target are external to serverless SQL pool. Everything related to reading files from storage might affect query performance.

Some generic guidelines are:

- Make sure your client applications are collocated with serverless SQL pool.
  - If you're using client applications outside Azure, make sure you're using serverless SQL pool in a region that's close to your client computer. Client application examples include Power BI Desktop, SQL Server Management Studio, and Azure Data Studio.
- Make sure the storage and serverless SQL pool are in the same region. Storage examples include Azure Data Lake Storage and Azure Cosmos DB.
- Try to optimize storage layout by using partitioning and keeping your files in the range between 100 MB and 10 GB.
- If you're returning a large number of results, make sure you're using SQL Server Management Studio or Azure Data Studio and not Azure Synapse Studio. Azure Synapse Studio is a web tool that isn't designed for large result sets.
- If you're filtering results by string column, try to use a `BIN2_UTF8` collation.
- Consider caching the results on the client side by using Power BI import mode or Azure Analysis Services, and periodically refresh them. Serverless SQL pools can't provide an interactive experience in Power BI Direct Query mode if you're using complex queries or processing a large amount of data.

Client applications and network connections

Make sure your client application is connected to the closest possible Azure Synapse workspace with the optimal connection.

- Colocate a client application with the Azure Synapse workspace. If you're using applications such as Power BI or Azure Analysis Service, make sure they're in the same region where you placed your Azure Synapse workspace. If needed, create the separate workspaces that are paired with your client applications. Placing a client application and the Azure Synapse workspace in different regions could cause bigger latency and slower streaming of results.
- If you're reading data from your on-premises application, make sure the Azure Synapse workspace is in the region that's close to your location.
- Make sure you don't have network bandwidth issues while reading a large amount of data.
- Don't use Azure Synapse Studio to return a large amount of data. Azure Synapse Studio is a web tool that uses the HTTPS protocol to transfer data. Use Azure Data Studio or SQL Server Management Studio to read a large amount of data.

Storage and content layout

Here are best practices for storage and content layout in serverless SQL pool.

Colocate your storage and serverless SQL pool

To minimize latency, colocate your Azure Storage account or Azure Cosmos DB analytic storage and your serverless SQL pool endpoint. Storage accounts and endpoints provisioned during workspace creation are located in the same region.

For optimal performance, if you access other storage accounts with serverless SQL pool, make sure they're in the same region. If they aren't in the same region, there will be increased latency for the data's network transfer between the remote region and the endpoint's region.

Azure Storage throttling

Multiple applications and services might access your storage account. Storage throttling occurs when the combined IOPS or throughput generated by applications, services, and serverless SQL pool workloads exceeds the limits of the storage account. As a result, you'll experience a significant negative effect on query performance.

When throttling is detected, serverless SQL pool has built-in handling to resolve it. Serverless SQL pool makes requests to storage at a slower pace until throttling is resolved.

> **TIP**
> For optimal query execution, don't stress the storage account with other workloads during query execution.

Prepare files for querying

If possible, you can prepare files for better performance:

- Convert large CSV and JSON files to Parquet. Parquet is a columnar format. Because it's compressed, its file sizes are smaller than CSV or JSON files that contain the same data. Serverless SQL pool skips the columns and rows that aren't needed in a query if you're reading Parquet files. Serverless SQL pool needs less time and fewer storage requests to read it.
- If a query targets a single large file, you'll benefit from splitting it into multiple smaller files.
- Try to keep your CSV file size between 100 MB and 10 GB.
- It's better to have equally sized files for a single OPENROWSET path or an external table LOCATION.
- Partition your data by storing partitions to different folders or file names. See Use filename and filepath functions to target specific partitions.

Colocate your Azure Cosmos DB analytical storage and serverless SQL pool

Make sure your Azure Cosmos DB analytical storage is placed in the same region as an Azure Synapse workspace. Cross-region queries might cause huge latencies. Use the region property in the connection string to explicitly specify the region where the analytical store is placed (see Query Azure Cosmos DB by using serverless SQL pool):

```
'account=<database account name>;database=<database name>;region=<region name>'
```

CSV optimizations

Here are best practices for using CSV files in serverless SQL pool.

Use PARSER_VERSION 2.0 to query CSV files

You can use a performance-optimized parser when you query CSV files. For details, see PARSER_VERSION.

Manually create statistics for CSV files

Serverless SQL pool relies on statistics to generate optimal query execution plans. Statistics are automatically created for columns in Parquet files when needed. At this moment, statistics aren't automatically created for columns in CSV files. Create statistics manually for columns that you use in queries, particularly those used in DISTINCT, JOIN, WHERE, ORDER BY, and GROUP BY. Check statistics in serverless SQL pool for details.

Data types

Here are best practices for using data types in serverless SQL pool.

Use appropriate data types

The data types you use in your query affect performance and concurrency. You can get better performance if you follow these guidelines:

- Use the smallest data size that can accommodate the largest possible value.
  - If the maximum character value length is 30 characters, use a character data type of length 30.
  - If all character column values are of a fixed size, use **char** or **nchar**. Otherwise, use **varchar** or **nvarchar**.
  - If the maximum integer column value is 500, use **smallint** because it's the smallest data type that can accommodate this value. You can find integer data type ranges in this article.
- If possible, use **varchar** and **char** instead of **nvarchar** and **nchar**.
  - Use the **varchar** type with some UTF8 collation if you're reading data from Parquet, Azure Cosmos DB, Delta Lake, or CSV with UTF-8 encoding.
  - Use the **varchar** type without UTF8 collation if you're reading data from CSV non-Unicode files (for example, ASCII).
  - Use the **nvarchar** type if you're reading data from a CSV UTF-16 file.
- Use integer-based data types if possible. SORT, JOIN, and GROUP BY operations complete faster on integers than on character data.
- If you're using schema inference, check inferred data types and override them explicitly with the smaller types if possible.

Check inferred data types

Schema inference helps you quickly write queries and explore data without knowing file schemas. The cost of this convenience is that inferred data types might be larger than the actual data types. This discrepancy happens when there isn't enough information in the source files to make sure the appropriate data type is used. For example, Parquet files don't contain metadata about maximum character column length. So serverless SQL pool infers it as varchar(8000).

You can use sp_describe_first_results_set to check the resulting data types of your query.

The following example shows how you can optimize inferred data types. This procedure is used to show the inferred data types:

```
EXEC sp_describe_first_result_set N'
 SELECT
        vendor_id, pickup_datetime, passenger_count
 FROM
  OPENROWSET(
         BULK ''https://sqlondemandstorage.blob.core.windows.net/parquet/taxi/*/*/*'',
         FORMAT=''PARQUET''
     ) AS nyc';
```

Here's the result set:

IS_HIDDEN	COLUMN_ORDINAL	NAME	SYSTEM_TYPE_NAME	MAX_LENGTH
0	1	vendor_id	varchar(8000)	8000
0	2	pickup_datetime	datetime2(7)	8
0	3	passenger_count	int	4

After you know the inferred data types for the query, you can specify appropriate data types:

```
SELECT
    vendorID, tpepPickupDateTime, passengerCount
FROM
 OPENROWSET(
  BULK
'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/puYear=2018/puMonth=*/*.snappy.parquet',
  FORMAT='PARQUET'
    )
 WITH (
  vendorID varchar(4), -- we used length of 4 instead of the inferred 8000
  tpepPickupDateTime datetime2,
  passengerCount int
 ) AS nyc;
```

Filter optimization

Here are best practices for using queries in serverless SQL pool.

Push wildcards to lower levels in the path

You can use wildcards in your path to query multiple files and folders. Serverless SQL pool lists files in your storage account, starting from the first asterisk (*), by using the storage API. It eliminates files that don't match the specified path. Reducing the initial list of files can improve performance if there are many files that match the specified path up to the first wildcard.

Use filename and filepath functions to target specific partitions

Data is often organized in partitions. You can instruct serverless SQL pool to query particular folders and files. Doing so reduces the number of files and the amount of data the query needs to read and process. An added bonus is that you'll achieve better performance.

For more information, read about the filename and filepath functions and see the examples for querying specific files.

> **TIP**
> Always cast the results of the filepath and filename functions to appropriate data types. If you use character data types, be sure to use the appropriate length.

Functions used for partition elimination, filepath and filename, aren't currently supported for external tables, other than those created automatically for each table created in Apache Spark for Azure Synapse Analytics.

If your stored data isn't partitioned, consider partitioning it. That way you can use these functions to optimize queries that target those files. When you query partitioned Apache Spark for Azure Synapse tables from serverless SQL pool, the query automatically targets only the necessary files.

Use proper collation to utilize predicate pushdown for character columns

Data in a Parquet file is organized in row groups. Serverless SQL pool skips row groups based on the specified predicate in the WHERE clause, which reduces IO. The result is increased query performance.

Predicate pushdown for character columns in Parquet files is supported for Latin1_General_100_BIN2_UTF8 collation only. You can specify collation for a particular column by using a WITH clause. If you don't specify this collation by using a WITH clause, the database collation is used.

Optimize repeating queries

Here are best practices for using CETAS in serverless SQL pool.

Use CETAS to enhance query performance and joins

CETAS is one of the most important features available in serverless SQL pool. CETAS is a parallel operation that creates external table metadata and exports the SELECT query results to a set of files in your storage account.

You can use CETAS to materialize frequently used parts of queries, like joined reference tables, to a new set of files. You can then join to this single external table instead of repeating common joins in multiple queries.

As CETAS generates Parquet files, statistics are automatically created when the first query targets this external table. The result is improved performance for subsequent queries targeting table generated with CETAS.

Next steps

Review the troubleshooting article for solutions to common problems. If you're working with a dedicated SQL pool rather than serverless SQL pool, see Best practices for dedicated SQL pools for specific guidance.

Best practices for dedicated SQL pools in Azure Synapse Analytics

8/19/2022 • 11 minutes to read • Edit Online

This article provides a collection of best practices to help you achieve optimal performance for dedicated SQL pools in Azure Synapse Analytics. If you're working with serverless SQL pool, see Best practices for serverless SQL pools for specific guidance. Below, you'll find basic guidance and important areas to focus on as you build your solution. Each section introduces you to a concept and then points you to more detailed articles that cover the concept in more depth.

Dedicated SQL pools loading

For dedicated SQL pools loading guidance, see Guidance for loading data.

Reduce cost with pause and scale

For more information about reducing costs through pausing and scaling, see Manage compute.

Maintain statistics

Dedicated SQL pool can be configured to automatically detect and create statistics on columns. The query plans created by the optimizer are only as good as the available statistics.

We recommend that you enable AUTO_CREATE_STATISTICS for your databases and keep the statistics updated daily or after each load to ensure that statistics on columns used in your queries are always up-to-date.

To shorten statistics maintenance time, be selective about which columns have statistics, or need the most frequent updating. For example, you might want to update date columns where new values may be added daily. Focus on having statistics for columns involved in joins, columns used in the WHERE clause, and columns found in GROUP BY.

Additional information on statistics can be found in the Manage table statistics, CREATE STATISTICS, and UPDATE STATISTICS articles.

Tune query performance

- Performance tuning with materialized views
- Performance tuning with ordered clustered columnstore index
- Performance tuning with result set caching

Group INSERT statements into batches

A one-time load to a small table with an INSERT statement such as `INSERT INTO MyLookup VALUES (1, 'Type 1')` may be the best approach depending on your needs. However, if you need to load thousands or millions of rows throughout the day, it's likely that singleton INSERTS aren't optimal.

One way to solve this issue is to develop one process that writes to a file, and then another process to periodically load this file. Refer to the INSERT article for more information.

Use PolyBase to load and export data quickly

Dedicated SQL pool supports loading and exporting data through several tools including Azure Data Factory, PolyBase, and BCP. For small amounts of data where performance isn't critical, any tool may be sufficient for your needs.

> **NOTE**
> PolyBase is the best choice when you are loading or exporting large volumes of data, or you need faster performance.

PolyBase loads can be run using CTAS or INSERT INTO. CTAS will minimize transaction logging and is the fastest way to load your data. Azure Data Factory also supports PolyBase loads and can achieve performance similar to CTAS. PolyBase supports various file formats including Gzip files.

To maximize throughput when using Gzip text files, break up files into 60 or more files to maximize parallelism of your load. For faster total throughput, consider loading data concurrently. Additional information relevant to this section is included in the following articles:

- Load data
- Guide for using PolyBase
- Dedicated SQL pool loading patterns and strategies
- Load Data with Azure Data Factory
- Move data with Azure Data Factory
- CREATE EXTERNAL FILE FORMAT
- Create table as select (CTAS)

Load then query external tables

PolyBase isn't optimal for queries. PolyBase tables for dedicated SQL pools currently only support Azure blob files and Azure Data Lake storage. These files don't have any compute resources backing them. As a result, dedicated SQL pools cannot offload this work and must read the entire file by loading it to `tempdb` so it can read the data.

If you have several queries for querying this data, it's better to load this data once and have queries use the local table. Further PolyBase guidance is included in the Guide for using PolyBase article.

Hash distribute large tables

By default, tables are Round Robin distributed. This default makes it easy for users to start creating tables without having to decide how their tables should be distributed. Round Robin tables may perform sufficiently for some workloads. But, in most cases, a distribution column provides better performance.

The most common example of a table distributed by a column outperforming a round robin table is when two large fact tables are joined.

For example, if you have an orders table distributed by order_id, and a transactions table also distributed by order_id, when you join your orders table to your transactions table on order_id, this query becomes a pass-through query. Data movement operations are then eliminated. Fewer steps mean a faster query. Less data movement also makes for faster queries.

> **TIP**
> When loading a distributed table, your incoming data shouldn't be sorted on the distribution key. Doing so will slow down your loads.

The article links provided below will give you additional details about improving performance via selecting a

distribution column. Also, you'll find information about how to define a distributed table in the WITH clause of your CREATE TABLE statement:

- Table overview
- Table distribution
- Selecting table distribution
- CREATE TABLE
- CREATE TABLE AS SELECT

Do not over-partition

While partitioning data can be effective for maintaining your data through partition switching or optimizing scans by with partition elimination, having too many partitions can slow down your queries. Often a high granularity partitioning strategy that may work well on SQL Server may not work well on dedicated SQL pool.

Having too many partitions can reduce the effectiveness of clustered columnstore indexes if each partition has fewer than 1 million rows. Dedicated SQL pools automatically partition your data into 60 databases. So, if you create a table with 100 partitions, the result will be 6000 partitions. Each workload is different, so the best advice is to experiment with partitioning to see what works best for your workload.

One option to consider is using a granularity that is lower than what you've implemented using SQL Server. For example, consider using weekly or monthly partitions instead of daily partitions.

More information about partitioning is detailed in the Table partitioning article.

Minimize transaction sizes

INSERT, UPDATE, and DELETE statements run in a transaction. When they fail, they must be rolled back. To reduce the potential for a long rollback, minimize transaction sizes whenever possible. Minimizing transaction sizes can be done by dividing INSERT, UPDATE, and DELETE statements into parts. For example, if you have an INSERT that you expect to take 1 hour, you can break up the INSERT into four parts. Each run will then be shortened to 15 minutes.

> **TIP**
> Leverage special Minimal Logging cases, like CTAS, TRUNCATE, DROP TABLE, or INSERT to empty tables to reduce rollback risk.

Another way to eliminate rollbacks is to use Metadata Only operations like partition switching for data management. For example, rather than execute a DELETE statement to delete all rows in a table where the order_date was in October of 2001, you could partition your data monthly. Then you can switch out the partition with data for an empty partition from another table (see ALTER TABLE examples).

For tables that are not partitioned, consider using a CTAS to write the data you want to keep in a table rather than using DELETE. If a CTAS takes the same amount of time, it's much safer to run since it has minimal transaction logging and can be canceled quickly if needed.

Further information on content related to this section is included in the articles below:

- Create table as select (CTAS)
- Understanding transactions
- Optimizing transactions
- Table partitioning
- TRUNCATE TABLE

- ALTER TABLE

Reduce query result sizes

Reducing query results sizes helps you avoid client-side issues caused by large query results. You can edit your query to reduce the number of rows returned. Some query generation tools allow you to add "top N" syntax to each query. You can also CETAS the query result to a temporary table and then use PolyBase export for the downlevel processing.

Use the smallest possible column size

When defining your DDL, use the smallest data type that will support your data as doing so will improve query performance. This recommendation is particularly important for CHAR and VARCHAR columns. If the longest value in a column is 25 characters, then define your column as VARCHAR(25). Avoid defining all character columns to a large default length. Additionally, define columns as VARCHAR when that is all that is needed rather than using NVARCHAR.

Please see the Table overview, Table data types, and CREATE TABLE articles for a more detailed review of essential concepts relevant to the above information.

Use temporary heap tables for transient data

When you're temporarily landing data on dedicated SQL pools, heap tables will generally make the overall process faster. If you're loading data only to stage it before running more transformations, loading the table to a heap table will be quicker than loading the data to a clustered columnstore table.

Loading data to a temp table will also load much faster than loading a table to permanent storage. Temporary tables start with a "#" and are only accessible by the session that created it. Consequently, they may only work in limited scenarios. Heap tables are defined in the WITH clause of a CREATE TABLE. If you do use a temporary table, remember to create statistics on that temporary table too.

For more information, see the Temporary tables, CREATE TABLE, and CREATE TABLE AS SELECT articles.

Optimize clustered columnstore tables

Clustered columnstore indexes are one of the most efficient ways you can store your data in dedicated SQL pool. By default, tables in dedicated SQL pool are created as Clustered ColumnStore. To get the best performance for queries on columnstore tables, having good segment quality is important. When rows are written to columnstore tables under memory pressure, columnstore segment quality may suffer.

Segment quality can be measured by the number of rows in a compressed Row Group. See the Causes of poor columnstore index quality in the Table indexes article for step-by-step instructions on detecting and improving segment quality for clustered columnstore tables.

Because high-quality columnstore segments are important, it's a good idea to use users IDs that are in the medium or large resource class for loading data. Using lower data warehouse units means you want to assign a larger resource class to your loading user.

Columnstore tables generally won't push data into a compressed columnstore segment until there are more than 1 million rows per table. Each dedicated SQL pool table is partitioned into 60 tables. As such, columnstore tables won't benefit a query unless the table has more than 60 million rows.

> **TIP**
> For tables with less than 60 million rows, having a columnstore index may not be the optimal solution.

If you partition your data, each partition will need to have 1 million rows to benefit from a clustered columnstore index. For a table with 100 partitions, it needs to have at least 6 billion rows to benefit from a clustered columns store (60 distributions *100 partitions* 1 million rows).

If your table doesn't have 6 billion rows, you have two main options. Either reduce the number of partitions or consider using a heap table instead. It also may be worth experimenting to see if better performance can be gained by using a heap table with secondary indexes rather than a columnstore table.

When querying a columnstore table, queries will run faster if you select only the columns you need. Further information on table and columnstore indexes and can be found in the articles below:

- Table indexes
- Columnstore indexes guide
- Rebuilding columnstore indexes
- Performance tuning with ordered clustered columnstore index

Use larger resource class to improve query performance

SQL pools use resource groups as a way to allocate memory to queries. Initially, all users are assigned to the small resource class, which grants 100 MB of memory per distribution. There are always 60 distributions. Each distribution is given a minimum of 100 MB. The total system-wide memory allocation is 6,000 MB, or just under 6 GB.

Certain queries, like large joins or loads to clustered columnstore tables, will benefit from larger memory allocations. Some queries, such as pure scans, will see no benefit. Utilizing larger resource classes impacts concurrency. So, you'll want to keep these facts in mind before moving all of your users to a large resource class.

For additional information on resource classes, refer to the Resource classes for workload management article.

Use smaller resource class to increase concurrency

If you notice a long delay in user queries, your users might be running in larger resource classes. This scenario promotes the consumption of concurrency slots, which can cause other queries to queue up. To determine if users queries are queued, run `SELECT * FROM sys.dm_pdw_waits` to see if any rows are returned.

The Resource classes for workload management and sys.dm_pdw_waits articles will provide you with more information.

Use DMVs to monitor and optimize your queries

Dedicated SQL pools have several DMVs that can be used to monitor query execution. The monitoring article below walks you through step-by-step instructions on how to view details of an executing query. To quickly find queries in these DMVs, using the LABEL option with your queries can help. For additional detailed information, please see the articles included in the list below:

- Monitor your workload using DMVs
- LABEL
- OPTION
- sys.dm_exec_sessions
- sys.dm_pdw_exec_requests
- sys.dm_pdw_request_steps
- sys.dm_pdw_sql_requests

- sys.dm_pdw_dms_workers
- DBCC PDW_SHOWEXECUTIONPLAN
- sys.dm_pdw_waits

Next steps

Also see the Troubleshooting article for common issues and solutions.

If you need information not provided in this article, search the Microsoft Q&A question page for Azure Synapse is a place for you to pose questions to other users and to the Azure Synapse Analytics Product Group.

We actively monitor this forum to ensure that your questions are answered either by another user or one of us. If you prefer to ask your questions on Stack Overflow, we also have an Azure Synapse Analytics Stack Overflow Forum.

Distribution Advisor in Azure Synapse SQL

8/19/2022 • 4 minutes to read • Edit Online

Applies to: Azure Synapse Analytics dedicated SQL pools (formerly SQL DW)

In Azure Synapse SQL, each table is distributed using the strategy chosen by the customer (Round Robin, Hash Distributed, Replicated). The chosen distribution strategy can affect query performance substantially.

The Distribution Advisor (DA) feature of Azure Synapse SQL analyzes customer queries and recommends the best distribution strategies for tables to improve query performance. Queries to be considered by the advisor can be provided by the customer or pulled from historic queries available in DMV.

> **NOTE**
> Distribution Advisor is currently in preview for Azure Synapse Analytics. Preview features are meant for testing only and should not be used on production instances or production data. As a preview feature, Distribution Advisor is subject to undergo changes in behavior or functionality. Please also keep a copy of your test data if the data is important. Distribution Advisor does not support Multi-Column distributed tables.

Prerequisites

- Execute the T-SQL statement `SELECT @@version` to ensure that your Azure Synapse Analytics dedicated SQL pool is version 10.0.15669 or higher. If your version is lower, a new version should automatically reach your provisioned dedicated SQL pools during their maintenance cycle.
- Ensure that statistics are available and up-to-date before running the advisor. For more details, Manage table statistics, CREATE STATISTICS, and UPDATE STATISTICS articles for more details on statistics.

Analyze workload and generate distribution recommendations

The follow tutorial explains the sample use case for using the Distribution Advisor feature to analyze customer queries and recommend the best distribution strategies.

Distribution Advisor only analyzes queries run on user tables.

1. Create Distribution Advisor stored procedures

To run the advisor easily, create two new stored procedures in the database. Run the CreateDistributionAdvisor_PublicPreview script available for download from GitHub:

COMMAND	DESCRIPTION
`dbo.write_dist_recommendation`	Defines queries that DA will analyze on. You can provide queries manually, or read from up to 100 past queries from the actual workloads in sys.dm_pdw_exec_requests.
`dbo.read_dist_recommendation`	Runs the advisor and generates recommendations.

Here is an example of how you could run the advisor.

2a. Run the advisor on past workload in DMV

Run the following commands to read up to the last 100 queries in the workload for analysis and distribution

recommendations:

```
EXEC dbo.write_dist_recommendation <Number of Queries max 100>, NULL
go
EXEC dbo.read_dist_recommendation;
go
```

To see which queries were analyzed by DA, run the e2e_queries_used_for_recommendations.sql script available for download from GitHub.

2b. Run the advisor on selected queries

The first parameter in `dbo.write_dist_recommendation` should be set to `0`, and the second parameter is a semi-colon separated list of up to 100 queries that DA will analyze. In the below example, we want to see the distribution recommendation for two statements separated by semicolons, `select count (*) from t1;` and `select * from t1 join t2 on t1.a1 = t2.a1;`.

```
EXEC dbo.write_dist_recommendation 0, 'select count (*) from t1; select * from t1 join t2 on t1.a1 = t2.a1;'
go
EXEC dbo.read_dist_recommendation;
go
```

3. View recommendations

The `dbo.read_dist_recommendation` system stored procedure will return recommendations in the following format when execution is completed:

COLUMN NAME	DESCRIPTION
Table_name	The table that DA analyzed. One line per table regardless of change in recommendation.
Current_Distribution	Current table distribution strategy.
Recommended_Distribution	Recommended distribution. This can be the same as `Current_Distribution` if there is no change recommended.
Distribution_Change_Command	A CTAS T-SQL command to implement the recommendation.

4. Implement the advice

- Run the CTAS command provided by Distribution Advisor to create new tables with the recommended distribution strategy.
- Modify queries to run on new tables.
- Execute queries on old and new tables to compare for performance improvements.

Troubleshooting

This section contains common troubleshooting scenarios and common mistakes that you may encounter.

1. Stale state from a previous run of the advisor

1a. Symptom:

You see this error message upon running the advisor:

```
Msg 110813, Level 16, State 1, Line 1
Calling GetLastScalarResult() before executing scalar subquery.
```

1b. Mitigation:

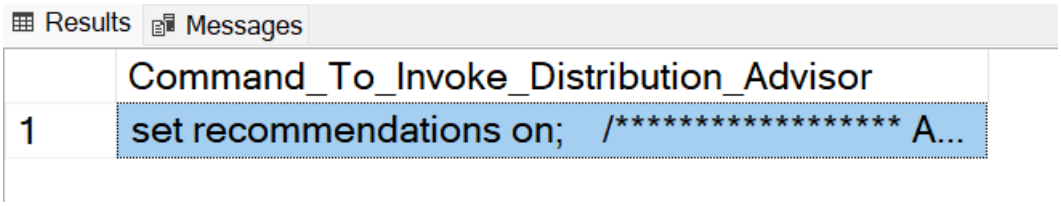
- Verify that you are using single quotes '' to run the advisor on select queries.
- Start a new session in SSMS and run the advisor.

2. Errors during running the advisor

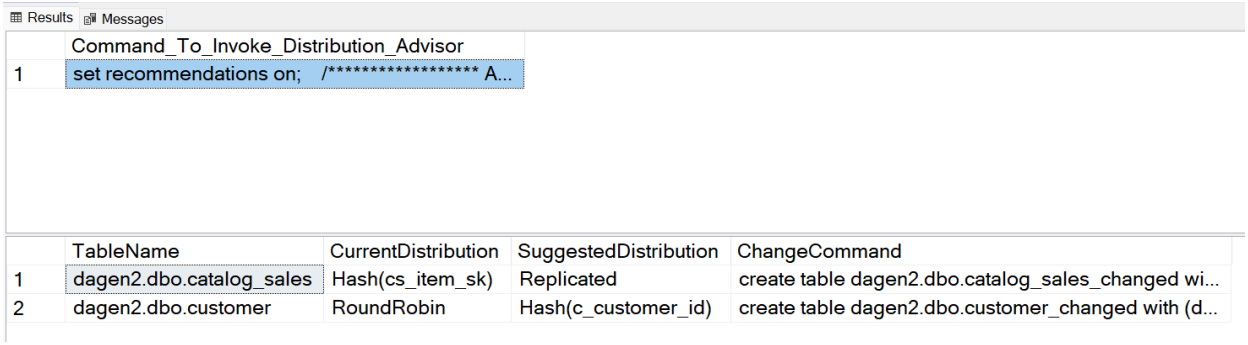
2a. Symptom:

The 'result' pane shows `CommandToInvokeAdvisorString` below but does not show the `RecommendationOutput` below.

For example, you see only the `Command_to_Invoke_Distribution_Advisor` resultset.



But not the second resultset containing the table change T-SQL commands:



2b. Mitigation:

- Check the output of `CommandToInvokeAdvisorString` above.
- Remove queries that may not be valid anymore which may have been added here from either the hand-selected queries or from the DMV by editing `WHERE` clause in: Queries Considered by DA.

3. Error during post-processing of recommendation output

3a. Symptom:

You see the following error message.

```
Invalid length parameter passed to the LEFT or SUBSTRING function.
```

3b. Mitigation:

Ensure that you have the most up to date version of the stored procedure from GitHub:

- e2e_queries_used_for_recommendations.sql script available for download from GitHub
- CreateDistributionAdvisor_PublicPreview.sql script available for download from GitHub

Azure Synapse product group feedback

If you need information not provided in this article, search the Microsoft Q&A question page for Azure Synapse is a place for you to pose questions to other users and to the Azure Synapse Analytics Product Group.

We actively monitor this forum to ensure that your questions are answered either by another user or one of us.

If you prefer to ask your questions on Stack Overflow, we also have an Azure Synapse Analytics Stack Overflow Forum.

For feature requests, use the Azure Synapse Analytics Feedback page. Adding your requests or up-voting other requests helps us to focus on the most in-demand features.

Next steps

- Loading data to dedicated SQL pool
- Data loading strategies for dedicated SQL pool in Azure Synapse Analytics.
- Dedicated SQL pool (formerly SQL DW) architecture in Azure Synapse Analytics
- Cheat sheet for dedicated SQL pool (formerly SQL DW) in Azure Synapse Analytics

Performance tuning with materialized views using dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 11 minutes to read • Edit Online

In dedicated SQL pool, materialized views provide a low maintenance method for complex analytical queries to get fast performance without any query change. This article discusses the general guidance on using materialized views.

Materialized views vs. standard views

SQL pool supports both standard and materialized views. Both are virtual tables created with SELECT expressions and presented to queries as logical tables. Views reveal the complexity of common data computation and add an abstraction layer to computation changes so there's no need to rewrite queries.

A standard view computes its data each time when the view is used. There's no data stored on disk. People typically use standard views as a tool that helps organize the logical objects and queries in a database. To use a standard view, a query needs to make direct reference to it.

A materialized view pre-computes, stores, and maintains its data in dedicated SQL pool just like a table. Recomputation isn't needed each time a materialized view is used. That's why queries that use all or a subset of the data in materialized views can gain faster performance. Even better, queries can use a materialized view without making direct reference to it, so there's no need to change application code.

Most of the standard view requirements still apply to a materialized view. For details on the materialized view syntax and other requirements, refer to CREATE MATERIALIZED VIEW AS SELECT.

COMPARISON	VIEW	MATERIALIZED VIEW
View definition	Stored in Azure data warehouse.	Stored in Azure data warehouse.
View content	Generated each time when the view is used.	Pre-processed and stored in Azure data warehouse during view creation. Updated as data is added to the underlying tables.
Data refresh	Always updated	Always updated
Speed to retrieve view data from complex queries	Slow	Fast
Extra storage	No	Yes
Syntax	CREATE VIEW	CREATE MATERIALIZED VIEW AS SELECT

Benefits of materialized views

A properly designed materialized view provides the following benefits:

- Reduced execution time for complex queries with JOINs and aggregate functions. The more complex the query, the higher the potential for execution-time saving. The most benefit is gained when a query's

computation cost is high and the resulting data set is small.

- The query optimizer in dedicated SQL pool can automatically use deployed materialized views to improve query execution plans. This process is transparent to users providing faster query performance and doesn't require queries to make direct reference to the materialized views.
- Requires low maintenance on the views. A materialized view stores data in two places, a clustered columnstore index for the initial data at the view creation time, and a delta store for the incremental data changes. All data changes from the base tables are automatically added to the delta store in a synchronous manner. A background process (tuple mover) periodically moves the data from the delta store to the view's columnstore index. This design allows querying materialized views to return the same data as directly querying the base tables.
- The data in a materialized view can be distributed differently from the base tables.
- Data in materialized views gets the same high availability and resiliency benefits as data in regular tables.

As compared to other data warehouse providers, the materialized views implemented in dedicated SQL pool also provide the following additional benefits:

- Automatic and synchronous data refresh with data changes in base tables. No user action is required.
- Broad aggregate function support. See CREATE MATERIALIZED VIEW AS SELECT (Transact-SQL).
- The support for query-specific materialized view recommendation. See EXPLAIN (Transact-SQL).

Common scenarios

Materialized views are typically used in following scenarios:

Need to improve the performance of complex analytical queries against large data in size

Complex analytical queries typically use more aggregation functions and table joins, causing more compute-heavy operations such as shuffles and joins in query execution. That's why those queries take longer to complete, particularly on large tables.

Users can create materialized views for the data returned from the common computations of queries, so there's no recomputation required when this data is needed by queries, allowing lower compute cost and faster query response.

Need faster performance with no or minimum query changes

Schema and query changes in data warehouses are typically kept to a minimum to support regular ETL operations and reporting. People can use materialized views for query performance tuning if the cost incurred by the views can be offset by the gain in query performance.

In comparison to other tuning options, such as scaling and statistics management, it's a much less impactful production change to create and maintain a materialized view and its potential performance gain is also higher.

- Creating or maintaining materialized views does not impact the queries running against the base tables.
- The query optimizer can automatically use the deployed materialized views without direct view reference in a query. This capability reduces the need for query change in performance tuning.

Need different data distribution strategy for faster query performance

Azure data warehouse is a distributed and massively parallel processing (MPP) system.

Synapse SQL is a distributed query system that enables enterprises to implement data warehousing and data virtualization scenarios using standard T-SQL experiences familiar to data engineers. It also expands the capabilities of SQL to address streaming and machine learning scenarios. Data in a data warehouse table is distributed across 60 nodes using one of three distribution strategies (hash, round_robin, or replicated).

The data distribution is specified at the table creation time and remains unchanged until the table is dropped. Materialized view being a virtual table on disk supports hash and round_robin data distributions. Users can choose a data distribution that's different from the base tables but optimal for the performance of queries that frequently use the views.

Design guidance

Here is the general guidance on using materialized views to improve query performance:

Design for your workload

Before you begin to create materialized views, it's important to have a deep understanding of your workload in terms of query patterns, importance, frequency, and the size of resulting data.

Users can run EXPLAIN WITH_RECOMMENDATIONS <SQL_statement> for the materialized views recommended by the query optimizer. Since these recommendations are query-specific, a materialized view that benefits a single query may not be optimal for other queries in the same workload.

Evaluate these recommendations with your workload needs in mind. The ideal materialized views are those that benefit the workload's performance.

Be aware of the tradeoff between faster queries and the cost

For each materialized view, there's a data storage cost and a cost for maintaining the view. As data changes in the base tables, the size of the materialized view increases and its physical structure also changes.

To avoid query performance degradation, each materialized view is maintained separately by the data warehouse engine, including moving rows from delta store to the columnstore index segments and consolidating data changes.

The maintenance workload climbs higher when the number of materialized views and base table changes increase. Users should check if the cost incurred from all materialized views can be offset by the query performance gain.

You can run this query for the list of materialized view in a database:

```
SELECT V.name as materialized_view, V.object_id
FROM sys.views V
JOIN sys.indexes I ON V.object_id= I.object_id AND I.index_id < 2;
```

Options to reduce the number of materialized views:

- Identify common data sets frequently used by the complex queries in your workload. Create materialized views to store those data sets so the optimizer can use them as building blocks when creating execution plans.
- Drop the materialized views that have low usage or are no longer needed. A disabled materialized view is not maintained but it still incurs storage cost.
- Combine materialized views created on the same or similar base tables even if their data doesn't overlap. Combining materialized views could result in a larger view in size than the sum of the separate views, however the view maintenance cost should reduce. For example:

```
-- Query 1 would benefit from having a materialized view created with this SELECT statement
SELECT A, SUM(B)
FROM T
GROUP BY A

-- Query 2 would benefit from having a materialized view created with this SELECT statement
SELECT C, SUM(D)
FROM T
GROUP BY C

-- You could create a single mateiralized view of this form
SELECT A, C, SUM(B), SUM(D)
FROM T
GROUP BY A, C
```

Not all performance tuning requires query change

The data warehouse optimizer can automatically use deployed materialized views to improve query performance. This support is applied transparently to queries that don't reference the views and to queries that use aggregates unsupported in materialized views creation. No query change is needed. You can check a query's estimated execution plan to confirm if a materialized view is used.

Monitor materialized views

A materialized view is stored in the data warehouse just like a table with clustered columnstore index (CCI). Reading data from a materialized view includes scanning the index and applying changes from the delta store. When the number of rows in the delta store is too high, resolving a query from a materialized view can take longer than directly querying the base tables.

To avoid query performance degradation, it's a good practice to run DBCC PDW_SHOWMATERIALIZEDVIEWOVERHEAD to monitor the view's overhead_ratio (total_rows / base_view_row). If the overhead_ratio is too high, consider rebuilding the materialized view so all rows in the delta store are moved to the columnstore index.

Materialized view and result set caching

These two features are introduced in dedicated SQL pool around the same time for query performance tuning. Result set caching is used for achieving high concurrency and fast response times from repetitive queries against static data.

To use the cached result, the form of the cache requesting query must match with the query that produced the cache. Additionally, the cached result must apply to the entire query.

Materialized views allow data changes in the base tables. Data in materialized views can be applied to a piece of a query. This support allows the same materialized views to be used by different queries that share some computation for faster performance.

Example

This example uses a TPCDS-like query that finds customers who spend more money via catalog than in stores. It also identifies the preferred customers and their country/region of origin. The query involves selecting TOP 100 records from the UNION of three sub-SELECT statements involving SUM() and GROUP BY.

```
WITH year_total AS (
SELECT c_customer_id customer_id
       ,c_first_name customer_first_name
       ,c_last_name customer_last_name
       ,c_preferred_cust_flag customer_preferred_cust_flag
       ,c_birth_country customer_birth_country
```

```
       ,c_birth_country customer_birth_country
       ,c_login customer_login
       ,c_email_address customer_email_address
       ,d_year dyear
       ,sum(isnull(ss_ext_list_price-ss_ext_wholesale_cost-ss_ext_discount_amt+ss_ext_sales_price, 0)/2)
year_total
       ,'s' sale_type
FROM customer
     ,store_sales
     ,date_dim
WHERE c_customer_sk = ss_customer_sk
   AND ss_sold_date_sk = d_date_sk
GROUP BY c_customer_id
         ,c_first_name
         ,c_last_name
         ,c_preferred_cust_flag
         ,c_birth_country
         ,c_login
         ,c_email_address
         ,d_year
UNION ALL
SELECT c_customer_id customer_id
       ,c_first_name customer_first_name
       ,c_last_name customer_last_name
       ,c_preferred_cust_flag customer_preferred_cust_flag
       ,c_birth_country customer_birth_country
       ,c_login customer_login
       ,c_email_address customer_email_address
       ,d_year dyear
       ,sum(isnull(cs_ext_list_price-cs_ext_wholesale_cost-cs_ext_discount_amt+cs_ext_sales_price, 0)/2)
year_total
       ,'c' sale_type
FROM customer
     ,catalog_sales
     ,date_dim
WHERE c_customer_sk = cs_bill_customer_sk
   AND cs_sold_date_sk = d_date_sk
GROUP BY c_customer_id
         ,c_first_name
         ,c_last_name
         ,c_preferred_cust_flag
         ,c_birth_country
         ,c_login
         ,c_email_address
         ,d_year
UNION ALL
SELECT c_customer_id customer_id
       ,c_first_name customer_first_name
       ,c_last_name customer_last_name
       ,c_preferred_cust_flag customer_preferred_cust_flag
       ,c_birth_country customer_birth_country
       ,c_login customer_login
       ,c_email_address customer_email_address
       ,d_year dyear
       ,sum(isnull(ws_ext_list_price-ws_ext_wholesale_cost-ws_ext_discount_amt+ws_ext_sales_price, 0)/2)
year_total
       ,'w' sale_type
FROM customer
     ,web_sales
     ,date_dim
WHERE c_customer_sk = ws_bill_customer_sk
   AND ws_sold_date_sk = d_date_sk
GROUP BY c_customer_id
         ,c_first_name
         ,c_last_name
         ,c_preferred_cust_flag
         ,c_birth_country
         ,c_login
         ,c_email_address
```

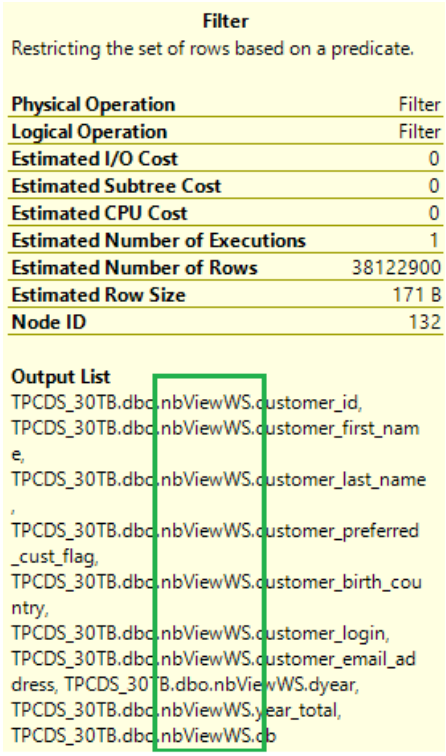
```
         ,d_year
         )
  SELECT TOP 100
                  t_s_secyear.customer_id
                 ,t_s_secyear.customer_first_name
                 ,t_s_secyear.customer_last_name
                 ,t_s_secyear.customer_birth_country
FROM year_total t_s_firstyear
     ,year_total t_s_secyear
     ,year_total t_c_firstyear
     ,year_total t_c_secyear
     ,year_total t_w_firstyear
     ,year_total t_w_secyear
WHERE t_s_secyear.customer_id = t_s_firstyear.customer_id
   AND t_s_firstyear.customer_id = t_c_secyear.customer_id
   AND t_s_firstyear.customer_id = t_c_firstyear.customer_id
   AND t_s_firstyear.customer_id = t_w_firstyear.customer_id
   AND t_s_firstyear.customer_id = t_w_secyear.customer_id
   AND t_s_firstyear.sale_type = 's'
   AND t_c_firstyear.sale_type = 'c'
   AND t_w_firstyear.sale_type = 'w'
   AND t_s_secyear.sale_type = 's'
   AND t_c_secyear.sale_type = 'c'
   AND t_w_secyear.sale_type = 'w'
   AND t_s_firstyear.dyear+0 =  1999
   AND t_s_secyear.dyear+0 = 1999+1
   AND t_c_firstyear.dyear+0 =  1999
   AND t_c_secyear.dyear+0 =  1999+1
   AND t_w_firstyear.dyear+0 = 1999
   AND t_w_secyear.dyear+0 = 1999+1
   AND t_s_firstyear.year_total > 0
   AND t_c_firstyear.year_total > 0
   AND t_w_firstyear.year_total > 0
   AND CASE WHEN t_c_firstyear.year_total > 0 THEN t_c_secyear.year_total / t_c_firstyear.year_total ELSE
NULL END
           > CASE WHEN t_s_firstyear.year_total > 0 THEN t_s_secyear.year_total / t_s_firstyear.year_total
ELSE NULL END
   AND CASE WHEN t_c_firstyear.year_total > 0 THEN t_c_secyear.year_total / t_c_firstyear.year_total ELSE
NULL END
           > CASE WHEN t_w_firstyear.year_total > 0 THEN t_w_secyear.year_total / t_w_firstyear.year_total
ELSE NULL END
ORDER BY t_s_secyear.customer_id
         ,t_s_secyear.customer_first_name
         ,t_s_secyear.customer_last_name
         ,t_s_secyear.customer_birth_country
OPTION ( LABEL = 'Query04-af359846-253-3');
```

Check the query's estimated execution plan. There are 18 shuffles and 17 joins operations, which take more time to execute. Now let's create one materialized view for each of the three sub-SELECT statements.

```
CREATE materialized view nbViewSS WITH (DISTRIBUTION=HASH(customer_id)) AS
SELECT c_customer_id customer_id
       ,c_first_name customer_first_name
       ,c_last_name customer_last_name
       ,c_preferred_cust_flag customer_preferred_cust_flag
       ,c_birth_country customer_birth_country
       ,c_login customer_login
       ,c_email_address customer_email_address
       ,d_year dyear
       ,sum(isnull(ss_ext_list_price-ss_ext_wholesale_cost-ss_ext_discount_amt+ss_ext_sales_price, 0)/2)
year_total
          , count_big(*) AS cb
FROM dbo.customer
     ,dbo.store_sales
     ,dbo.date_dim
WHERE c_customer_sk = ss_customer_sk
   AND ss_sold_date_sk = d_date_sk
```

```
GROUP BY c_customer_id
         ,c_first_name
         ,c_last_name
         ,c_preferred_cust_flag
         ,c_birth_country
         ,c_login
         ,c_email_address
         ,d_year
GO
CREATE materialized view nbViewCS WITH (DISTRIBUTION=HASH(customer_id)) AS
SELECT c_customer_id customer_id
       ,c_first_name customer_first_name
       ,c_last_name customer_last_name
       ,c_preferred_cust_flag customer_preferred_cust_flag
       ,c_birth_country customer_birth_country
       ,c_login customer_login
       ,c_email_address customer_email_address
       ,d_year dyear
       ,sum(isnull(cs_ext_list_price-cs_ext_wholesale_cost-cs_ext_discount_amt+cs_ext_sales_price, 0)/2)
year_total
          , count_big(*) as cb
FROM dbo.customer
     ,dbo.catalog_sales
     ,dbo.date_dim
WHERE c_customer_sk = cs_bill_customer_sk
   AND cs_sold_date_sk = d_date_sk
GROUP BY c_customer_id
         ,c_first_name
         ,c_last_name
         ,c_preferred_cust_flag
         ,c_birth_country
         ,c_login
         ,c_email_address
         ,d_year

GO
CREATE materialized view nbViewWS WITH (DISTRIBUTION=HASH(customer_id)) AS
SELECT c_customer_id customer_id
       ,c_first_name customer_first_name
       ,c_last_name customer_last_name
       ,c_preferred_cust_flag customer_preferred_cust_flag
       ,c_birth_country customer_birth_country
       ,c_login customer_login
       ,c_email_address customer_email_address
       ,d_year dyear
       ,sum(isnull(ws_ext_list_price-ws_ext_wholesale_cost-ws_ext_discount_amt+ws_ext_sales_price, 0)/2)
year_total
          , count_big(*) AS cb
FROM dbo.customer
     ,dbo.web_sales
     ,dbo.date_dim
WHERE c_customer_sk = ws_bill_customer_sk
   AND ws_sold_date_sk = d_date_sk
GROUP BY c_customer_id
         ,c_first_name
         ,c_last_name
         ,c_preferred_cust_flag
         ,c_birth_country
         ,c_login
         ,c_email_address
         ,d_year
```

Check the execution plan of the original query again. Now the number of joins changes from 17 to 5 and there's no shuffle anymore. Select the Filter operation icon in the plan. Its Output List shows the data is read from the materialized views instead of base tables.



With materialized views, the same query runs much faster without any code change.

Next steps

For more development tips, see Synapse SQL development overview.

Performance tuning with ordered clustered columnstore index

8/19/2022 • 7 minutes to read • Edit Online

Applies to: Azure Synapse Analytics dedicated SQL pools, SQL Server 2022 (16.x) and later

When users query a columnstore table in dedicated SQL pool, the optimizer checks the minimum and maximum values stored in each segment. Segments that are outside the bounds of the query predicate aren't read from disk to memory. A query can finish faster if the number of segments to read and their total size are small.

Ordered vs. non-ordered clustered columnstore index

By default, for each table created without an index option, an internal component (index builder) creates a non-ordered clustered columnstore index (CCI) on it. Data in each column is compressed into a separate CCI rowgroup segment. There's metadata on each segment's value range, so segments that are outside the bounds of the query predicate aren't read from disk during query execution. CCI offers the highest level of data compression and reduces the size of segments to read so queries can run faster. However, because the index builder doesn't sort data before compressing them into segments, segments with overlapping value ranges could occur, causing queries to read more segments from disk and take longer to finish.

When creating an ordered CCI, the dedicated SQL pool engine sorts the existing data in memory by the order key(s) before the index builder compresses them into index segments. With sorted data, segment overlapping is reduced allowing queries to have a more efficient segment elimination and thus faster performance because the number of segments to read from disk is smaller. If all data can be sorted in memory at once, then segment overlapping can be avoided. Due to large tables in data warehouses, this scenario doesn't happen often.

To check the segment ranges for a column, run the following command with your table name and column name:

```
SELECT o.name, pnp.index_id,
cls.row_count, pnp.data_compression_desc,
pnp.pdw_node_id, pnp.distribution_id, cls.segment_id,
cls.column_id,
cls.min_data_id, cls.max_data_id,
cls.max_data_id-cls.min_data_id as difference
FROM sys.pdw_nodes_partitions AS pnp
   JOIN sys.pdw_nodes_tables AS Ntables ON pnp.object_id = NTables.object_id AND pnp.pdw_node_id =
NTables.pdw_node_id
   JOIN sys.pdw_table_mappings AS Tmap  ON NTables.name = TMap.physical_name AND
substring(TMap.physical_name,40, 10) = pnp.distribution_id
   JOIN sys.objects AS o ON TMap.object_id = o.object_id
   JOIN sys.pdw_nodes_column_store_segments AS cls ON pnp.partition_id = cls.partition_id AND
pnp.distribution_id  = cls.distribution_id
JOIN sys.columns as cols ON o.object_id = cols.object_id AND cls.column_id = cols.column_id
WHERE o.name = '<Table Name>' and cols.name = '<Column Name>'  and TMap.physical_name  not like '%HdTable%'
ORDER BY o.name, pnp.distribution_id, cls.min_data_id;
```

> **NOTE**
> In an ordered CCI table, the new data resulting from the same batch of DML or data loading operations are sorted within that batch, there is no global sorting across all data in the table. Users can REBUILD the ordered CCI to sort all data in the table. In dedicated SQL pool, the columnstore index REBUILD is an offline operation. For a partitioned table, the REBUILD is done one partition at a time. Data in the partition that is being rebuilt is "offline" and unavailable until the REBUILD is complete for that partition.

Query performance

A query's performance gain from an ordered CCI depends on the query patterns, the size of data, how well the data is sorted, the physical structure of segments, and the DWU and resource class chosen for the query execution. Users should review all these factors before choosing the ordering columns when designing an ordered CCI table.

Queries with all these patterns typically run faster with ordered CCI.

1. The queries have equality, inequality, or range predicates
2. The predicate columns and the ordered CCI columns are the same.

In this example, table T1 has a clustered columnstore index ordered in the sequence of Col_C, Col_B, and Col_A.

```
CREATE CLUSTERED COLUMNSTORE INDEX MyOrderedCCI ON  T1
ORDER (Col_C, Col_B, Col_A);
```

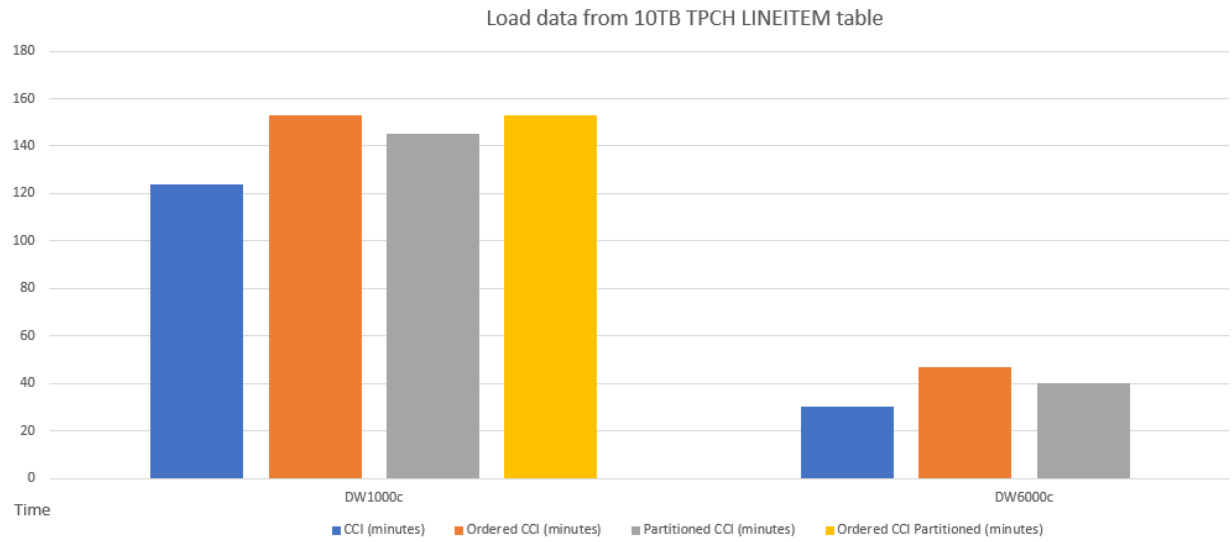
The performance of query 1 and query 2 can benefit most from ordered CCI than the other queries as they reference all the ordered CCI columns.

```
-- Query #1:

SELECT * FROM T1 WHERE Col_C = 'c' AND Col_B = 'b' AND Col_A = 'a';

-- Query #2

SELECT * FROM T1 WHERE Col_B = 'b' AND Col_C = 'c' AND Col_A = 'a';

-- Query #3
SELECT * FROM T1 WHERE Col_B = 'b' AND Col_A = 'a';

-- Query #4
SELECT * FROM T1 WHERE Col_A = 'a' AND Col_C = 'c';
```

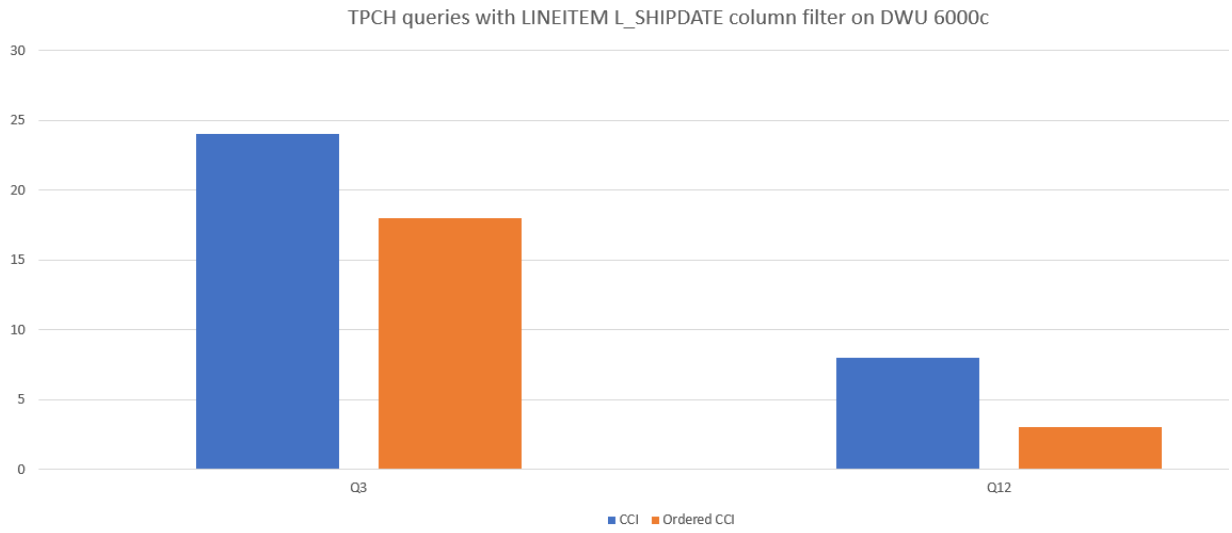
Data loading performance

The performance of data loading into an ordered CCI table is similar to a partitioned table. Loading data into an ordered CCI table can take longer than a non-ordered CCI table because of the data sorting operation, however queries can run faster afterwards with ordered CCI.

Here is an example performance comparison of loading data into tables with different schemas.



Here is an example query performance comparison between CCI and ordered CCI.



Reduce segment overlapping

The number of overlapping segments depends on the size of data to sort, the available memory, and the maximum degree of parallelism (MAXDOP) setting during ordered CCI creation. Below are options to reduce segment overlapping when creating ordered CCI.

- Use `xlargerc` resource class on a higher DWU to allow more memory for data sorting before the index builder compresses the data into segments. Once in an index segment, the physical location of the data cannot be changed. There's no data sorting within a segment or across segments.
- Create ordered CCI with `OPTION (MAXDOP = 1)`. Each thread used for ordered CCI creation works on a subset of data and sorts it locally. There's no global sorting across data sorted by different threads. Using parallel threads can reduce the time to create an ordered CCI but will generate more overlapping segments than using a single thread. Currently, the MAXDOP option is only supported in creating an ordered CCI table using CREATE TABLE AS SELECT command. Creating an ordered CCI via CREATE INDEX or CREATE TABLE commands does not support the MAXDOP option. For example:

```
CREATE TABLE Table1 WITH (DISTRIBUTION = HASH(c1), CLUSTERED COLUMNSTORE INDEX ORDER(c1) )
AS SELECT * FROM ExampleTable
OPTION (MAXDOP 1);
```

- Pre-sort the data by the sort key(s) before loading them into tables.

Here is an example of an ordered CCI table distribution that has zero segment overlapping following above recommendations. The ordered CCI table is created in a DWU1000c database via CTAS from a 20-GB heap table using MAXDOP 1 and `xlargerc`. The CCI is ordered on a BIGINT column with no duplicates.

Distribution_id	Segment_id	Min_data_id	Max_data_id	Min_Max_Difference	Row_count
1	0	10241	57382203	57371962	1048576
1	1	57382204	118528606	61146402	1048576
1	2	118528607	177460494	58931887	1048576
1	3	177460495	243567533	66107038	1048576
1	4	243567534	306639061	63071527	1048576
1	5	306639062	368575285	61936223	1048576
1	6	368575286	430037457	61462171	1048576
1	7	430037458	499756489	69719031	1048576
1	8	499756490	560819016	61062526	1048576
1	9	560819017	629362726	68543709	1048576
1	10	629362727	693056367	63693640	1048576
1	11	693056368	755671165	62614797	1048576
1	12	755671166	818307227	62636061	1048576
1	13	818307228	880813709	62506481	1048576
1	14	880813710	941012631	60198921	1048576
1	15	941012632	1009551689	68539057	1048576
1	16	1009551690	1048530944	38979254	704512

Create ordered CCI on large tables

Creating an ordered CCI is an offline operation. For tables with no partitions, the data won't be accessible to users until the ordered CCI creation process completes. For partitioned tables, since the engine creates the ordered CCI partition by partition, users can still access the data in partitions where ordered CCI creation isn't in process. You can use this option to minimize the downtime during ordered CCI creation on large tables:

1. Create partitions on the target large table (called `Table_A`).
2. Create an empty ordered CCI table (called `Table_B`) with the same table and partition schema as `Table_A`.
3. Switch one partition from `Table_A` to `Table_B`.
4. Run `ALTER INDEX <Ordered_CCI_Index> ON <Table_B> REBUILD PARTITION = <Partition_ID>` to rebuild the switched-in partition on `Table_B`.
5. Repeat step 3 and 4 for each partition in `Table_A`.
6. Once all partitions are switched from `Table_A` to `Table_B` and have been rebuilt, drop `Table_A`, and rename `Table_B` to `Table_A`.

> **TIP**
>
> For a dedicated SQL pool table with an ordered CCI, ALTER INDEX REBUILD will re-sort the data using `tempdb`. Monitor `tempdb` during rebuild operations. If you need more `tempdb` space, scale up the pool. Scale back down once the index rebuild is complete.
>
> For a dedicated SQL pool table with an ordered CCI, ALTER INDEX REORGANIZE does not re-sort the data. To resort data, use ALTER INDEX REBUILD.
>
> For more information on ordered CCI maintenance, see Optimizing clustered columnstore indexes.

Feature differences in SQL Server 2022 capabilities

SQL Server 2022 (16.x) introduced ordered clustered columnstore indexes similar to the feature in Azure Synapse dedicated SQL pools.

- Currently, only SQL Server 2022 (16.x) and later support clustered columnstore enhanced rowgroup elimination capabilities for string, binary, and guid data types, and the datetimeoffset data type for scale greater than two.
- Currently, only SQL Server 2022 (16.x) supports clustered columnstore rowgroup elimination for the prefix

of `LIKE` predicates.

For more information, see What's New in Columnstore Indexes.

Examples

A. To check for ordered columns and order ordinal:

```
SELECT object_name(c.object_id) table_name, c.name column_name, i.column_store_order_ordinal
FROM sys.index_columns i
JOIN sys.columns c ON i.object_id = c.object_id AND c.column_id = i.column_id
WHERE column_store_order_ordinal <>0;
```

B. To change column ordinal, add or remove columns from the order list, or to change from CCI to ordered CCI:

```
CREATE CLUSTERED COLUMNSTORE INDEX InternetSales ON dbo.InternetSales
ORDER (ProductKey, SalesAmount)
WITH (DROP_EXISTING = ON);
```

Next steps

For more development tips, see development overview.

Performance tuning with result set caching

8/19/2022 • 3 minutes to read • Edit Online

When result set caching is enabled, dedicated SQL pool automatically caches query results in the user database for repetitive use. This allows subsequent query executions to get results directly from the persisted cache so recomputation is not needed. Result set caching improves query performance and reduces compute resource usage. In addition, queries using cached results set do not use any concurrency slots and thus do not count against existing concurrency limits. For security, users can only access the cached results if they have the same data access permissions as the users creating the cached results. Result set caching is OFF by default at the database and session levels.

> **NOTE**
> Result set caching should not be used in conjunction with DECRYPTBYKEY. If this cryptographic function must be used, ensure you have result set caching disabled (either at session-level or database-level) at the time of execution.

Key commands

Turn ON/OFF result set caching for a user database

Turn ON/OFF result set caching for a session

Check the size of cached result set

Clean up the cache

What's not cached

Once result set caching is turned ON for a database, results are cached for all queries until the cache is full, except for these queries:

- Queries with built-in functions or runtime expressions that are non-deterministic even when there's no change in base tables' data or query. For example, DateTime.Now(), GetDate().
- Queries using user defined functions
- Queries using tables with row level security or column level security enabled
- Queries returning data with row size larger than 64KB
- Queries returning large data in size (>10GB)

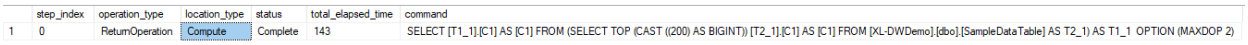
> **NOTE**
> - Some non-deterministic functions and runtime expressions can be deterministic to repetitive queries against the same data. For example, ROW_NUMBER().
> - Use ORDER BY in your query if the order/sequence of rows in the query result set is important to your application logic.
> - If data in the ORDER BY columns are not unique, there's no guaranteed row order for rows with the same values in the ORDER BY columns, regardless if result set caching is enabled or disabled.

> **IMPORTANT**
> The operations to create result set cache and retrieve data from the cache happen on the control node of a dedicated SQL pool instance. When result set caching is turned ON, running queries that return large result set (for example, >1GB) can cause high throttling on the control node and slow down the overall query response on the instance. Those queries are commonly used during data exploration or ETL operations. To avoid stressing the control node and cause performance issue, users should turn OFF result set caching on the database before running those types of queries.

Run this query for the time taken by result set caching operations for a query:

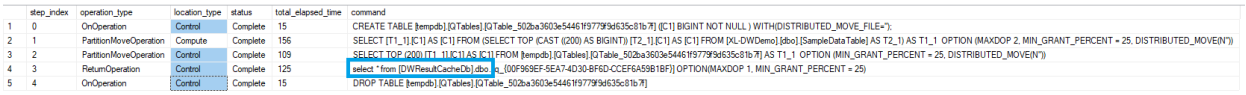
```
SELECT step_index, operation_type, location_type, status, total_elapsed_time, command
FROM sys.dm_pdw_request_steps
WHERE request_id  = <'request_id'>;
```

Here is an example output for a query executed with result set caching disabled.



	step_index	operation_type	location_type	status	total_elapsed_time	command
1	0	ReturnOperation	Compute	Complete	143	SELECT [T1_1].[C1] AS [C1] FROM (SELECT TOP (CAST ((200) AS BIGINT)) [T2_1].[C1] AS [C1] FROM [XL-DWDemo].[dbo].[SampleDataTable] AS T2_1) AS T1_1 OPTION (MAXDOP 2)

Here is an example output for a query executed with result set caching enabled.



	step_index	operation_type	location_type	status	total_elapsed_time	command
1	0	OnOperation	Control	Complete	15	CREATE TABLE [tempdb].[QTables].[QTable_502ba3603e54461f9779f9d635c81b7f] ([C1] BIGINT NOT NULL) WITH(DISTRIBUTED_MOVE_FILE='');
2	1	PartitionMoveOperation	Compute	Complete	156	SELECT [T1_1].[C1] AS [C1] FROM (SELECT TOP (CAST ((200) AS BIGINT)) [T2_1].[C1] AS [C1] FROM [XL-DWDemo].[dbo].[SampleDataTable] AS T2_1) AS T1_1 OPTION (MAXDOP 2, MIN_GRANT_PERCENT = 25, DISTRIBUTED_MOVE(N''))
3	2	PartitionMoveOperation	Control	Complete	109	SELECT TOP (200) [T1_1].[C1] AS [C1] FROM [tempdb].[QTables].[QTable_502ba3603e54461f9779f9d635c81b7f] AS T1_1 OPTION (MIN_GRANT_PERCENT = 25, DISTRIBUTED_MOVE(N''))
4	3	ReturnOperation	Control	Complete	125	select * from [DWResultCacheDb].[dbo].[x_{00F969EF-5EA7-4D30-BF6D-CCEF6A59B1BF}] OPTION(MAXDOP 1, MIN_GRANT_PERCENT = 25)
5	4	OnOperation	Control	Complete	15	DROP TABLE [tempdb].[QTables].[QTable_502ba3603e54461f9779f9d635c81b7f]

When cached results are used

Cached result set is reused for a query if all of the following requirements are all met:

- The user who's running the query has access to all the tables referenced in the query.
- There is an exact match between the new query and the previous query that generated the result set cache.
- There is no data or schema changes in the tables where the cached result set was generated from.

Run this command to check if a query was executed with a result cache hit or miss. The result_cache_hit column returns 1 for cache hit, 0 for cache miss, and negative values for reasons why result set caching was not used. Check sys.dm_pdw_exec_requests for details.

```
SELECT request_id, command, result_cache_hit FROM sys.dm_pdw_exec_requests
WHERE request_id = <'Your_Query_Request_ID'>
```

Manage cached results

The maximum size of result set cache is 1 TB per database. The cached results are automatically invalidated when the underlying query data change.

The cache eviction is managed by dedicated SQL pool automatically following this schedule:

- Every 48 hours if the result set hasn't been used or has been invalidated.
- When the result set cache approaches the maximum size.

Users can manually empty the entire result set cache by using one of these options:

- Turn OFF the result set cache feature for the database
- Run DBCC DROPRESULTSETCACHE while connected to the database

Pausing a database won't empty cached result set.

Next steps

For more development tips, see development overview.

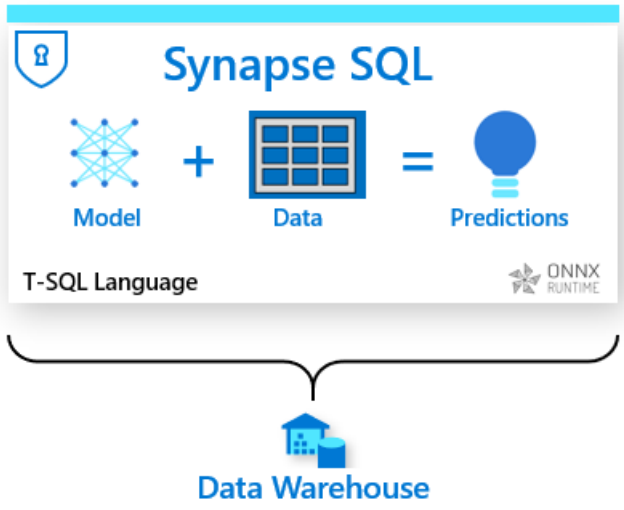
Score machine learning models with PREDICT

8/19/2022 • 3 minutes to read • Edit Online

Dedicated SQL pool provides you the capability to score machine learning models using the familiar T-SQL language. With T-SQL PREDICT, you can bring your existing machine learning models trained with historical data and score them within the secure boundaries of your data warehouse. PREDICT function takes an ONNX (Open Neural Network Exchange) model and data as inputs. This feature eliminates the step of moving valuable data outside the data warehouse for scoring. It aims to empower data professionals to easily deploy machine learning models with the familiar T-SQL interface as well as collaborate seamlessly with data scientists working with the right framework for their task.

> **NOTE**
> This functionality is currently not supported in serverless SQL pool.

The functionality requires that the model is trained outside of Synapse SQL. After building the model, load it into the data warehouse and score it with the T-SQL Predict syntax to get insights from the data.



Training the Model

Dedicated SQL pool expects a pre-trained model. Keep the following factors in mind when training a machine learning model that is used for performing predictions in dedicated SQL pool.

- Dedicated SQL pool only supports ONNX format models. ONNX is an open-source model format that allows you to exchange models between various frameworks to enable interoperability. You can convert your existing models to ONNX format using frameworks that either support it natively or have converting packages available. For example, sklearn-onnx package convert scikit-learn models to ONNX. ONNX GitHub repository provides a list of supported frameworks and examples.

  If you are using Automated ML for training, make sure to set the *enable_onnx_compatible_models* parameter to TRUE to produce an ONNX format model. Automated Machine Learning Notebook shows an example of how to use automated ML to create a machine learning model of ONNX format.

- The following data types are supported for the input data:

    - int, bigint, real, float
    - char, varchar, nvarchar
- The scoring data needs to be in the same format as the training data. Complex data types such as multi-dimensional arrays are not supported by PREDICT. So, for training make sure that each input of the model corresponds to a single column of the scoring table instead of passing a single array containing all inputs.
- Make sure that the names and data types of the model inputs match the column names and data types of the new prediction data. Visualizing an ONNX model using various open-source tools available online can further help with debugging.

Loading the model

The model is stored in a dedicated SQL pool user table as a hexadecimal string. Additional columns such as ID and description may be added in the model table to identify the model. Use varbinary(max) as the data type of the model column. Here is a code example for a table that can be used for storing models:

```
-- Sample table schema for storing a model and related data
CREATE TABLE [dbo].[Models]
(
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [Model] [varbinary](max) NULL,
    [Description] [varchar](200) NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    HEAP
)
GO
```

Once the model is converted to a hexadecimal string and the table definition specified, use the COPY command or Polybase to load the model in the dedicated SQL pool table. The following code sample uses the Copy command to load the model.

```
-- Copy command to load hexadecimal string of the model from Azure Data Lake storage location
COPY INTO [Models] (Model)
FROM '<enter your storage location>'
WITH (
    FILE_TYPE = 'CSV',
    CREDENTIAL=(IDENTITY= 'Shared Access Signature', SECRET='<enter your storage key here>')
)
```

Scoring the model

Once the model and data are loaded in the data warehouse, use the **T-SQL PREDICT** function to score the model. Make sure that the new input data is in the same format as the training data used for building the model. T-SQL PREDICT takes two inputs: model and new scoring input data, and generates new columns for the output.The model can be specified as a variable, a literal or a scalar sub_query. Use WITH common_table_expression to specify a named result set for the data parameter.

The example below shows a sample query using prediction function. An additional column with name *Score* and data type *float* is created containing the prediction results. All the input data columns as well as output prediction columns are available to display with the select statement. For more details, see PREDICT (Transact-SQL).

```
-- Query for ML predictions
SELECT d.*, p.Score
FROM PREDICT(MODEL = (SELECT Model FROM Models WHERE Id = 1),
DATA = dbo.mytable AS d, RUNTIME = ONNX) WITH (Score float) AS p;
```

Next steps

To learn more about the PREDICT function, see PREDICT (Transact-SQL).

Design tables using Synapse SQL in Azure Synapse Analytics

8/19/2022 • 12 minutes to read • Edit Online

This document includes key concepts for designing tables with dedicated SQL pool and serverless SQL pool.

Serverless SQL pool is a query service over the data in your data lake. It doesn't have local storage for data ingestion. Dedicated SQL pool represents a collection of analytic resources that are being provisioned when using Synapse SQL. The size of a dedicated SQL pool is determined by Data Warehousing Units (DWU).

The following table lists the topics that are relevant to dedicated SQL pool vs. serverless SQL pool:

TOPIC	DEDICATED SQL POOL	SERVERLESS SQL POOL
Determine table category	Yes	No
Schema names	Yes	Yes
Table names	Yes	No
Table persistence	Yes	No
Regular table	Yes	No
Temporary table	Yes	Yes
External table	Yes	Yes
Data types	Yes	Yes
Distributed tables	Yes	No
Hash-distributed tables	Yes	No
Replicated tables	Yes	No
Round-robin tables	Yes	No
Common distribution methods for tables	Yes	No
Partitions	Yes	Yes
Columnstore indexes	Yes	No
Statistics	Yes	Yes
Primary key and unique key	Yes	No

TOPIC	DEDICATED SQL POOL	SERVERLESS SQL POOL
Commands for creating tables	Yes	No
Aligning source data with the data warehouse	Yes	No
Unsupported table features	Yes	No
Table size queries	Yes	No

Determine table category

A star schema organizes data into fact and dimension tables. Some tables are used for integration or staging data before moving to a fact or dimension table. As you design a table, decide whether the table data belongs in a fact, dimension, or integration table. This decision informs the appropriate table structure and distribution.

- **Fact tables** contain quantitative data that are commonly generated in a transactional system, and then loaded into the data warehouse. For example, a retail business generates sales transactions every day, and then loads the data into a data warehouse fact table for analysis.
- **Dimension tables** contain attribute data that might change but usually changes infrequently. For example, a customer's name and address are stored in a dimension table and updated only when the customer's profile changes. To minimize the size of a large fact table, the customer's name and address don't need to be in every row of a fact table. Instead, the fact table and the dimension table can share a customer ID. A query can join the two tables to associate a customer's profile and transactions.
- **Integration tables** provide a place for integrating or staging data. You can create an integration table as a regular table, an external table, or a temporary table. For example, you can load data to a staging table, perform transformations on the data in staging, and then insert the data into a production table.

Schema names

Schemas are a good way to group together objects that are used in a similar fashion. The following code creates a user-defined schema called wwi.

```
CREATE SCHEMA wwi;
```

Table names

If you're migrating multiple databases from an on-prem solution to dedicated SQL pool, the best practice is to migrate all of the fact, dimension, and integration tables to one SQL pool schema. For example, you could store all the tables in the WideWorldImportersDW sample data warehouse within one schema called wwi.

To show the organization of the tables in dedicated SQL pool, you could use fact, dim, and int as prefixes to the table names. The table below shows some of the schema and table names for WideWorldImportersDW.

WIDEWORLDIMPORTERSDW TABLE	TABLE TYPE	DEDICATED SQL POOL
City	Dimension	wwi.DimCity
Order	Fact	wwi.FactOrder

Table persistence

Tables store data either permanently in Azure Storage, temporarily in Azure Storage, or in a data store external to the data warehouse.

Regular table

A regular table stores data in Azure Storage as part of the data warehouse. The table and the data persist whether or not a session is open. The example below creates a regular table with two columns.

```
CREATE TABLE MyTable (col1 int, col2 int );
```

Temporary table

A temporary table only exists for the duration of the session. You can use a temporary table to prevent other users from seeing temporary results. Using temporary tables also reduces the need for cleanup. Temporary tables utilize local storage and, in dedicated SQL pools, can offer faster performance.

Serverless SQL pool supports temporary tables. But, its usage is limited since you can select from a temporary table but cannot join it with files in storage.

For more information, see Temporary tables.

External table

External tables point to data located in Azure Storage blob or Azure Data Lake Storage.

Import data from external tables into dedicated SQL pools using the CREATE TABLE AS SELECT statement. For a loading tutorial, see Use PolyBase to load data from Azure blob storage.

For serverless SQL pool, you can use CETAS to save the query result to an external table in Azure Storage.

Data types

Dedicated SQL pool supports the most commonly used data types. For a list of the supported data types, see data types in CREATE TABLE reference in the CREATE TABLE statement. For more information on using data types, see Data types.

Distributed tables

A fundamental feature of dedicated SQL pool is the way it can store and operate on tables across distributions. Dedicated SQL pool supports three methods for distributing data:

- Round-robin (default)
- Hash
- Replicated

Hash-distributed tables

A hash distributed table distributes rows based on the value in the distribution column. A hash distributed table is designed to achieve high performance for queries on large tables. There are several factors to consider when choosing a distribution column.

For more information, see Design guidance for distributed tables.

Replicated tables

A replicated table has a full copy of the table available on every Compute node. Queries run fast on replicated tables because joins on replicated tables don't require data movement. Replication requires extra storage, though, and isn't practical for large tables.

For more information, see Design guidance for replicated tables.

Round-robin tables

A round-robin table distributes table rows evenly across all distributions. The rows are distributed randomly. Loading data into a round-robin table is fast. But, queries can require more data movement than the other distribution methods.

For more information, see Design guidance for distributed tables.

Common distribution methods for tables

The table category often determines the optimal option for table distribution.

TABLE CATEGORY	RECOMMENDED DISTRIBUTION OPTION
Fact	Use hash-distribution with clustered columnstore index. Performance improves when two hash tables are joined on the same distribution column.
Dimension	Use replicated for smaller tables. If tables are too large to store on each Compute node, use hash-distributed.
Staging	Use round-robin for the staging table. The load with CTAS is fast. Once the data is in the staging table, use INSERT...SELECT to move the data to production tables.

Partitions

In dedicated SQL pools, a partitioned table stores and executes operations on the table rows according to data ranges. For example, a table could be partitioned by day, month, or year. You can improve query performance through partition elimination, which limits a query scan to data within a partition.

You can also maintain the data through partition switching. Since the data in a dedicated SQL pool is already distributed, too many partitions can slow query performance. For more information, see Partitioning guidance.

> **TIP**
> When partition switching into table partitions that are not empty, consider using the TRUNCATE_TARGET option in your ALTER TABLE statement if the existing data is to be truncated.

The code below switches the transformed daily data into a SalesFact partition and overwrites any existing data.

```
ALTER TABLE SalesFact_DailyFinalLoad SWITCH PARTITION 256 TO SalesFact PARTITION 256 WITH (TRUNCATE_TARGET =
ON);
```

In serverless SQL pool, you can limit the files/folders (partitions) that will be read by your query. Partitioning by path is supported using the filepath and fileinfo functions described in Querying storage files. The following example reads a folder with data for year 2017:

```
SELECT
    nyc.filepath(1) AS [year],
    payment_type,
    SUM(fare_amount) AS fare_total
FROM
    OPENROWSET(
        BULK 'https://sqlondemandstorage.blob.core.windows.net/parquet/taxi/year=*/month=*/*.parquet',
        FORMAT='PARQUET'
    ) AS nyc
WHERE
    nyc.filepath(1) = 2017
GROUP BY
    nyc.filepath(1),
    payment_type
ORDER BY
    nyc.filepath(1),
    payment_type
```

Columnstore indexes

By default, dedicated SQL pool stores a table as a clustered columnstore index. This form of data storage achieves high data compression and query performance on large tables. The clustered columnstore index is usually the best choice, but in some cases a clustered index or a heap is the appropriate storage structure.

> **TIP**
> A heap table can be especially useful for loading transient data, such as a staging table, which is transformed into a final table.

For a list of columnstore features, see What's new for columnstore indexes. To improve columnstore index performance, see Maximizing rowgroup quality for columnstore indexes.

Statistics

The query optimizer uses column-level statistics when it creates the plan for executing a query. To improve query performance, it's important to have statistics on individual columns, especially columns used in query joins. Synapse SQL supports automatic creation of statistics.

Statistical updating doesn't happen automatically. Update statistics after a significant number of rows are added or changed. For instance, update statistics after a load. Additional information is provided in the Statistics guidance article.

Primary key and unique key

For dedicated SQL pool, PRIMARY KEY is only supported when NONCLUSTERED and NOT ENFORCED are both used. UNIQUE constraint is only supported when NOT ENFORCED is used. For more information, see the dedicated SQL pool table constraints article.

Commands for creating tables

For dedicated SQL pool, you can create a table as a new empty table. You can also create and populate a table with the results of a select statement. The following are the T-SQL commands for creating a table.

T-SQL STATEMENT	DESCRIPTION
CREATE TABLE	Creates an empty table by defining all the table columns and options.
CREATE EXTERNAL TABLE	Creates an external table. The definition of the table is stored in dedicated SQL pool. The table data is stored in Azure Blob storage or Azure Data Lake Storage.
CREATE TABLE AS SELECT	Populates a new table with the results of a select statement. The table columns and data types are based on the select statement results. To import data, this statement can select from an external table.
CREATE EXTERNAL TABLE AS SELECT	Creates a new external table by exporting the results of a select statement to an external location. The location is either Azure Blob storage or Azure Data Lake Storage.

Align source data with the data warehouse

Dedicated SQL pool tables are populated by loading data from another data source. To achieve a successful load, the number and data types of the columns in the source data must align with the table definition in the data warehouse.

> **NOTE**
> Getting the data to align might be the hardest part of designing your tables.

If data is coming from multiple data stores, you can port the data into the data warehouse and store it in an integration table. Once data is in the integration table, you can use the power of dedicated SQL pool to implement transformation operations. Once the data is prepared, you can insert it into production tables.

Unsupported table features

Dedicated SQL pool supports many, but not all, of the table features offered by other databases. The following list shows some of the table features that aren't supported in dedicated SQL pool.

- Foreign key, check Table Constraints
- Computed Columns
- Indexed Views
- Sequence
- Sparse Columns
- Surrogate Keys, implement with Identity
- Synonyms
- Triggers
- Unique Indexes
- User-Defined Types

Table size queries

In dedicated SQL pool, one simple way to identify space and rows consumed by a table in each of the 60 distributions is to use DBCC PDW_SHOWSPACEUSED.

```
DBCC PDW_SHOWSPACEUSED('dbo.FactInternetSales');
```

Keep in mind that using DBCC commands can be quite limiting. Dynamic management views (DMVs) show more detail than DBCC commands. Start by creating the view below.

```
CREATE VIEW dbo.vTableSizes
AS
WITH base
AS
(
SELECT
 GETDATE()                                                              AS  [execution_time]
, DB_NAME()                                                             AS  [database_name]
, s.name                                                                AS  [schema_name]
, t.name                                                                AS  [table_name]
, QUOTENAME(s.name)+'.'+QUOTENAME(t.name)                               AS  [two_part_name]
, nt.[name]                                                             AS  [node_table_name]
, ROW_NUMBER() OVER(PARTITION BY nt.[name] ORDER BY (SELECT NULL))      AS  [node_table_name_seq]
, tp.[distribution_policy_desc]                                         AS  [distribution_policy_name]
, c.[name]                                                              AS  [distribution_column]
, nt.[distribution_id]                                                  AS  [distribution_id]
, i.[type]                                                              AS  [index_type]
, i.[type_desc]                                                         AS  [index_type_desc]
, nt.[pdw_node_id]                                                      AS  [pdw_node_id]
, pn.[type]                                                             AS  [pdw_node_type]
, pn.[name]                                                             AS  [pdw_node_name]
, di.name                                                               AS  [dist_name]
, di.position                                                           AS  [dist_position]
, nps.[partition_number]                                                AS  [partition_nmbr]
, nps.[reserved_page_count]                                             AS  [reserved_space_page_count]
, nps.[reserved_page_count] - nps.[used_page_count]                     AS  [unused_space_page_count]
, nps.[in_row_data_page_count]
    + nps.[row_overflow_used_page_count]
    + nps.[lob_used_page_count]                                         AS  [data_space_page_count]
, nps.[reserved_page_count]
 - (nps.[reserved_page_count] - nps.[used_page_count])
 - ([in_row_data_page_count]
        + [row_overflow_used_page_count]+[lob_used_page_count])         AS  [index_space_page_count]
, nps.[row_count]                                                       AS  [row_count]
from
    sys.schemas s
INNER JOIN sys.tables t
    ON s.[schema_id] = t.[schema_id]
INNER JOIN sys.indexes i
    ON  t.[object_id] = i.[object_id]
    AND i.[index_id] <= 1
INNER JOIN sys.pdw_table_distribution_properties tp
    ON t.[object_id] = tp.[object_id]
INNER JOIN sys.pdw_table_mappings tm
    ON t.[object_id] = tm.[object_id]
INNER JOIN sys.pdw_nodes_tables nt
    ON tm.[physical_name] = nt.[name]
INNER JOIN sys.dm_pdw_nodes pn
    ON  nt.[pdw_node_id] = pn.[pdw_node_id]
INNER JOIN sys.pdw_distributions di
    ON  nt.[distribution_id] = di.[distribution_id]
INNER JOIN sys.dm_pdw_nodes_db_partition_stats nps
    ON nt.[object_id] = nps.[object_id]
    AND nt.[pdw_node_id] = nps.[pdw_node_id]
    AND nt.[distribution_id] = nps.[distribution_id]
LEFT OUTER JOIN (select * from sys.pdw_column_distribution_properties where distribution_ordinal = 1) cdp
    ON t.[object_id] = cdp.[object_id]
LEFT OUTER JOIN sys.columns c
    ON cdp.[object_id] = c.[object_id]
    AND cdp.[column_id] = c.[column_id]
WHERE pn.[type] = 'COMPUTE'
```

```
WHERE pnp.[type]    = 'CMPUTE'
)
, size
AS
(
SELECT
   [execution_time]
,  [database_name]
,  [schema_name]
,  [table_name]
,  [two_part_name]
,  [node_table_name]
,  [node_table_name_seq]
,  [distribution_policy_name]
,  [distribution_column]
,  [distribution_id]
,  [index_type]
,  [index_type_desc]
,  [pdw_node_id]
,  [pdw_node_type]
,  [pdw_node_name]
,  [dist_name]
,  [dist_position]
,  [partition_nmbr]
,  [reserved_space_page_count]
,  [unused_space_page_count]
,  [data_space_page_count]
,  [index_space_page_count]
,  [row_count]
,  ([reserved_space_page_count] * 8.0)                             AS [reserved_space_KB]
,  ([reserved_space_page_count] * 8.0)/1000                        AS [reserved_space_MB]
,  ([reserved_space_page_count] * 8.0)/1000000                     AS [reserved_space_GB]
,  ([reserved_space_page_count] * 8.0)/1000000000                  AS [reserved_space_TB]
,  ([unused_space_page_count]   * 8.0)                             AS [unused_space_KB]
,  ([unused_space_page_count]   * 8.0)/1000                        AS [unused_space_MB]
,  ([unused_space_page_count]   * 8.0)/1000000                     AS [unused_space_GB]
,  ([unused_space_page_count]   * 8.0)/1000000000                  AS [unused_space_TB]
,  ([data_space_page_count]     * 8.0)                             AS [data_space_KB]
,  ([data_space_page_count]     * 8.0)/1000                        AS [data_space_MB]
,  ([data_space_page_count]     * 8.0)/1000000                     AS [data_space_GB]
,  ([data_space_page_count]     * 8.0)/1000000000                  AS [data_space_TB]
,  ([index_space_page_count]  * 8.0)                               AS [index_space_KB]
,  ([index_space_page_count]  * 8.0)/1000                          AS [index_space_MB]
,  ([index_space_page_count]  * 8.0)/1000000                       AS [index_space_GB]
,  ([index_space_page_count]  * 8.0)/1000000000                    AS [index_space_TB]
FROM base
)
SELECT *
FROM size
;
```

Table space summary

This query returns the rows and space by table. Table space summary allows you to see which tables are your largest tables. You'll also see whether they're round-robin, replicated, or hash-distributed. For hash-distributed tables, the query shows the distribution column.

```
SELECT
     database_name
,    schema_name
,    table_name
,    distribution_policy_name
,      distribution_column
,    index_type_desc
,    COUNT(distinct partition_nmbr) as nbr_partitions
,    SUM(row_count)                 as table_row_count
,    SUM(reserved_space_GB)         as table_reserved_space_GB
,    SUM(data_space_GB)             as table_data_space_GB
,    SUM(index_space_GB)            as table_index_space_GB
,    SUM(unused_space_GB)           as table_unused_space_GB
FROM
    dbo.vTableSizes
GROUP BY
     database_name
,    schema_name
,    table_name
,    distribution_policy_name
,      distribution_column
,    index_type_desc
ORDER BY
    table_reserved_space_GB desc
;
```

Table space by distribution type

```
SELECT
     distribution_policy_name
,    SUM(row_count)                 as table_type_row_count
,    SUM(reserved_space_GB)         as table_type_reserved_space_GB
,    SUM(data_space_GB)             as table_type_data_space_GB
,    SUM(index_space_GB)            as table_type_index_space_GB
,    SUM(unused_space_GB)           as table_type_unused_space_GB
FROM dbo.vTableSizes
GROUP BY distribution_policy_name
;
```

Table space by index type

```
SELECT
     index_type_desc
,    SUM(row_count)                 as table_type_row_count
,    SUM(reserved_space_GB)         as table_type_reserved_space_GB
,    SUM(data_space_GB)             as table_type_data_space_GB
,    SUM(index_space_GB)            as table_type_index_space_GB
,    SUM(unused_space_GB)           as table_type_unused_space_GB
FROM dbo.vTableSizes
GROUP BY index_type_desc
;
```

Distribution space summary

```
SELECT
    distribution_id
,    SUM(row_count)                as total_node_distribution_row_count
,    SUM(reserved_space_MB)        as total_node_distribution_reserved_space_MB
,    SUM(data_space_MB)            as total_node_distribution_data_space_MB
,    SUM(index_space_MB)           as total_node_distribution_index_space_MB
,    SUM(unused_space_MB)          as total_node_distribution_unused_space_MB
FROM dbo.vTableSizes
GROUP BY     distribution_id
ORDER BY    distribution_id
;
```

Next steps

After creating the tables for your data warehouse, the next step is to load data into the table. For a loading tutorial, see Loading data into dedicated SQL pool.

CETAS with Synapse SQL

8/19/2022 • 3 minutes to read • Edit Online

You can use CREATE EXTERNAL TABLE AS SELECT (CETAS) in dedicated SQL pool or serverless SQL pool to complete the following tasks:

- Create an external table
- Export, in parallel, the results of a Transact-SQL SELECT statement to:
  - Hadoop
  - Azure Storage Blob
  - Azure Data Lake Storage Gen2

CETAS in dedicated SQL pool

For dedicated SQL pool, CETAS usage and syntax, check the CREATE EXTERNAL TABLE AS SELECT article. Additionally, for guidance on CTAS using dedicated SQL pool, see the CREATE TABLE AS SELECT article.

CETAS in serverless SQL pool

When using serverless SQL pool, CETAS is used to create an external table and export query results to Azure Storage Blob or Azure Data Lake Storage Gen2.

Syntax

```
CREATE EXTERNAL TABLE [ [database_name  . [ schema_name ] . ] | schema_name . ] table_name
    WITH (
        LOCATION = 'path_to_folder',
        DATA_SOURCE = external_data_source_name,
        FILE_FORMAT = external_file_format_name
)
    AS <select_statement>
[;]

<select_statement> ::=
    [ WITH <common_table_expression> [ ,...n ] ]
    SELECT <select_criteria>
```

Arguments

[[database_name . [schema_name] .] | schema_name .] table_name

The one to three-part name of the table to create. For an external table, serverless SQL pool stores only the table metadata. No actual data is moved or stored in serverless SQL pool.

LOCATION = *'path_to_folder'*

Specifies where to write the results of the SELECT statement on the external data source. The root folder is the data location specified in the external data source. LOCATION must point to a folder and have a trailing /. Example: aggregated_data/

DATA_SOURCE = *external_data_source_name*

Specifies the name of the external data source object that contains the location where the external data will be stored. To create an external data source, use CREATE EXTERNAL DATA SOURCE (Transact-SQL).

FILE_FORMAT = *external_file_format_name*

Specifies the name of the external file format object that contains the format for the external data file. To create an external file format, use CREATE EXTERNAL FILE FORMAT (Transact-SQL). Only external file formats with FORMAT_TYPE=PARQUET and FORMAT_TYPE=DELIMITEDTEXT are currently supported. GZip compression for DELIMITEDTEXT format is not supported.

WITH *<common_table_expression>*

Specifies a temporary named result set, known as a common table expression (CTE). For more information, see WITH common_table_expression (Transact-SQL).

SELECT <select_criteria>

Populates the new table with the results from a SELECT statement. *select_criteria* is the body of the SELECT statement that determines which data to copy to the new table. For information about SELECT statements, see SELECT (Transact-SQL).

> **NOTE**
> ORDER BY clause in SELECT is not supported for CETAS.

Permissions

You need to have permissions to list folder content and write to the LOCATION folder for CETAS to work.

Examples

These examples use CETAS to save total population aggregated by year and state to an aggregated_data folder that is located in the population_ds datasource.

This sample relies on the credential, data source, and external file format created previously. Refer to the external tables document. To save query results to a different folder in the same data source, change the LOCATION argument.

To save results to a different storage account, create and use a different data source for DATA_SOURCE argument.

> **NOTE**
> The samples that follow use a public Azure Open Data storage account. It is read-only. To execute these queries, you need to provide the data source for which you have write permissions.

```
-- use CETAS to export select statement with OPENROWSET result to  storage
CREATE EXTERNAL TABLE population_by_year_state
WITH (
    LOCATION = 'aggregated_data/',
    DATA_SOURCE = population_ds,
    FILE_FORMAT = census_file_format
)
AS
SELECT decennialTime, stateName, SUM(population) AS population
FROM
    OPENROWSET(BULK
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us_population_county/year=*/
*.parquet',
    FORMAT='PARQUET') AS [r]
GROUP BY decennialTime, stateName
GO

-- you can query the newly created external table
SELECT * FROM population_by_year_state
```

The following sample uses an external table as the source for CETAS. It relies on the credential, data source, external file format, and external table created previously. Refer to the external tables document.

```
-- use CETAS with select from external table
CREATE EXTERNAL TABLE population_by_year_state
WITH (
    LOCATION = 'aggregated_data/',
    DATA_SOURCE = population_ds,
    FILE_FORMAT = census_file_format
)
AS
SELECT decennialTime, stateName, SUM(population) AS population
FROM census_external_table
GROUP BY decennialTime, stateName
GO

-- you can query the newly created external table
SELECT * FROM population_by_year_state
```

Supported data types

CETAS can be used to store result sets with following SQL data types:

- binary
- varbinary
- char
- varchar
- nchar
- nvarchar
- smalldate
- date
- datetime
- datetime2
- datetimeoffset
- time
- decimal
- numeric

- float
- real
- bigint
- tinyint
- smallint
- int
- bigint
- bit
- money
- smallmoney
- uniqueidentifier

> **NOTE**
> LOBs larger than 1MB can't be used with CETAS.

Next steps

Try querying Apache Spark for Azure Synapse external tables.

CREATE TABLE AS SELECT (CTAS)

8/19/2022 • 9 minutes to read • Edit Online

This article explains the CREATE TABLE AS SELECT (CTAS) T-SQL statement in Synapse SQL for developing solutions. The article also provides code examples.

CREATE TABLE AS SELECT

The CREATE TABLE AS SELECT (CTAS) statement is one of the most important T-SQL features available. CTAS is a parallel operation that creates a new table based on the output of a SELECT statement. CTAS is the simplest and fastest way to create and insert data into a table with a single command.

SELECT...INTO vs. CTAS

CTAS is a more customizable version of the SELECT...INTO statement.

The following is an example of a simple SELECT...INTO:

```
SELECT *
INTO    [dbo].[FactInternetSales_new]
FROM    [dbo].[FactInternetSales]
```

SELECT...INTO doesn't allow you to change either the distribution method or the index type as part of the operation. You create `[dbo].[FactInternetSales_new]` by using the default distribution type of ROUND_ROBIN, and the default table structure of CLUSTERED COLUMNSTORE INDEX.

With CTAS, on the other hand, you can specify both the distribution of the table data as well as the table structure type. To convert the previous example to CTAS:

```
CREATE TABLE [dbo].[FactInternetSales_new]
WITH
(
 DISTRIBUTION = ROUND_ROBIN
 ,CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT  *
FROM    [dbo].[FactInternetSales];
```

> **NOTE**
>
> If you're only trying to change the index in your CTAS operation, and the source table is hash distributed, maintain the same distribution column and data type. This avoids cross-distribution data movement during the operation, which is more efficient.

Use CTAS to copy a table

Perhaps one of the most common uses of CTAS is creating a copy of a table in order to change the DDL. Let's say you originally created your table as `ROUND_ROBIN`, and now want to change it to a table distributed on a column. CTAS is how you would change the distribution column. You can also use CTAS to change partitioning, indexing, or column types.

Let's say you created this table by using the default distribution type of `ROUND_ROBIN`, not specifying a distribution column in the `CREATE TABLE`.

```
CREATE TABLE FactInternetSales
(
    ProductKey int NOT NULL,
    OrderDateKey int NOT NULL,
    DueDateKey int NOT NULL,
    ShipDateKey int NOT NULL,
    CustomerKey int NOT NULL,
    PromotionKey int NOT NULL,
    CurrencyKey int NOT NULL,
    SalesTerritoryKey int NOT NULL,
    SalesOrderNumber nvarchar(20) NOT NULL,
    SalesOrderLineNumber tinyint NOT NULL,
    RevisionNumber tinyint NOT NULL,
    OrderQuantity smallint NOT NULL,
    UnitPrice money NOT NULL,
    ExtendedAmount money NOT NULL,
    UnitPriceDiscountPct float NOT NULL,
    DiscountAmount float NOT NULL,
    ProductStandardCost money NOT NULL,
    TotalProductCost money NOT NULL,
    SalesAmount money NOT NULL,
    TaxAmt money NOT NULL,
    Freight money NOT NULL,
    CarrierTrackingNumber nvarchar(25),
    CustomerPONumber nvarchar(25));
```

Now you want to create a new copy of this table, with a `Clustered Columnstore Index`, so you can take advantage of the performance of Clustered Columnstore tables. You also want to distribute this table on `ProductKey`, because you're anticipating joins on this column and want to avoid data movement during joins on `ProductKey`. Lastly, you also want to add partitioning on `OrderDateKey`, so you can quickly delete old data by dropping old partitions. Here is the CTAS statement, which copies your old table into a new table.

```
CREATE TABLE FactInternetSales_new
WITH
(
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = HASH(ProductKey),
    PARTITION
    (
        OrderDateKey RANGE RIGHT FOR VALUES
        (
        20000101,20010101,20020101,20030101,20040101,20050101,20060101,20070101,20080101,20090101,
        20100101,20110101,20120101,20130101,20140101,20150101,20160101,20170101,20180101,20190101,
        20200101,20210101,20220101,20230101,20240101,20250101,20260101,20270101,20280101,20290101
        )
    )
)
AS SELECT * FROM FactInternetSales;
```

Finally, you can rename your tables, to swap in your new table and then drop your old table.

```
RENAME OBJECT FactInternetSales TO FactInternetSales_old;
RENAME OBJECT FactInternetSales_new TO FactInternetSales;

DROP TABLE FactInternetSales_old;
```

Use CTAS to work around unsupported features

You can also use CTAS to work around a number of the unsupported features listed below. This method can often prove helpful, because not only will your code be compliant, but it will often run faster on Synapse SQL. This performance is a result of its fully parallelized design. Scenarios include:

- ANSI JOINS on UPDATEs
- ANSI JOINs on DELETEs
- MERGE statement

> **TIP**
> Try to think "CTAS first." Solving a problem by using CTAS is generally a good approach, even if you're writing more data as a result.

ANSI join replacement for update statements

You might find that you have a complex update. The update joins more than two tables together by using ANSI join syntax to perform the UPDATE or DELETE.

Imagine you had to update this table:

```
CREATE TABLE [dbo].[AnnualCategorySales]
( [EnglishProductCategoryName]    NVARCHAR(50)    NOT NULL
, [CalendarYear]                    SMALLINT        NOT NULL
, [TotalSalesAmount]                MONEY            NOT NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN
);
```

The original query might have looked something like this example:

```
UPDATE    acs
SET        [TotalSalesAmount] = [fis].[TotalSalesAmount]
FROM    [dbo].[AnnualCategorySales]     AS acs
JOIN    (
        SELECT [EnglishProductCategoryName]
        , [CalendarYear]
        , SUM([SalesAmount])                AS [TotalSalesAmount]
        FROM    [dbo].[FactInternetSales]        AS s
        JOIN    [dbo].[DimDate]                    AS d    ON s.[OrderDateKey]                = d.[DateKey]
        JOIN    [dbo].[DimProduct]                AS p    ON s.[ProductKey]                = p.[ProductKey]
        JOIN    [dbo].[DimProductSubCategory]    AS u    ON p.[ProductSubcategoryKey]    = u.
[ProductSubcategoryKey]
        JOIN    [dbo].[DimProductCategory]        AS c    ON u.[ProductCategoryKey]        = c.
[ProductCategoryKey]
        WHERE     [CalendarYear] = 2004
        GROUP BY
                [EnglishProductCategoryName]
        ,        [CalendarYear]
        ) AS fis
ON    [acs].[EnglishProductCategoryName]    = [fis].[EnglishProductCategoryName]
AND    [acs].[CalendarYear]                = [fis].[CalendarYear];
```

Synapse SQL doesn't support ANSI joins in the `FROM` clause of an `UPDATE` statement, so you can't use the previous example without modifying it.

You can use a combination of a CTAS and an implicit join to replace the previous example:

```
-- Create an interim table
CREATE TABLE CTAS_acs
WITH (DISTRIBUTION = ROUND_ROBIN)
AS
SELECT    ISNULL(CAST([EnglishProductCategoryName] AS NVARCHAR(50)),0) AS [EnglishProductCategoryName]
, ISNULL(CAST([CalendarYear] AS SMALLINT),0)  AS [CalendarYear]
, ISNULL(CAST(SUM([SalesAmount]) AS MONEY),0)  AS [TotalSalesAmount]
FROM    [dbo].[FactInternetSales]        AS s
JOIN    [dbo].[DimDate]                    AS d    ON s.[OrderDateKey]                = d.[DateKey]
JOIN    [dbo].[DimProduct]                 AS p    ON s.[ProductKey]                = p.[ProductKey]
JOIN    [dbo].[DimProductSubCategory]    AS u    ON p.[ProductSubcategoryKey]    = u.[ProductSubcategoryKey]
JOIN    [dbo].[DimProductCategory]         AS c    ON u.[ProductCategoryKey]        = c.[ProductCategoryKey]
WHERE     [CalendarYear] = 2004
GROUP BY [EnglishProductCategoryName]
, [CalendarYear];

-- Use an implicit join to perform the update
UPDATE  AnnualCategorySales
SET     AnnualCategorySales.TotalSalesAmount = CTAS_ACS.TotalSalesAmount
FROM    CTAS_acs
WHERE   CTAS_acs.[EnglishProductCategoryName] = AnnualCategorySales.[EnglishProductCategoryName]
AND     CTAS_acs.[CalendarYear]  = AnnualCategorySales.[CalendarYear] ;

--Drop the interim table
DROP TABLE CTAS_acs;
```

ANSI join replacement for MERGE

In Azure Synapse Analytics, MERGE (preview) with NOT MATCHED BY TARGET requires the target to be a HASH distributed table. Users can use the ANSI JOIN with UPDATE or DELETE as a workaround to modify target table data based on the result from joining with another table. Here is an example.

```
CREATE TABLE dbo.Table1
    (ColA INT NOT NULL, ColB DECIMAL(10,3) NOT NULL);
GO
CREATE TABLE dbo.Table2
    (ColA INT NOT NULL, ColB DECIMAL(10,3) NOT NULL);
GO
INSERT INTO dbo.Table1 VALUES(1, 10.0);
INSERT INTO dbo.Table2 VALUES(1, 0.0);
GO
UPDATE dbo.Table2
SET dbo.Table2.ColB = dbo.Table2.ColB + dbo.Table1.ColB
FROM dbo.Table2
    INNER JOIN dbo.Table1
    ON (dbo.Table2.ColA = dbo.Table1.ColA);
GO
SELECT ColA, ColB
FROM dbo.Table2;
```

Explicitly state data type and nullability of output

When migrating code, you might find you run across this type of coding pattern:

```
DECLARE @d decimal(7,2) = 85.455
,       @f float(24)    = 85.455

CREATE TABLE result
(result DECIMAL(7,2) NOT NULL
)
WITH (DISTRIBUTION = ROUND_ROBIN)

INSERT INTO result
SELECT @d*@f;
```

You might think you should migrate this code to CTAS, and you'd be correct. However, there's a hidden issue here.

The following code doesn't yield the same result:

```
DECLARE @d decimal(7,2) = 85.455
, @f float(24)    = 85.455;

CREATE TABLE ctas_r
WITH (DISTRIBUTION = ROUND_ROBIN)
AS
SELECT @d*@f as result;
```

Notice that the column "result" carries forward the data type and nullability values of the expression. Carrying the data type forward can lead to subtle variances in values if you aren't careful.

Try this example:

```
SELECT result,result*@d
from result;

SELECT result,result*@d
from ctas_r;
```

The value stored for result is different. As the persisted value in the result column is used in other expressions, the error becomes even more significant.

	result	second_result
1	7302.98	624112.6708

	result	second_result
1	7302.984	624113.1

This is important for data migrations. Even though the second query is arguably more accurate, there's a problem. The data would be different compared to the source system, and that leads to questions of integrity in the migration. This is one of those rare cases where the "wrong" answer is actually the right one!

The reason we see a disparity between the two results is due to implicit type casting. In the first example, the table defines the column definition. When the row is inserted, an implicit type conversion occurs. In the second example, there is no implicit type conversion as the expression defines the data type of the column.

Notice also that the column in the second example has been defined as a NULLable column, whereas in the first example it has not. When the table was created in the first example, column nullability was explicitly defined. In the second example, it was left to the expression, and by default would result in a NULL definition.

To resolve these issues, you must explicitly set the type conversion and nullability in the SELECT portion of the

CTAS statement. You can't set these properties in 'CREATE TABLE'. The following example demonstrates how to fix the code:

```
DECLARE @d decimal(7,2) = 85.455
, @f float(24)    = 85.455

CREATE TABLE ctas_r
WITH (DISTRIBUTION = ROUND_ROBIN)
AS
SELECT ISNULL(CAST(@d*@f AS DECIMAL(7,2)),0) as result
```

Note the following:

- You can use CAST or CONVERT.
- Use ISNULL, not COALESCE, to force NULLability. See the following note.
- ISNULL is the outermost function.
- The second part of the ISNULL is a constant, 0.

> **NOTE**
>
> For the nullability to be correctly set, it's vital to use ISNULL and not COALESCE. COALESCE is not a deterministic function, and so the result of the expression will always be NULLable. ISNULL is different. It's deterministic. Therefore, when the second part of the ISNULL function is a constant or a literal, the resulting value will be NOT NULL.

Ensuring the integrity of your calculations is also important for table partition switching. Imagine you have this table defined as a fact table:

```
CREATE TABLE [dbo].[Sales]
(
    [date]      INT     NOT NULL
, [product]   INT     NOT NULL
, [store]     INT     NOT NULL
, [quantity]  INT     NOT NULL
, [price]     MONEY   NOT NULL
, [amount]    MONEY   NOT NULL
)
WITH
(   DISTRIBUTION = HASH([product])
,   PARTITION   (   [date] RANGE RIGHT FOR VALUES
                    (20000101,20010101,20020101
                    ,20030101,20040101,20050101
                    )
                )
);
```

However, the amount field is a calculated expression. It isn't part of the source data.

To create your partitioned dataset, you might want to use the following code:

```
CREATE TABLE [dbo].[Sales_in]
WITH
( DISTRIBUTION = HASH([product])
, PARTITION   (   [date] RANGE RIGHT FOR VALUES
                    (20000101,20010101
                    )
                )
)
AS
SELECT
    [date]
,   [product]
,   [store]
,   [quantity]
,   [price]
,   [quantity]*[price]  AS [amount]
FROM [stg].[source]
OPTION (LABEL = 'CTAS : Partition IN table : Create');
```

The query would run perfectly well. The problem comes when you try to do the partition switch. The table definitions don't match. To make the table definitions match, modify the CTAS to add an `ISNULL` function to preserve the column's nullability attribute.

```
CREATE TABLE [dbo].[Sales_in]
WITH
( DISTRIBUTION = HASH([product])
, PARTITION   (   [date] RANGE RIGHT FOR VALUES
                    (20000101,20010101
                    )
                )
)
AS
SELECT
  [date]
, [product]
, [store]
, [quantity]
, [price]
, ISNULL(CAST([quantity]*[price] AS MONEY),0) AS [amount]
FROM [stg].[source]
OPTION (LABEL = 'CTAS : Partition IN table : Create');
```

You can see that type consistency and maintaining nullability properties on a CTAS is an engineering best practice. It helps to maintain integrity in your calculations, and also ensures that partition switching is possible.

CTAS is one of the most important statements in Synapse SQL. Make sure you thoroughly understand it. See the CTAS documentation.

Next steps

For more development tips, see the development overview.

Table data types in Synapse SQL

8/19/2022 • 2 minutes to read • Edit Online

In this article, you'll find recommendations for defining table data types in Synapse SQL Dedicated Pool.

Data types

Synapse SQL Dedicated Pool supports the most commonly used data types. For a list of the supported data types, see data types in the CREATE TABLE statement. For Synapse SQL Serverless please refer to article Query storage files with serverless SQL pool in Azure Synapse Analytics and How to use OPENROWSET using serverless SQL pool in Azure Synapse Analytics

Minimize row length

Minimizing the size of data types shortens the row length, which leads to better query performance. Use the smallest data type that works for your data.

- Avoid defining character columns with a large default length. For example, if the longest value is 25 characters, then define your column as VARCHAR(25).
- Avoid using [NVARCHAR][NVARCHAR] when you only need VARCHAR.
- When possible, use NVARCHAR(4000) or VARCHAR(8000) instead of NVARCHAR(MAX) or VARCHAR(MAX).
- Avoid using floats and decimals with 0 (zero) scale. These should be TINYINT, SMALLINT, INT or BIGINT.

> **NOTE**
> If you are using PolyBase external tables to load your Synapse SQL tables, the defined length of the table row cannot exceed 1 MB. When a row with variable-length data exceeds 1 MB, you can load the row with BCP, but not with PolyBase.

Identify unsupported data types

If you are migrating your database from another SQL database, you might encounter data types that are not supported in Synapse SQL. Use this query to discover unsupported data types in your existing SQL schema.

```
SELECT  t.[name], c.[name], c.[system_type_id], c.[user_type_id], y.[is_user_defined], y.[name]
FROM sys.tables  t
JOIN sys.columns c on t.[object_id]    = c.[object_id]
JOIN sys.types   y on c.[user_type_id] = y.[user_type_id]
WHERE y.[name] IN ('geography','geometry','hierarchyid','image','text','ntext','sql_variant','xml')
 OR  y.[is_user_defined] = 1;
```

Workarounds for unsupported data types

The following list shows the data types that Synapse SQL does not support and gives alternatives that you can use instead of the unsupported data types.

UNSUPPORTED DATA TYPE	WORKAROUND
geometry	varbinary

UNSUPPORTED DATA TYPE	WORKAROUND
geography	varbinary
hierarchyid	nvarchar(4000)
image	varbinary
text	varchar
ntext	nvarchar
sql_variant	Split column into several strongly typed columns.
table	Convert to temporary tables or consider storing data to storage using CETAS.
timestamp	Rework code to use datetime2 and the CURRENT_TIMESTAMP function. Only constants are supported as defaults, therefore current_timestamp cannot be defined as a default constraint. If you need to migrate row version values from a timestamp typed column, then use BINARY(8) or VARBINARY(8) for NOT NULL or NULL row version values.
xml	varchar
user-defined type	Convert back to the native data type when possible.
default values	Default values support literals and constants only.

Next steps

For more information on developing tables, see Table Overview.

Guidance for designing distributed tables using dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 9 minutes to read • Edit Online

This article contains recommendations for designing hash-distributed and round-robin distributed tables in dedicated SQL pools.

This article assumes you are familiar with data distribution and data movement concepts in dedicated SQL pool. For more information, see Azure Synapse Analytics architecture.

What is a distributed table?

A distributed table appears as a single table, but the rows are actually stored across 60 distributions. The rows are distributed with a hash or round-robin algorithm.

Hash-distribution improves query performance on large fact tables, and is the focus of this article. **Round-robin distribution** is useful for improving loading speed. These design choices have a significant impact on improving query and loading performance.

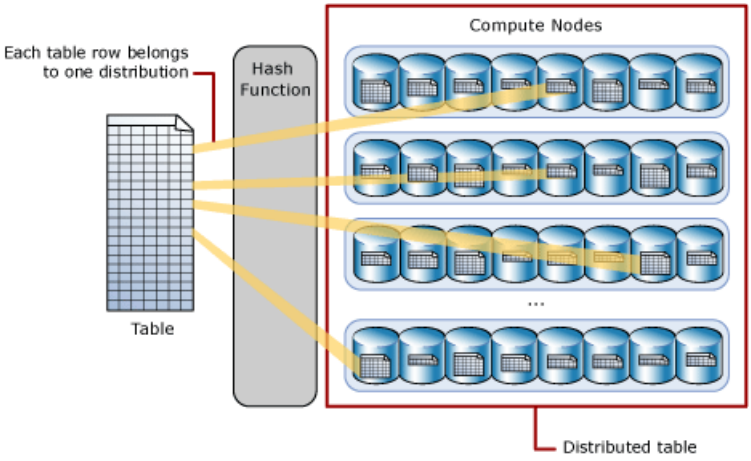
Another table storage option is to replicate a small table across all the Compute nodes. For more information, see Design guidance for replicated tables. To quickly choose among the three options, see Distributed tables in the tables overview.

As part of table design, understand as much as possible about your data and how the data is queried. For example, consider these questions:

- How large is the table?
- How often is the table refreshed?
- Do I have fact and dimension tables in a dedicated SQL pool?

Hash distributed

A hash-distributed table distributes table rows across the Compute nodes by using a deterministic hash function to assign each row to one distribution.



Since identical values always hash to the same distribution, SQL Analytics has built-in knowledge of the row locations. In dedicated SQL pool this knowledge is used to minimize data movement during queries, which improves query performance.

Hash-distributed tables work well for large fact tables in a star schema. They can have very large numbers of

rows and still achieve high performance. There are some design considerations that help you to get the performance the distributed system is designed to provide. Choosing a good distribution column is one such consideration that is described in this article.

Consider using a hash-distributed table when:

- The table size on disk is more than 2 GB.
- The table has frequent insert, update, and delete operations.

Round-robin distributed

A round-robin distributed table distributes table rows evenly across all distributions. The assignment of rows to distributions is random. Unlike hash-distributed tables, rows with equal values are not guaranteed to be assigned to the same distribution.

As a result, the system sometimes needs to invoke a data movement operation to better organize your data before it can resolve a query. This extra step can slow down your queries. For example, joining a round-robin table usually requires reshuffling the rows, which is a performance hit.

Consider using the round-robin distribution for your table in the following scenarios:

- When getting started as a simple starting point since it is the default
- If there is no obvious joining key
- If there is no good candidate column for hash distributing the table
- If the table does not share a common join key with other tables
- If the join is less significant than other joins in the query
- When the table is a temporary staging table

The tutorial Load New York taxicab data gives an example of loading data into a round-robin staging table.

Choose a distribution column

A hash-distributed table has a distribution column that is the hash key. For example, the following code creates a hash-distributed table with ProductKey as the distribution column.

```
CREATE TABLE [dbo].[FactInternetSales]
(   [ProductKey]            int          NOT NULL
,   [OrderDateKey]          int          NOT NULL
,   [CustomerKey]           int          NOT NULL
,   [PromotionKey]          int          NOT NULL
,   [SalesOrderNumber]      nvarchar(20) NOT NULL
,   [OrderQuantity]         smallint     NOT NULL
,   [UnitPrice]             money        NOT NULL
,   [SalesAmount]           money        NOT NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,  DISTRIBUTION = HASH([ProductKey])
);
```

Data stored in the distribution column can be updated. Updates to data in the distribution column could result in data shuffle operation.

Choosing a distribution column is an important design decision since the values in this column determine how the rows are distributed. The best choice depends on several factors, and usually involves tradeoffs. Once a distribution column is chosen, you cannot change it.

If you didn't choose the best column the first time, you can use CREATE TABLE AS SELECT (CTAS) to re-create the table with a different distribution column.

Choose a distribution column with data that distributes evenly

For best performance, all of the distributions should have approximately the same number of rows. When one or more distributions have a disproportionate number of rows, some distributions finish their portion of a parallel query before others. Since the query can't complete until all distributions have finished processing, each query is only as fast as the slowest distribution.

- Data skew means the data is not distributed evenly across the distributions
- Processing skew means that some distributions take longer than others when running parallel queries. This can happen when the data is skewed.

To balance the parallel processing, select a distribution column that:

- **Has many unique values.** The column can have duplicate values. All rows with the same value are assigned to the same distribution. Since there are 60 distributions, some distributions can have > 1 unique values while others may end with zero values.
- **Does not have NULLs, or has only a few NULLs.** For an extreme example, if all values in the column are NULL, all the rows are assigned to the same distribution. As a result, query processing is skewed to one distribution, and does not benefit from parallel processing.
- **Is not a date column**. All data for the same date lands in the same distribution. If several users are all filtering on the same date, then only 1 of the 60 distributions do all the processing work.

Choose a distribution column that minimizes data movement

To get the correct query result queries might move data from one Compute node to another. Data movement commonly happens when queries have joins and aggregations on distributed tables. Choosing a distribution column that helps minimize data movement is one of the most important strategies for optimizing performance of your dedicated SQL pool.

To minimize data movement, select a distribution column that:

- Is used in `JOIN`, `GROUP BY`, `DISTINCT`, `OVER`, and `HAVING` clauses. When two large fact tables have frequent joins, query performance improves when you distribute both tables on one of the join columns. When a table is not used in joins, consider distributing the table on a column that is frequently in the `GROUP BY` clause.
- Is *not* used in `WHERE` clauses. This could narrow the query to not run on all the distributions.
- Is *not* a date column. `WHERE` clauses often filter by date. When this happens, all the processing could run on only a few distributions.

What to do when none of the columns are a good distribution column

If none of your columns have enough distinct values for a distribution column, you can create a new column as a composite of one or more values. To avoid data movement during query execution, use the composite distribution column as a join column in queries.

Once you design a hash-distributed table, the next step is to load data into the table. For loading guidance, see Loading overview.

How to tell if your distribution column is a good choice

After data is loaded into a hash-distributed table, check to see how evenly the rows are distributed across the 60 distributions. The rows per distribution can vary up to 10% without a noticeable impact on performance.

Determine if the table has data skew

A quick way to check for data skew is to use DBCC PDW_SHOWSPACEUSED. The following SQL code returns the number of table rows that are stored in each of the 60 distributions. For balanced performance, the rows in your distributed table should be spread evenly across all the distributions.

```
-- Find data skew for a distributed table
DBCC PDW_SHOWSPACEUSED('dbo.FactInternetSales');
```

To identify which tables have more than 10% data skew:

1. Create the view `dbo.vTableSizes` that is shown in the Tables overview article.
2. Run the following query:

```
select *
from dbo.vTableSizes
where two_part_name in
    (
    select two_part_name
    from dbo.vTableSizes
    where row_count > 0
    group by two_part_name
    having (max(row_count * 1.000) - min(row_count * 1.000))/max(row_count * 1.000) >= .10
    )
order by two_part_name, row_count
;
```

Check query plans for data movement

A good distribution column enables joins and aggregations to have minimal data movement. This affects the way joins should be written. To get minimal data movement for a join on two hash-distributed tables, one of the join columns needs to be the distribution column. When two hash-distributed tables join on a distribution column of the same data type, the join does not require data movement. Joins can use additional columns without incurring data movement.

To avoid data movement during a join:

- The tables involved in the join must be hash distributed on **one** of the columns participating in the join.
- The data types of the join columns must match between both tables.
- The columns must be joined with an equals operator.
- The join type may not be a `CROSS JOIN`.

To see if queries are experiencing data movement, you can look at the query plan.

Resolve a distribution column problem

It is not necessary to resolve all cases of data skew. Distributing data is a matter of finding the right balance between minimizing data skew and data movement. It is not always possible to minimize both data skew and data movement. Sometimes the benefit of having the minimal data movement might outweigh the impact of having data skew.

To decide if you should resolve data skew in a table, you should understand as much as possible about the data volumes and queries in your workload. You can use the steps in the Query monitoring article to monitor the impact of skew on query performance. Specifically, look for how long it takes large queries to complete on individual distributions.

Since you cannot change the distribution column on an existing table, the typical way to resolve data skew is to re-create the table with a different distribution column.

Re-create the table with a new distribution column

This example uses CREATE TABLE AS SELECT to re-create a table with a different hash distribution column.

```
CREATE TABLE [dbo].[FactInternetSales_CustomerKey]
WITH (  CLUSTERED COLUMNSTORE INDEX
     ,  DISTRIBUTION =  HASH([CustomerKey])
     ,  PARTITION       ( [OrderDateKey] RANGE RIGHT FOR VALUES (   20000101, 20010101, 20020101, 20030101
                                                                ,   20040101, 20050101, 20060101, 20070101
                                                                ,   20080101, 20090101, 20100101, 20110101
                                                                ,   20120101, 20130101, 20140101, 20150101
                                                                ,   20160101, 20170101, 20180101, 20190101
                                                                ,   20200101, 20210101, 20220101, 20230101
                                                                ,   20240101, 20250101, 20260101, 20270101
                                                                ,   20280101, 20290101
                                                                )
                          )
    )
AS
SELECT  *
FROM    [dbo].[FactInternetSales]
OPTION  (LABEL  = 'CTAS : FactInternetSales_CustomerKey')
;

--Create statistics on new table
CREATE STATISTICS [ProductKey] ON [FactInternetSales_CustomerKey] ([ProductKey]);
CREATE STATISTICS [OrderDateKey] ON [FactInternetSales_CustomerKey] ([OrderDateKey]);
CREATE STATISTICS [CustomerKey] ON [FactInternetSales_CustomerKey] ([CustomerKey]);
CREATE STATISTICS [PromotionKey] ON [FactInternetSales_CustomerKey] ([PromotionKey]);
CREATE STATISTICS [SalesOrderNumber] ON [FactInternetSales_CustomerKey] ([SalesOrderNumber]);
CREATE STATISTICS [OrderQuantity] ON [FactInternetSales_CustomerKey] ([OrderQuantity]);
CREATE STATISTICS [UnitPrice] ON [FactInternetSales_CustomerKey] ([UnitPrice]);
CREATE STATISTICS [SalesAmount] ON [FactInternetSales_CustomerKey] ([SalesAmount]);

--Rename the tables
RENAME OBJECT [dbo].[FactInternetSales] TO [FactInternetSales_ProductKey];
RENAME OBJECT [dbo].[FactInternetSales_CustomerKey] TO [FactInternetSales];
```

Next steps

To create a distributed table, use one of these statements:

- CREATE TABLE (dedicated SQL pool)
- CREATE TABLE AS SELECT (dedicated SQL pool)

Primary key, foreign key, and unique key using dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

Learn about table constraints in dedicated SQL pool, including primary key, foreign key, and unique key.

Table constraints

Dedicated SQL pool supports these table constraints:

- PRIMARY KEY is only supported when NONCLUSTERED and NOT ENFORCED are both used.
- UNIQUE constraint is only supported when NOT ENFORCED is used.

For syntax, check ALTER TABLE and CREATE TABLE.

FOREIGN KEY constraint is not supported in dedicated SQL pool.

Remarks

Having primary key and/or unique key allows dedicated SQL pool engine to generate an optimal execution plan for a query. All values in a primary key column or a unique constraint column should be unique.

> **IMPORTANT**
> After creating a table with primary key or unique constraint in dedicated SQL pool, users need to make sure all values in those columns are unique. A violation of that may cause the query to return inaccurate result.

This example shows how a query may return inaccurate result if the primary key or unique constraint column includes duplicate values.

```
 -- Create table t1
CREATE TABLE t1 (a1 INT NOT NULL, b1 INT) WITH (DISTRIBUTION = ROUND_ROBIN)

-- Insert values to table t1 with duplicate values in column a1.
INSERT INTO t1 VALUES (1, 100)
INSERT INTO t1 VALUES (1, 1000)
INSERT INTO t1 VALUES (2, 200)
INSERT INTO t1 VALUES (3, 300)
INSERT INTO t1 VALUES (4, 400)

-- Run this query.  No primary key or unique constraint.  4 rows returned. Correct result.
SELECT a1, COUNT(*) AS total FROM t1 GROUP BY a1

/*
a1          total
----------- -----------
1           2
2           1
3           1
4           1

(4 rows affected)
*/

-- Add unique constraint
```

```
ALTER TABLE t1 ADD CONSTRAINT unique_t1_a1 unique (a1) NOT ENFORCED

-- Re-run this query.  5 rows returned.  Incorrect result.
SELECT a1, count(*) AS total FROM t1 GROUP BY a1

/*
a1          total
----------- -----------
2           1
4           1
1           1
3           1
1           1

(5 rows affected)
*/

-- Drop unique constraint.
ALTER TABLE t1 DROP CONSTRAINT unique_t1_a1

-- Add primary key constraint
ALTER TABLE t1 add CONSTRAINT PK_t1_a1 PRIMARY KEY NONCLUSTERED (a1) NOT ENFORCED

-- Re-run this query.  5 rows returned.  Incorrect result.
SELECT a1, COUNT(*) AS total FROM t1 GROUP BY a1

/*
a1          total
----------- -----------
2           1
4           1
1           1
3           1
1           1

(5 rows affected)
*/

-- Manually fix the duplicate values in a1
UPDATE t1 SET a1 = 0 WHERE b1 = 1000

-- Verify no duplicate values in column a1
SELECT * FROM t1

/*
a1          b1
----------- -----------
2           200
3           300
4           400
0           1000
1           100

(5 rows affected)
*/

-- Add unique constraint
ALTER TABLE t1 add CONSTRAINT unique_t1_a1 UNIQUE (a1) NOT ENFORCED

-- Re-run this query.  5 rows returned.  Correct result.
SELECT a1, COUNT(*) as total FROM t1 GROUP BY a1

/*
a1          total
----------- -----------
2           1
3           1
4           1
0           1
```

```
1           1

(5 rows affected)
*/

-- Drop unique constraint.
ALTER TABLE t1 DROP CONSTRAINT unique_t1_a1

-- Add primary key contraint
ALTER TABLE t1 ADD CONSTRAINT PK_t1_a1 PRIMARY KEY NONCLUSTERED (a1) NOT ENFORCED

-- Re-run this query.  5 rows returned.  Correct result.
SELECT a1, COUNT(*) AS total FROM t1 GROUP BY a1

/*
a1          total
----------- -----------
2           1
3           1
4           1
0           1
1           1

(5 rows affected)
*/
```

Examples

Create a dedicated SQL pool table with a primary key:

```
CREATE TABLE mytable (c1 INT PRIMARY KEY NONCLUSTERED NOT ENFORCED, c2 INT);
```

Create a dedicated SQL pool table with a unique constraint:

```
CREATE TABLE t6 (c1 INT UNIQUE NOT ENFORCED, c2 INT);
```

Next steps

After creating the tables for your dedicated SQL pool, the next step is to load data into the table. For a loading tutorial, see Loading data to dedicated SQL pool.

Use external tables with Synapse SQL

8/19/2022 • 17 minutes to read • Edit Online

An external table points to data located in Hadoop, Azure Storage blob, or Azure Data Lake Storage. External tables are used to read data from files or write data to files in Azure Storage. With Synapse SQL, you can use external tables to read external data using dedicated SQL pool or serverless SQL pool.

Depending on the type of the external data source, you can use two types of external tables:

- Hadoop external tables that you can use to read and export data in various data formats such as CSV, Parquet, and ORC. Hadoop external tables are available in dedicated SQL pools, but they aren't available in serverless SQL pools.
- Native external tables that you can use to read and export data in various data formats such as CSV and Parquet. Native external tables are available in serverless SQL pools, and they are in **public preview** in dedicated SQL pools.

The key differences between Hadoop and native external tables are presented in the following table:

EXTERNAL TABLE TYPE	HADOOP	NATIVE
Dedicated SQL pool	Available	Only Parquet tables are available in **public preview**.
Serverless SQL pool	Not available	Available
Supported formats	Delimited/CSV, Parquet, ORC, Hive RC, and RC	Serverless SQL pool: Delimited/CSV, Parquet, and Delta Lake Dedicated SQL pool: Parquet (preview)
Folder partition elimination	No	Partition elimination is available only in the partitioned tables created on Parquet or CSV formats that are synchronized from Apache Spark pools. You might create external tables on Parquet partitioned folders, but the partitioning columns will be inaccessible and ignored, while the partition elimination will not be applied. Do not create external tables on Delta Lake folders because they are not supported. Use Delta partitioned views if you need to query partitioned Delta Lake data.
File elimination (predicate pushdown)	No	Yes in serverless SQL pool. For the string pushdown, you need to use `Latin1_General_100_BIN2_UTF8` collation on the `VARCHAR` columns to enable pushdown.
Custom format for location	No	Yes, using wildcards like `/year=*/month=*/day=*` for Parquet or CSV formats. Custom folder paths are not available in Delta Lake. In the serverless SQL pool you can also use recursive wildcards `/logs/**` to reference Parquet or CSV files in any sub-folder beneath the referenced folder.
Recursive folder scan	Yes	Yes. In serverless SQL pools must be specified `/**` at the end of the location path. In Dedicated pool the folders are alwasy scanned recursively.

EXTERNAL TABLE TYPE	HADOOP	NATIVE
Storage authentication	Storage Access Key(SAK), AAD passthrough, Managed identity, Custom application Azure AD identity	Shared Access Signature(SAS), AAD passthrough, Managed identity, Custom application Azure AD identity.
Column mapping	Ordinal - the columns in the external table definition are mapped to the columns in the underlying Parquet files by position.	Serverless pool: by name. The columns in the external table definition are mapped to the columns in the underlying Parquet files by column name matching. Dedicated pool: ordinal matching. The columns in the external table definition are mapped to the columns in the underlying Parquet files by position.

> **NOTE**
> The native external tables are the recommended solution in the pools where they are generally available. If you need to access external data, always use the native tables in serverless pools. In dedicated pools, you should switch to the native tables for reading Parquet files once they are in GA. Use the Hadoop tables only if you need to access some types that are not supported in native external tables (for example - ORC, RC), or if the native version is not available.

External tables in dedicated SQL pool and serverless SQL pool

You can use external tables to:

- Query Azure Blob Storage and Azure Data Lake Gen2 with Transact-SQL statements.
- Store query results to files in Azure Blob Storage or Azure Data Lake Storage using CETAS.
- Import data from Azure Blob Storage and Azure Data Lake Storage and store it in a dedicated SQL pool (only Hadoop tables in dedicated pool).

> **NOTE**
> When used in conjunction with the CREATE TABLE AS SELECT statement, selecting from an external table imports data into a table within the **dedicated** SQL pool.
>
> If performance of Hadoop external tables in the dedicated pools do not satisfy your performance goals, consider loading external data into the Datawarehouse tables using the COPY statement.
>
> For a loading tutorial, see Use PolyBase to load data from Azure Blob Storage.

You can create external tables in Synapse SQL pools via the following steps:

1. CREATE EXTERNAL DATA SOURCE to reference an external Azure storage and specify the credential that should be used to access the storage.
2. CREATE EXTERNAL FILE FORMAT to describe format of CSV or Parquet files.
3. CREATE EXTERNAL TABLE on top of the files placed on the data source with the same file format.

Folder partition elimination

The native external tables in Synapse pools are able to ignore the files placed in the folders that are not relevant for the queries. If your files are stored in a folder hierarchy (for example - **/year=2020/month=03/day=16**) and the values for **year**, **month**, and **day** are exposed as the columns, the queries that contain filters like `year=2020` will read the files only from the subfolders placed within the **year=2020** folder. The files and folders placed in other folders (**year=2021** or **year=2022**) will be ignored in this query. This elimination is known as **partition elimination**.

The folder partition elimination is available in the native external tables that are synchronized from the Synapse Spark pools. If you have partitioned data set and you would like to leverage the partition elimination with the external tables that you create, use the partitioned views instead of the external tables.

File elimination

Some data formats such as Parquet and Delta contain file statistics for each column (for example, min/max values for each column). The queries that filter data will not read the files where the required column values do

not exist. The query will first explore min/max values for the columns used in the query predicate to find the files that do not contain the required data. These files will be ignored and eliminated from the query plan. This technique is also known as filter predicate pushdown and it can improve the performance of your queries. Filter pushdown is available in the serverless SQL pools on Parquet and Delta formats. To leverage filter pushdown for the string types, use the VARCHAR type with the `Latin1_General_100_BIN2_UTF8` collation.

Security

User must have `SELECT` permission on an external table to read the data. External tables access underlying Azure storage using the database scoped credential defined in data source using the following rules:

- Data source without credential enables external tables to access publicly available files on Azure storage.
- Data source can have a credential that enables external tables to access only the files on Azure storage using SAS token or workspace Managed Identity - For examples, see the Develop storage files storage access control article.

CREATE EXTERNAL DATA SOURCE

External data sources are used to connect to storage accounts. The complete documentation is outlined here.

Syntax for CREATE EXTERNAL DATA SOURCE

- Hadoop
- Native

External data sources with `TYPE=HADOOP` are available only in dedicated SQL pools.

```
CREATE EXTERNAL DATA SOURCE <data_source_name>
WITH
(    LOCATION         = '<prefix>://<path>'
     [, CREDENTIAL = <database scoped credential> ]
     , TYPE = HADOOP
)
[;]
```

Arguments for CREATE EXTERNAL DATA SOURCE

data_source_name

Specifies the user-defined name for the data source. The name must be unique within the database.

Location

LOCATION = `'<prefix>://<path>'` - Provides the connectivity protocol and path to the external data source. The following patterns can be used in location:

EXTERNAL DATA SOURCE	LOCATION PREFIX	LOCATION PATH
Azure Blob Storage	`wasb[s]`	`<container>@<storage_account>.blob.core.windows.net`
Azure Blob Storage	`http[s]`	`<storage_account>.blob.core.windows.net/<container>/s`
Azure Data Lake Store Gen 1	`http[s]`	`<storage_account>.azuredatalakestore.net/webhdfs/v1`
Azure Data Lake Store Gen 2	`http[s]`	`<storage_account>.dfs.core.windows.net/<container>/su`

`https:` prefix enables you to use subfolder in the path.

Credential

CREDENTIAL = `<database scoped credential>` is optional credential that will be used to authenticate on Azure storage. External data source without credential can access public storage account or use the caller's Azure AD identity to access files on storage.

- In dedicated SQL pool, database scoped credential can specify custom application identity, workspace Managed Identity, or SAK key.
- In serverless SQL pool, database scoped credential can specify workspace Managed Identity, or SAS key.

TYPE

TYPE = `HADOOP` is the option that specifies that Java-based technology should be used to access underlying files.

This parameter can't be used in serverless SQL pool that uses built-in native reader.

Example for CREATE EXTERNAL DATA SOURCE

- Hadoop
- Native

The following example creates a Hadoop external data source in dedicated SQL pool for Azure Data Lake Gen2 pointing to the New York data set:

```
CREATE DATABASE SCOPED CREDENTIAL [ADLS_credential]
WITH IDENTITY='SHARED ACCESS SIGNATURE',
SECRET = 'sv=2018-03-28&ss=bf&srt=sco&sp=rl&st=2019-10-14T12%3A10%3A25Z&se=2061-12-
31T12%3A10%3A00Z&sig=KlSU2ullCscyTS0An0nozEpo4tO5JAgGBvw%2FJX2lguw%3D'
GO

CREATE EXTERNAL DATA SOURCE AzureDataLakeStore
WITH
  -- Please note the abfss endpoint when your account has secure transfer enabled
  ( LOCATION = 'abfss://data@newyorktaxidataset.dfs.core.windows.net' ,
    CREDENTIAL = ADLS_credential ,
    TYPE = HADOOP
  ) ;
```

The following example creates an external data source for Azure Data Lake Gen2 pointing to the publicly available New York data set:

```
CREATE EXTERNAL DATA SOURCE YellowTaxi
WITH ( LOCATION = 'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/',
       TYPE = HADOOP)
```

CREATE EXTERNAL FILE FORMAT

Creates an external file format object that defines external data stored in Azure Blob Storage or Azure Data Lake Storage. Creating an external file format is a prerequisite for creating an external table. The complete documentation is here.

By creating an external file format, you specify the actual layout of the data referenced by an external table.

Syntax for CREATE EXTERNAL FILE FORMAT

```
-- Create an external file format for PARQUET files.
CREATE EXTERNAL FILE FORMAT file_format_name
WITH (
    FORMAT_TYPE = PARQUET
    [ , DATA_COMPRESSION = {
        'org.apache.hadoop.io.compress.SnappyCodec'
      | 'org.apache.hadoop.io.compress.GzipCodec'      }
    ]);

--Create an external file format for DELIMITED TEXT files
CREATE EXTERNAL FILE FORMAT file_format_name
WITH (
    FORMAT_TYPE = DELIMITEDTEXT
    [ , DATA_COMPRESSION = 'org.apache.hadoop.io.compress.GzipCodec' ]
    [ , FORMAT_OPTIONS ( <format_options> [ ,...n  ] ) ]
    );

<format_options> ::=
{
    FIELD_TERMINATOR = field_terminator
    | STRING_DELIMITER = string_delimiter
    | FIRST_ROW = integer
    | USE_TYPE_DEFAULT = { TRUE | FALSE }
    | ENCODING = {'UTF8' | 'UTF16'}
    | PARSER_VERSION = {'parser_version'}
}
```

Arguments for CREATE EXTERNAL FILE FORMAT

file_format_name- Specifies a name for the external file format.

FORMAT_TYPE = [PARQUET | DELIMITEDTEXT]- Specifies the format of the external data.

- PARQUET - Specifies a Parquet format.
- DELIMITEDTEXT - Specifies a text format with column delimiters, also called field terminators.

FIELD_TERMINATOR = *field_terminator* - Applies only to delimited text files. The field terminator specifies one or more characters that mark the end of each field (column) in the text-delimited file. The default is the pipe character ('|').

Examples:

- FIELD_TERMINATOR = '|'
- FIELD_TERMINATOR = ' '
- FIELD_TERMINATOR = '\t'

STRING_DELIMITER = *string_delimiter* - Specifies the field terminator for data of type string in the text-delimited file. The string delimiter is one or more characters in length and is enclosed with single quotes. The default is the empty string ("").

Examples:

- STRING_DELIMITER = '""'
- STRING_DELIMITER = '*'
- STRING_DELIMITER = ','

FIRST_ROW = *First_row_int* - Specifies the row number that is read first and applies to all files. Setting the value to two causes the first row in every file (header row) to be skipped when the data is loaded. Rows are skipped based on the existence of row terminators (/r/n, /r, /n).

USE_TYPE_DEFAULT = { TRUE | **FALSE** } - Specifies how to handle missing values in delimited text files when retrieving data from the text file.

> **NOTE**
> Please note that USE_TYPE_DEFAULT=true is not supported for FORMAT_TYPE = DELIMITEDTEXT, PARSER_VERSION = '2.0'.

TRUE - If you're retrieving data from the text file, store each missing value by using the default value's data type for the corresponding column in the external table definition. For example, replace a missing value with:

- 0 if the column is defined as a numeric column. Decimal columns aren't supported and will cause an error.
- Empty string ("") if the column is a string column.
- 1900-01-01 if the column is a date column.

FALSE - Store all missing values as NULL. Any NULL values that are stored by using the word NULL in the delimited text file are imported as the string 'NULL'.

Encoding = {'UTF8' | 'UTF16'} - Serverless SQL pool can read UTF8 and UTF16 encoded delimited text files.

DATA_COMPRESSION = *data_compression_method* - This argument specifies the data compression method for the external data.

The PARQUET file format type supports the following compression methods:

- DATA_COMPRESSION = 'org.apache.hadoop.io.compress.GzipCodec'
- DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'

When reading from PARQUET external tables, this argument is ignored, but is used when writing to external tables using CETAS.

The DELIMITEDTEXT file format type supports the following compression method:

- DATA_COMPRESSION = 'org.apache.hadoop.io.compress.GzipCodec'

PARSER_VERSION = 'parser_version' Specifies parser version to be used when reading CSV files. The available parser versions are `1.0` and `2.0`. This option is available only in serverless SQL pools.

Example for CREATE EXTERNAL FILE FORMAT

The following example creates an external file format for census files:

```
CREATE EXTERNAL FILE FORMAT census_file_format
WITH
(
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
)
```

CREATE EXTERNAL TABLE

The CREATE EXTERNAL TABLE command creates an external table for Synapse SQL to access data stored in Azure Blob Storage or Azure Data Lake Storage.

Syntax for CREATE EXTERNAL TABLE

```
CREATE EXTERNAL TABLE { database_name.schema_name.table_name | schema_name.table_name | table_name }
    ( <column_definition> [ ,...n ] )
    WITH (
        LOCATION = 'folder_or_filepath',
        DATA_SOURCE = external_data_source_name,
        FILE_FORMAT = external_file_format_name
        [, TABLE_OPTIONS = N'{"READ_OPTIONS":["ALLOW_INCONSISTENT_READS"]}' ]
        [, <reject_options> [ ,...n ] ]
    )
[;]

<column_definition> ::=
column_name <data_type>
    [ COLLATE collation_name ]

<reject_options> ::=
{
    | REJECT_TYPE = value,
    | REJECT_VALUE = reject_value,
    | REJECT_SAMPLE_VALUE = reject_sample_value,
    | REJECTED_ROW_LOCATION = '/REJECT_Directory'
}
```

Arguments CREATE EXTERNAL TABLE

{ database_name.schema_name.table_name | schema_name.table_name | table_name }

The one to three-part name of the table to create. For an external table, Synapse SQL pool stores only the table metadata. No actual data is moved or stored in Synapse SQL database.

<column_definition>, *...n*]

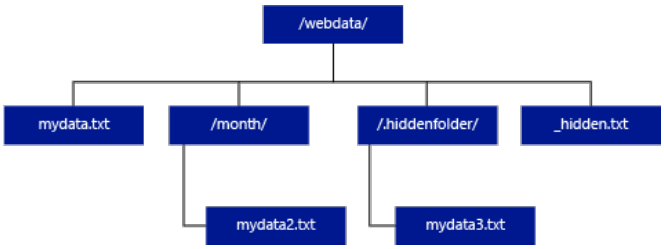
CREATE EXTERNAL TABLE supports the ability to configure column name, data type, and collation. You can't use the DEFAULT CONSTRAINT on external tables.

> **IMPORTANT**
> The column definitions, including the data types and number of columns, must match the data in the external files. If there's a mismatch, the file rows will be rejected when querying the actual data. See reject options to control rejected rows behavior.

When reading from Parquet files, you can specify only the columns you want to read and skip the rest.

LOCATION = '*folder_or_filepath*'

Specifies the folder or the file path and file name for the actual data in Azure Blob Storage. The location starts from the root folder. The root folder is the data location specified in the external data source.



Unlike Hadoop external tables, native external tables don't return subfolders unless you specify /** at the end of path. In this example, if LOCATION='/webdata/', a serverless SQL pool query, will return rows from mydata.txt. It won't return mydata2.txt and mydata3.txt because they're located in a subfolder. Hadoop tables will return all files within any sub-folder.

Both Hadoop and native external tables will skip the files with the names that begin with an underline (_) or a period (.).

DATA_SOURCE = *external_data_source_name*

Specifies the name of the external data source that contains the location of the external data. To create an external data source, use CREATE EXTERNAL DATA SOURCE.

FILE_FORMAT = *external_file_format_name*

Specifies the name of the external file format object that stores the file type and compression method for the external data. To create an external file format, use CREATE EXTERNAL FILE FORMAT.

Reject Options

> **NOTE**
> Rejected rows feature is in Public Preview. Please note that rejected rows feature works for delimited text files and PARSER_VERSION 1.0.

You can specify reject parameters that determine how service will handle *dirty* records it retrieves from the external data source. A data record is considered 'dirty' if actual data types don't match the column definitions of the external table.

When you don't specify or change reject options, service uses default values. This information about the reject parameters is stored as additional metadata when you create an external table with CREATE EXTERNAL TABLE statement. When a future SELECT statement or SELECT INTO SELECT statement selects data from the external table, service will use the reject options to determine the number of rows that can be rejected before the actual query fails. The query will return (partial) results until the reject threshold is exceeded. It then fails with the appropriate error message.

REJECT_TYPE = **value**

This is the only supported value at the moment. Clarifies that the REJECT_VALUE option is specified as a literal value.

value

REJECT_VALUE is a literal value. The query will fail when the number of rejected rows exceeds *reject_value*.

For example, if REJECT_VALUE = 5 and REJECT_TYPE = value, the SELECT query will fail after five rows have been rejected.

REJECT_VALUE = *reject_value*

Specifies the number of rows that can be rejected before the query fails.

For REJECT_TYPE = value, *reject_value* must be an integer between 0 and 2,147,483,647.

REJECTED_ROW_LOCATION = *Directory Location*

Specifies the directory within the External Data Source that the rejected rows and the corresponding error file should be written. If the specified path doesn't exist, service will create one on your behalf. A child directory is created with the name "*rejectedrows*". *The* '" character ensures that the directory is escaped for other data processing unless explicitly named in the location parameter. Within this directory, there's a folder created based

on the time of load submission in the format YearMonthDay_HourMinuteSecond_StatementID (Ex. 20180330-173205-559EE7D2-196D-400A-806D-3BF5D007F891). You can use statement id to correlate folder with query that generated it. In this folder, two files are written: error.json file and the data file.

error.json file contains json array with encountered errors related to rejected rows. Each element representing error contains following attributes:

ATTRIBUTE	DESCRIPTION
Error	Reason why row is rejected.
Row	Rejected row ordinal number in file.
Column	Rejected column ordinal number.
Value	Rejected column value. If the value is larger than 100 characters, only the first 100 characters will be displayed.
File	Path to file that row belongs to.

TABLE_OPTIONS

TABLE_OPTIONS = *json options* - Specifies the set of options that describe how to read the underlying files. Currently, the only option that is available is `"READ_OPTIONS":["ALLOW_INCONSISTENT_READS"]` that instructs the external table to ignore the updates that are made on the underlying files, even if this might cause some inconsistent read operations. Use this option only in special cases where you have frequently appended files. This option is available in serverless SQL pool for CSV format.

Permissions CREATE EXTERNAL TABLE

To select from an external table, you need proper credentials with list and read permissions.

Example CREATE EXTERNAL TABLE

The following example creates an external table. It returns the first row:

```
CREATE EXTERNAL TABLE census_external_table
(
    decennialTime varchar(20),
    stateName varchar(100),
    countyName varchar(100),
    population int,
    race varchar(50),
    sex    varchar(10),
    minAge int,
    maxAge int
)
WITH (
    LOCATION = '/parquet/',
    DATA_SOURCE = population_ds,
    FILE_FORMAT = census_file_format
)
GO

SELECT TOP 1 * FROM census_external_table
```

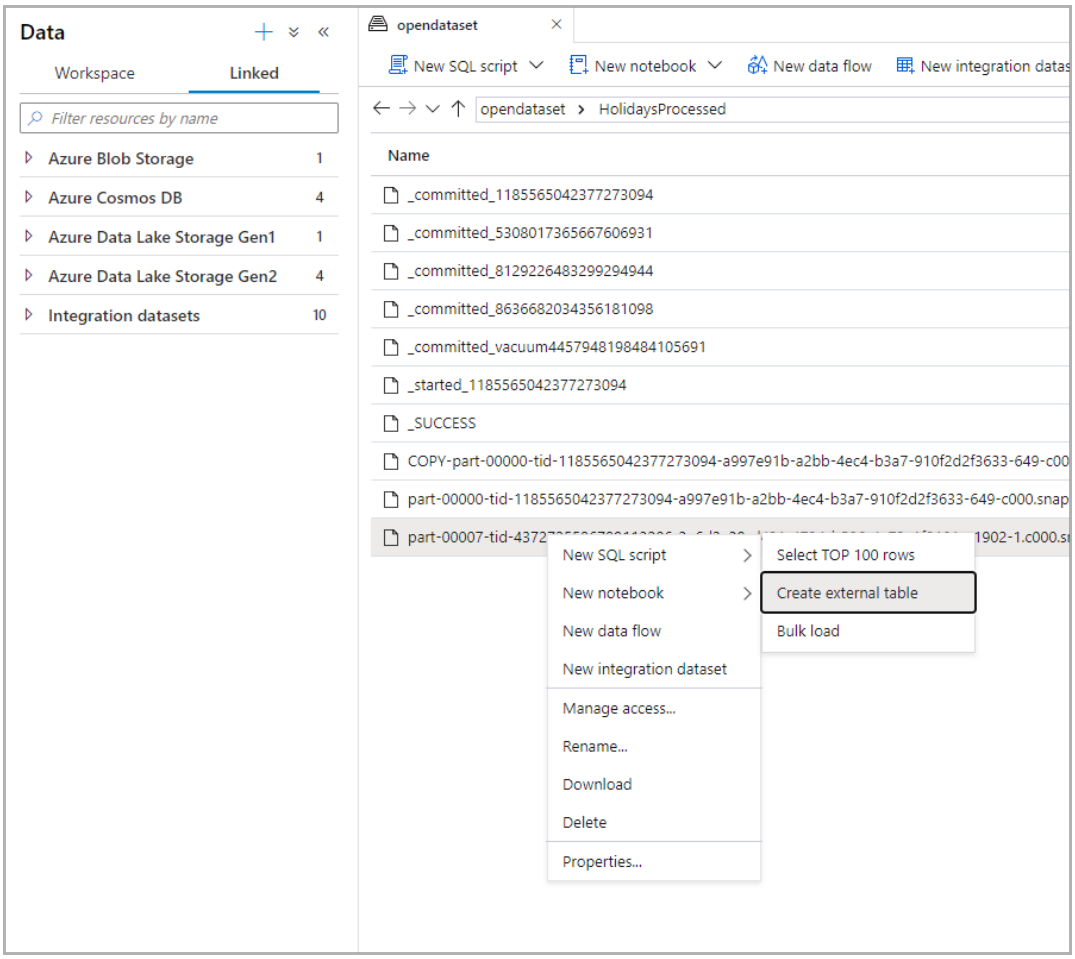
Create and query external tables from a file in Azure Data Lake

Using Data Lake exploration capabilities of Synapse Studio you can now create and query an external table using Synapse SQL pool with a simple right-click on the file. The one-click gesture to create external tables from the ADLS Gen2 storage account is only supported for Parquet files.

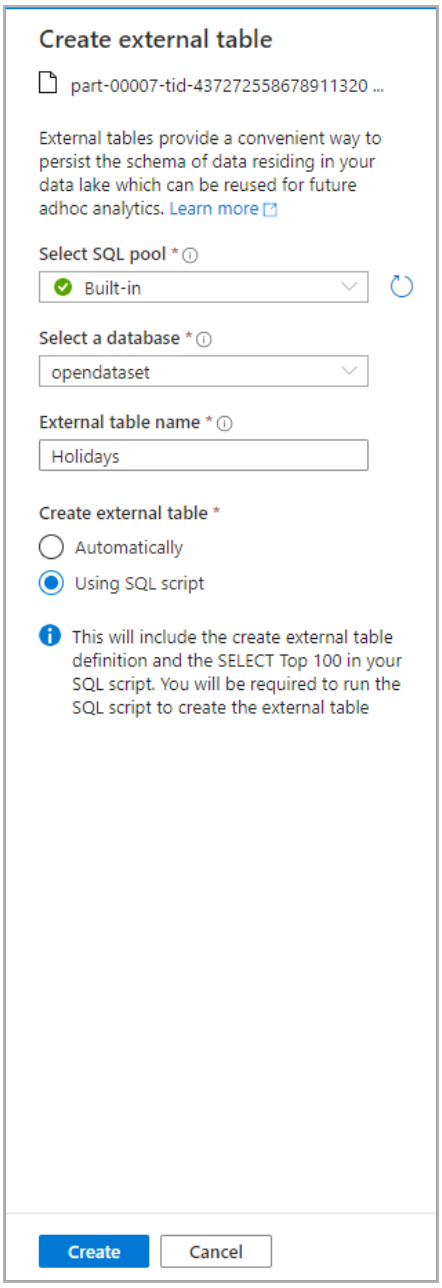
Prerequisites

- You must have access to the workspace with at least the `Storage Blob Data Contributor` access role to the ADLS Gen2 Account or Access Control Lists (ACL) that enable you to query the files.
- You must have at least permissions to create and query external tables on the Synapse SQL pool (dedicated or serverless).

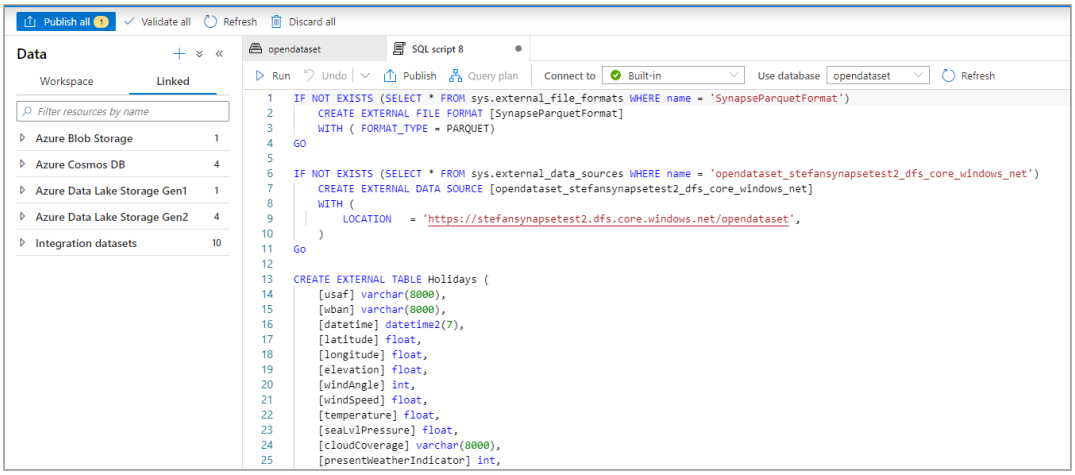
From the Data panel, select the file that you would like to create the external table from:



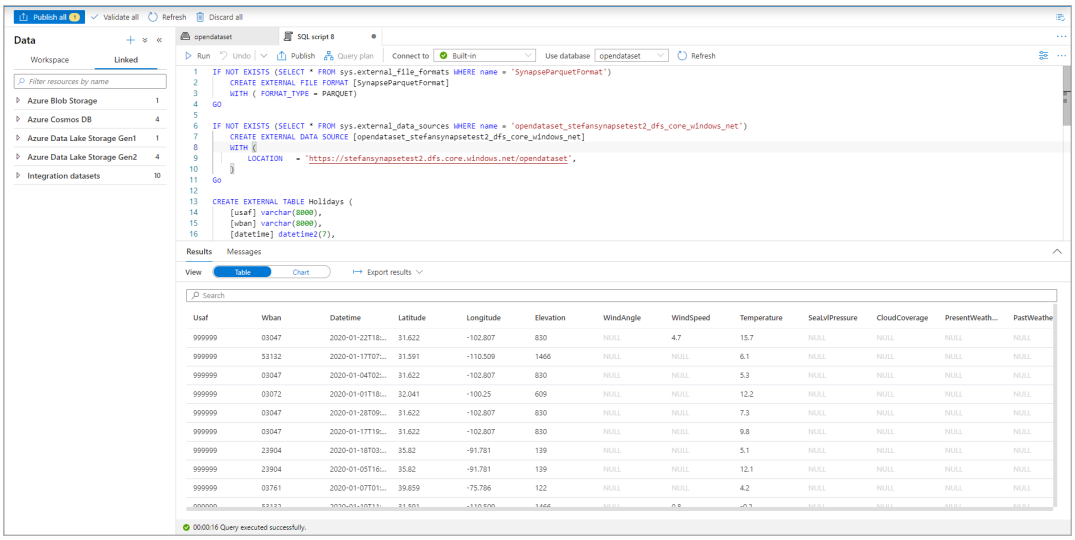
A dialog window will open. Select dedicated SQL pool or serverless SQL pool, give a name to the table and select open script:



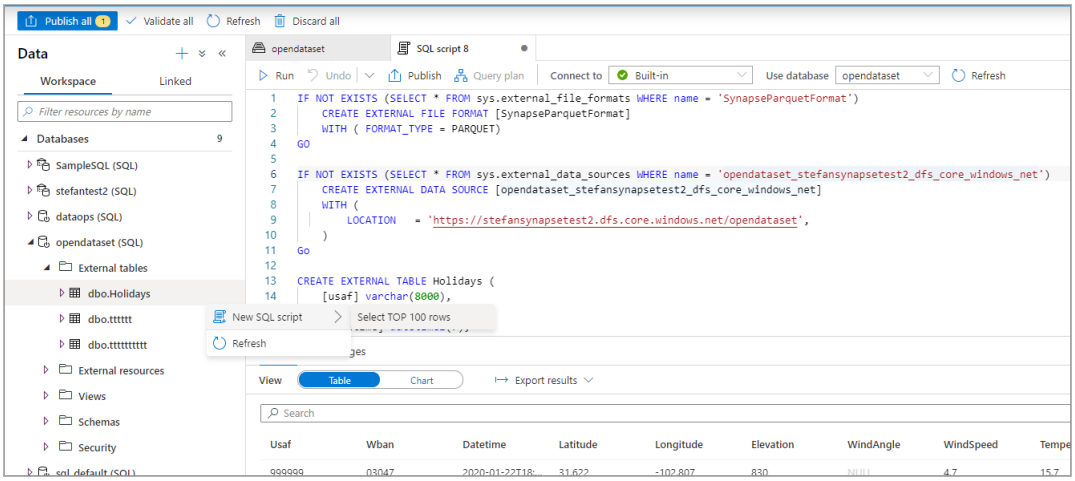
The SQL Script is autogenerated inferring the schema from the file:



Run the script. The script will automatically run a Select Top 100 *.:



The external table is now created, for future exploration of the content of this external table the user can query it directly from the Data pane:



Next steps

See the CETAS article for how to save query results to an external table in Azure Storage. Or you can start querying Apache Spark for Azure Synapse external tables.

Indexes on dedicated SQL pool tables in Azure Synapse Analytics

8/19/2022 • 14 minutes to read • Edit Online

Recommendations and examples for indexing tables in dedicated SQL pool in Azure Synapse Analytics.

Index types

Dedicated SQL pool offers several indexing options including clustered columnstore indexes, clustered indexes and nonclustered indexes, and a non-index option also known as heap.

To create a table with an index, see the CREATE TABLE (dedicated SQL pool) documentation.

Clustered columnstore indexes

By default, dedicated SQL pool creates a clustered columnstore index when no index options are specified on a table. Clustered columnstore tables offer both the highest level of data compression and the best overall query performance. Clustered columnstore tables will generally outperform clustered index or heap tables and are usually the best choice for large tables. For these reasons, clustered columnstore is the best place to start when you are unsure of how to index your table.

To create a clustered columnstore table, simply specify `CLUSTERED COLUMNSTORE INDEX` in the WITH clause, or leave the WITH clause off:

```
CREATE TABLE myTable
  (
    id int NOT NULL,
    lastName varchar(20),
    zipCode varchar(6)
  )
WITH ( CLUSTERED COLUMNSTORE INDEX );
```

There are a few scenarios where clustered columnstore may not be a good option:

- Columnstore tables do not support varchar(max), nvarchar(max), and varbinary(max). Consider heap or clustered index instead.
- Columnstore tables may be less efficient for transient data. Consider heap and perhaps even temporary tables.
- Small tables with less than 60 million rows. Consider heap tables.

Heap tables

When you are temporarily landing data in dedicated SQL pool, you may find that using a heap table makes the overall process faster. This is because loads to heaps are faster than to index tables and in some cases the subsequent read can be done from cache. If you are loading data only to stage it before running more transformations, loading the table to heap table is much faster than loading the data to a clustered columnstore table. In addition, loading data to a temporary table loads faster than loading a table to permanent storage. After data loading, you can create indexes in the table for faster query performance.

Cluster columnstore tables begin to achieve optimal compression once there is more than 60 million rows. For small lookup tables, less than 60 million rows, consider using HEAP or clustered index for faster query

performance.

To create a heap table, simply specify HEAP in the WITH clause:

```
CREATE TABLE myTable
  (
    id int NOT NULL,
    lastName varchar(20),
    zipCode varchar(6)
  )
WITH ( HEAP );
```

Clustered and nonclustered indexes

Clustered indexes may outperform clustered columnstore tables when a single row needs to be quickly retrieved. For queries where a single or very few row lookup is required to perform with extreme speed, consider a clustered index or nonclustered secondary index. The disadvantage to using a clustered index is that only queries that benefit are the ones that use a highly selective filter on the clustered index column. To improve filter on other columns, a nonclustered index can be added to other columns. However, each index that is added to a table adds both space and processing time to loads.

To create a clustered index table, simply specify CLUSTERED INDEX in the WITH clause:

```
CREATE TABLE myTable
  (
    id int NOT NULL,
    lastName varchar(20),
    zipCode varchar(6)
  )
WITH ( CLUSTERED INDEX (id) );
```

To add a non-clustered index on a table, use the following syntax:

```
CREATE INDEX zipCodeIndex ON myTable (zipCode);
```

Optimizing clustered columnstore indexes

Clustered columnstore tables organize data into segments. Having high segment quality is critical to achieving optimal query performance on a columnstore table. Segment quality can be measured by the number of rows in a compressed row group. Segment quality is most optimal where there are at least 100 K rows per compressed row group and gain in performance as the number of rows per row group approach 1,048,576 rows, which is the most rows a row group can contain.

The below view can be created and used on your system to compute the average rows per row group and identify any suboptimal cluster columnstore indexes. The last column on this view generates a SQL statement that can be used to rebuild your indexes.

```
CREATE VIEW dbo.vColumnstoreDensity
AS
SELECT
        GETDATE()                                                                   AS [execution_date]
,       DB_Name()                                                                   AS [database_name]
,       s.name                                                                      AS [schema_name]
,       t.name                                                                      AS [table_name]
,    COUNT(DISTINCT rg.[partition_number])                    AS [table_partition_count]
,       SUM(rg.[total_rows])                                                        AS [row_count_total]
,       SUM(rg.[total_rows])/COUNT(DISTINCT rg.[distribution_id])                   AS
[row_count_per_distribution_MAX]
,    CEILING    ((SUM(rg.[total_rows])*1.0/COUNT(DISTINCT rg.[distribution_id]))/1048576) AS
[rowgroup_per_distribution_MAX]
,       SUM(CASE WHEN rg.[State] = 0 THEN 1                   ELSE 0    END)    AS
[INVISIBLE_rowgroup_count]
,       SUM(CASE WHEN rg.[State] = 0 THEN rg.[total_rows]     ELSE 0    END)    AS [INVISIBLE_rowgroup_rows]
,       MIN(CASE WHEN rg.[State] = 0 THEN rg.[total_rows]     ELSE NULL END)    AS
[INVISIBLE_rowgroup_rows_MIN]
,       MAX(CASE WHEN rg.[State] = 0 THEN rg.[total_rows]     ELSE NULL END)    AS
[INVISIBLE_rowgroup_rows_MAX]
,       AVG(CASE WHEN rg.[State] = 0 THEN rg.[total_rows]     ELSE NULL END)    AS
[INVISIBLE_rowgroup_rows_AVG]
,       SUM(CASE WHEN rg.[State] = 1 THEN 1                   ELSE 0    END)    AS [OPEN_rowgroup_count]
,       SUM(CASE WHEN rg.[State] = 1 THEN rg.[total_rows]     ELSE 0    END)    AS [OPEN_rowgroup_rows]
,       MIN(CASE WHEN rg.[State] = 1 THEN rg.[total_rows]     ELSE NULL END)    AS [OPEN_rowgroup_rows_MIN]
,       MAX(CASE WHEN rg.[State] = 1 THEN rg.[total_rows]     ELSE NULL END)    AS [OPEN_rowgroup_rows_MAX]
,       AVG(CASE WHEN rg.[State] = 1 THEN rg.[total_rows]     ELSE NULL END)    AS [OPEN_rowgroup_rows_AVG]
,       SUM(CASE WHEN rg.[State] = 2 THEN 1                   ELSE 0    END)    AS [CLOSED_rowgroup_count]
,       SUM(CASE WHEN rg.[State] = 2 THEN rg.[total_rows]     ELSE 0    END)    AS [CLOSED_rowgroup_rows]
,       MIN(CASE WHEN rg.[State] = 2 THEN rg.[total_rows]     ELSE NULL END)    AS
[CLOSED_rowgroup_rows_MIN]
,       MAX(CASE WHEN rg.[State] = 2 THEN rg.[total_rows]     ELSE NULL END)    AS
[CLOSED_rowgroup_rows_MAX]
,       AVG(CASE WHEN rg.[State] = 2 THEN rg.[total_rows]     ELSE NULL END)    AS
[CLOSED_rowgroup_rows_AVG]
,       SUM(CASE WHEN rg.[State] = 3 THEN 1                   ELSE 0    END)    AS
[COMPRESSED_rowgroup_count]
,       SUM(CASE WHEN rg.[State] = 3 THEN rg.[total_rows]     ELSE 0    END)    AS
[COMPRESSED_rowgroup_rows]
,       SUM(CASE WHEN rg.[State] = 3 THEN rg.[deleted_rows]   ELSE 0    END)    AS
[COMPRESSED_rowgroup_rows_DELETED]
,       MIN(CASE WHEN rg.[State] = 3 THEN rg.[total_rows]     ELSE NULL END)    AS
[COMPRESSED_rowgroup_rows_MIN]
,       MAX(CASE WHEN rg.[State] = 3 THEN rg.[total_rows]     ELSE NULL END)    AS
[COMPRESSED_rowgroup_rows_MAX]
,       AVG(CASE WHEN rg.[State] = 3 THEN rg.[total_rows]     ELSE NULL END)    AS
[COMPRESSED_rowgroup_rows_AVG]
,       'ALTER INDEX ALL ON ' + s.name + '.' + t.NAME + ' REBUILD;'               AS [Rebuild_Index_SQL]
FROM    sys.[pdw_nodes_column_store_row_groups] rg
JOIN    sys.[pdw_nodes_tables] nt                   ON  rg.[object_id]          = nt.[object_id]
                                                    AND rg.[pdw_node_id]        = nt.[pdw_node_id]
                                                    AND rg.[distribution_id]    = nt.[distribution_id]
JOIN    sys.[pdw_table_mappings] mp                 ON  nt.[name]               = mp.[physical_name]
JOIN    sys.[tables] t                              ON  mp.[object_id]          = t.[object_id]
JOIN    sys.[schemas] s                             ON t.[schema_id]            = s.[schema_id]
GROUP BY
        s.[name]
,       t.[name];
```

Now that you have created the view, run this query to identify tables with row groups with less than 100 K rows. You may want to increase the threshold of 100 K if you are looking for more optimal segment quality.

```
SELECT    *
FROM    [dbo].[vColumnstoreDensity]
WHERE    COMPRESSED_rowgroup_rows_AVG < 100000
        OR INVISIBLE_rowgroup_rows_AVG < 100000;
```

Once you have run the query, you can begin to look at the data and analyze your results. This table explains what to look for in your row group analysis.

COLUMN	HOW TO USE THIS DATA
[table_partition_count]	If the table is partitioned, then you may expect to see higher Open row group counts. Each partition in the distribution could in theory have an open row group associated with it. Factor this into your analysis. A small table that has been partitioned could be optimized by removing the partitioning altogether as this would improve compression.
[row_count_total]	Total row count for the table. For example, you can use this value to calculate percentage of rows in the compressed state.
[row_count_per_distribution_MAX]	If all rows are evenly distributed this value would be the target number of rows per distribution. Compare this value with the compressed_rowgroup_count.
[COMPRESSED_rowgroup_rows]	Total number of rows in columnstore format for the table.
[COMPRESSED_rowgroup_rows_AVG]	If the average number of rows is significantly less than the maximum # of rows for a row group, then consider using CTAS or ALTER INDEX REBUILD to recompress the data
[COMPRESSED_rowgroup_count]	Number of row groups in columnstore format. If this number is very high in relation to the table, it is an indicator that the columnstore density is low.
[COMPRESSED_rowgroup_rows_DELETED]	Rows are logically deleted in columnstore format. If the number is high relative to table size, consider recreating the partition or rebuilding the index as this removes them physically.
[COMPRESSED_rowgroup_rows_MIN]	Use this with the AVG and MAX columns to understand the range of values for the row groups in your columnstore. A low number over the load threshold (102,400 per partition aligned distribution) suggests that optimizations are available in the data load
[COMPRESSED_rowgroup_rows_MAX]	As above
[OPEN_rowgroup_count]	Open row groups are normal. One would reasonably expect one OPEN row group per table distribution (60). Excessive numbers suggest data loading across partitions. Double check the partitioning strategy to make sure it is sound
[OPEN_rowgroup_rows]	Each row group can have 1,048,576 rows in it as a maximum. Use this value to see how full the open row groups are currently

COLUMN	HOW TO USE THIS DATA
[OPEN_rowgroup_rows_MIN]	Open groups indicate that data is either being trickle loaded into the table or that the previous load spilled over remaining rows into this row group. Use the MIN, MAX, AVG columns to see how much data is sat in OPEN row groups. For small tables, it could be 100% of all the data! In which case ALTER INDEX REBUILD to force the data to columnstore.
[OPEN_rowgroup_rows_MAX]	As above
[OPEN_rowgroup_rows_AVG]	As above
[CLOSED_rowgroup_rows]	Look at the closed row group rows as a sanity check.
[CLOSED_rowgroup_count]	The number of closed row groups should be low if any are seen at all. Closed row groups can be converted to compressed row groups using the ALTER INDEX ... REORGANIZE command. However, this is not normally required. Closed groups are automatically converted to columnstore row groups by the background "tuple mover" process.
[CLOSED_rowgroup_rows_MIN]	Closed row groups should have a very high fill rate. If the fill rate for a closed row group is low, then further analysis of the columnstore is required.
[CLOSED_rowgroup_rows_MAX]	As above
[CLOSED_rowgroup_rows_AVG]	As above
[Rebuild_Index_SQL]	SQL to rebuild columnstore index for a table

Impact of index maintenance

The column `Rebuild_Index_SQL` in the `vColumnstoreDensity` view contains an `ALTER INDEX REBUILD` statement that can be used to rebuild your indexes. When rebuilding your indexes, be sure that you allocate enough memory to the session that rebuilds your index. To do this, increase the resource class of a user that has permissions to rebuild the index on this table to the recommended minimum. For an example, see Rebuilding indexes to improve segment quality later in this article.

For a table with an ordered clustered columnstore index, `ALTER INDEX REBUILD` will re-sort the data using tempdb. Monitor tempdb during rebuild operations. If you need more tempdb space, scale up the database pool. Scale back down once the index rebuild is complete.

For a table with an ordered clustered columnstore index, `ALTER INDEX REORGANIZE` does not re-sort the data. To re-sort data, use `ALTER INDEX REBUILD`.

For more information on ordered clustered columnstore indexes, see Performance tuning with ordered clustered columnstore index.

Causes of poor columnstore index quality

If you have identified tables with poor segment quality, you want to identify the root cause. Below are some other common causes of poor segment quality:

1. Memory pressure when index was built
2. High volume of DML operations
3. Small or trickle load operations
4. Too many partitions

These factors can cause a columnstore index to have significantly less than the optimal 1 million rows per row group. They can also cause rows to go to the delta row group instead of a compressed row group.

Memory pressure when index was built

The number of rows per compressed row group are directly related to the width of the row and the amount of memory available to process the row group. When rows are written to columnstore tables under memory pressure, columnstore segment quality may suffer. Therefore, the best practice is to give the session that is writing to your columnstore index tables access to as much memory as possible. Since there is a trade-off between memory and concurrency, the guidance on the right memory allocation depends on the data in each row of your table, the data warehouse units allocated to your system, and the number of concurrency slots you can give to the session that is writing data to your table.

High volume of DML operations

A high volume of DML operations that update and delete rows can introduce inefficiency into the columnstore. This is especially true when most the rows in a row group are modified.

- Deleting a row from a compressed row group only logically marks the row as deleted. The row remains in the compressed row group until the partition or table is rebuilt.
- Inserting a row adds the row to an internal rowstore table called a delta row group. The inserted row is not converted to columnstore until the delta row group is full and is marked as closed. Row groups are closed once they reach the maximum capacity of 1,048,576 rows.
- Updating a row in columnstore format is processed as a logical delete and then an insert. The inserted row may be stored in the delta store.

Batched update and insert operations that exceed the bulk threshold of 102,400 rows per partition-aligned distribution go directly to the columnstore format. However, assuming an even distribution, you would need to be modifying more than 6.144 million rows in a single operation for this to occur. If the number of rows for a given partition-aligned distribution is less than 102,400, the rows go to the delta store and stay there until sufficient rows have been inserted or modified to close the row group or the index has been rebuilt.

Small or trickle load operations

Small loads that flow into dedicated SQL pool are also sometimes known as trickle loads. They typically represent a near constant stream of data being ingested by the system. However, as this stream is near continuous the volume of rows is not particularly large. More often than not the data is significantly under the threshold required for a direct load to columnstore format.

In these situations, it is often better to land the data first in Azure blob storage and let it accumulate prior to loading. This technique is often known as *micro-batching*.

Too many partitions

Another thing to consider is the impact of partitioning on your clustered columnstore tables. Before partitioning, dedicated SQL pool already divides your data into 60 databases. Partitioning further divides your data. If you partition your data, then consider that **each** partition needs at least 1 million rows to benefit from a clustered columnstore index. If you partition your table into 100 partitions, then your table needs at least 6 billion rows to benefit from a clustered columnstore index (60 distributions *100 partitions* 1 million rows). If your 100-partition table does not have 6 billion rows, either reduce the number of partitions or consider using a heap table instead.

Once your tables have been loaded with some data, follow the below steps to identify and rebuild tables with sub-optimal clustered columnstore indexes.

Rebuild indexes to improve segment quality

Step 1: Identify or create user which uses the right resource class

One quick way to immediately improve segment quality is to rebuild the index. The SQL returned by the above view contains an ALTER INDEX REBUILD statement, which can be used to rebuild your indexes. When rebuilding your indexes, be sure that you allocate enough memory to the session that rebuilds your index. To do this, increase the resource class of a user that has permissions to rebuild the index on this table to the recommended minimum.

Below is an example of how to allocate more memory to a user by increasing their resource class. To work with resource classes, see Resource classes for workload management.

```
EXEC sp_addrolemember 'xlargerc', 'LoadUser';
```

Step 2: Rebuild clustered columnstore indexes with higher resource class user

Sign in as the user from step 1 (`LoadUser`), which is now using a higher resource class, and execute the ALTER INDEX statements. Be sure that this user has ALTER permission to the tables where the index is being rebuilt. These examples show how to rebuild the entire columnstore index or how to rebuild a single partition. On large tables, it is more practical to rebuild indexes a single partition at a time.

Alternatively, instead of rebuilding the index, you could copy the table to a new table using CTAS. Which way is best? For large volumes of data, CTAS is usually faster than ALTER INDEX. For smaller volumes of data, ALTER INDEX is easier to use and won't require you to swap out the table.

```
-- Rebuild the entire clustered index
ALTER INDEX ALL ON [dbo].[DimProduct] REBUILD;
```

```
-- Rebuild a single partition
ALTER INDEX ALL ON [dbo].[FactInternetSales] REBUILD Partition = 5;
```

```
-- Rebuild a single partition with archival compression
ALTER INDEX ALL ON [dbo].[FactInternetSales] REBUILD Partition = 5 WITH (DATA_COMPRESSION =
COLUMNSTORE_ARCHIVE);
```

```
-- Rebuild a single partition with columnstore compression
ALTER INDEX ALL ON [dbo].[FactInternetSales] REBUILD Partition = 5 WITH (DATA_COMPRESSION = COLUMNSTORE);
```

Rebuilding an index in dedicated SQL pool is an offline operation. For more information about rebuilding indexes, see the ALTER INDEX REBUILD section in Columnstore Indexes Defragmentation, and ALTER INDEX.

Step 3: Verify clustered columnstore segment quality has improved

Rerun the query which identified table with poor segment quality and verify segment quality has improved. If segment quality did not improve, it could be that the rows in your table are extra wide. Consider using a higher resource class or DWU when rebuilding your indexes.

Rebuild indexes with CTAS and partition switching

This example uses the CREATE TABLE AS SELECT (CTAS) statement and partition switching to rebuild a table partition.

```
-- Step 1: Select the partition of data and write it out to a new table using CTAS
CREATE TABLE [dbo].[FactInternetSales_20000101_20010101]
    WITH    (   DISTRIBUTION = HASH([ProductKey])
            ,   CLUSTERED COLUMNSTORE INDEX
            ,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                                (20000101,20010101
                                )
                            )
            )
AS
SELECT  *
FROM    [dbo].[FactInternetSales]
WHERE   [OrderDateKey] >= 20000101
AND     [OrderDateKey] <  20010101
;

-- Step 2: Switch IN the rebuilt data with TRUNCATE_TARGET option
ALTER TABLE [dbo].[FactInternetSales_20000101_20010101] SWITCH PARTITION 2 TO  [dbo].[FactInternetSales]
PARTITION 2 WITH (TRUNCATE_TARGET = ON);
```

For more information about re-creating partitions using CTAS, see Using partitions in dedicated SQL pool.

Next steps

For more information about developing tables, see Developing tables.

Using IDENTITY to create surrogate keys using dedicated SQL pool in AzureSynapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

In this article, you'll find recommendations and examples for using the IDENTITY property to create surrogate keys on tables in dedicated SQL pool.

What is a surrogate key

A surrogate key on a table is a column with a unique identifier for each row. The key is not generated from the table data. Data modelers like to create surrogate keys on their tables when they design data warehouse models. You can use the IDENTITY property to achieve this goal simply and effectively without affecting load performance.

> **NOTE**
>
> In Azure Synapse Analytics, the IDENTITY value increases on its own in each distribution and does not overlap with IDENTITY values in other distributions. The IDENTITY value in Synapse is not guaranteed to be unique if the user explicitly inserts a duplicate value with "SET IDENTITY_INSERT ON" or reseeds IDENTITY. For details, see CREATE TABLE (Transact-SQL) IDENTITY (Property).

Creating a table with an IDENTITY column

The IDENTITY property is designed to scale out across all the distributions in the dedicated SQL pool without affecting load performance. Therefore, the implementation of IDENTITY is oriented toward achieving these goals.

You can define a table as having the IDENTITY property when you first create the table by using syntax that is similar to the following statement:

```
CREATE TABLE dbo.T1
(    C1 INT IDENTITY(1,1) NOT NULL
,    C2 INT NULL
)
WITH
(   DISTRIBUTION = HASH(C2)
,   CLUSTERED COLUMNSTORE INDEX
)
;
```

You can then use `INSERT..SELECT` to populate the table.

This remainder of this section highlights the nuances of the implementation to help you understand them more fully.

Allocation of values

The IDENTITY property doesn't guarantee the order in which the surrogate values are allocated due to the distributed architecture of the data warehouse. The IDENTITY property is designed to scale out across all the distributions in the dedicated SQL pool without affecting load performance.

The following example is an illustration:

```
CREATE TABLE dbo.T1
(    C1 INT IDENTITY(1,1)    NOT NULL
,    C2 VARCHAR(30)                NULL
)
WITH
(   DISTRIBUTION = HASH(C2)
,   CLUSTERED COLUMNSTORE INDEX
)
;

INSERT INTO dbo.T1
VALUES (NULL);

INSERT INTO dbo.T1
VALUES (NULL);

SELECT *
FROM dbo.T1;

DBCC PDW_SHOWSPACEUSED('dbo.T1');
```

In the preceding example, two rows landed in distribution 1. The first row has the surrogate value of 1 in column `C1`, and the second row has the surrogate value of 61. Both of these values were generated by the IDENTITY property. However, the allocation of the values is not contiguous. This behavior is by design.

Skewed data

The range of values for the data type are spread evenly across the distributions. If a distributed table suffers from skewed data, then the range of values available to the datatype can be exhausted prematurely. For example, if all the data ends up in a single distribution, then effectively the table has access to only one-sixtieth of the values of the data type. For this reason, the IDENTITY property is limited to `INT` and `BIGINT` data types only.

SELECT..INTO

When an existing IDENTITY column is selected into a new table, the new column inherits the IDENTITY property, unless one of the following conditions is true:

- The SELECT statement contains a join.
- Multiple SELECT statements are joined by using UNION.
- The IDENTITY column is listed more than one time in the SELECT list.
- The IDENTITY column is part of an expression.

If any one of these conditions is true, the column is created NOT NULL instead of inheriting the IDENTITY property.

CREATE TABLE AS SELECT

CREATE TABLE AS SELECT (CTAS) follows the same SQL Server behavior that's documented for SELECT..INTO. However, you can't specify an IDENTITY property in the column definition of the `CREATE TABLE` part of the statement. You also can't use the IDENTITY function in the `SELECT` part of the CTAS. To populate a table, you need to use `CREATE TABLE` to define the table followed by `INSERT..SELECT` to populate it.

Explicitly inserting values into an IDENTITY column

Dedicated SQL pool supports `SET IDENTITY_INSERT <your table> ON|OFF` syntax. You can use this syntax to explicitly insert values into the IDENTITY column.

Many data modelers like to use predefined negative values for certain rows in their dimensions. An example is the -1 or "unknown member" row.

The next script shows how to explicitly add this row by using SET IDENTITY_INSERT:

```
SET IDENTITY_INSERT dbo.T1 ON;

INSERT INTO dbo.T1
(   C1
,   C2
)
VALUES (-1,'UNKNOWN')
;

SET IDENTITY_INSERT dbo.T1 OFF;

SELECT     *
FROM    dbo.T1
;
```

Loading data

The presence of the IDENTITY property has some implications to your data-loading code. This section highlights some basic patterns for loading data into tables by using IDENTITY.

To load data into a table and generate a surrogate key by using IDENTITY, create the table and then use INSERT..SELECT or INSERT..VALUES to perform the load.

The following example highlights the basic pattern:

```
--CREATE TABLE with IDENTITY
CREATE TABLE dbo.T1
(    C1 INT IDENTITY(1,1)
,    C2 VARCHAR(30)
)
WITH
(   DISTRIBUTION = HASH(C2)
,   CLUSTERED COLUMNSTORE INDEX
)
;

--Use INSERT..SELECT to populate the table from an external table
INSERT INTO dbo.T1
(C2)
SELECT     C2
FROM    ext.T1
;

SELECT *
FROM   dbo.T1
;

DBCC PDW_SHOWSPACEUSED('dbo.T1');
```

> **NOTE**
>
> It's not possible to use `CREATE TABLE AS SELECT` currently when loading data into a table with an IDENTITY column.

For more information on loading data, see Designing Extract, Load, and Transform (ELT) for dedicated SQL pool and Loading best practices.

System views

You can use the sys.identity_columns catalog view to identify a column that has the IDENTITY property.

To help you better understand the database schema, this example shows how to integrate sys.identity_column` with other system catalog views:

```
SELECT  sm.name
,       tb.name
,       co.name
,       CASE WHEN ic.column_id IS NOT NULL
             THEN 1
        ELSE 0
        END AS is_identity
FROM        sys.schemas AS sm
JOIN        sys.tables  AS tb           ON  sm.schema_id = tb.schema_id
JOIN        sys.columns AS co           ON  tb.object_id = co.object_id
LEFT JOIN   sys.identity_columns AS ic  ON  co.object_id = ic.object_id
                                        AND co.column_id = ic.column_id
WHERE   sm.name = 'dbo'
AND     tb.name = 'T1'
;
```

Limitations

The IDENTITY property can't be used:

- When the column data type is not INT or BIGINT
- When the column is also the distribution key
- When the table is an external table

The following related functions are not supported in dedicated SQL pool:

- IDENTITY()
- @@IDENTITY
- SCOPE_IDENTITY
- IDENT_CURRENT
- IDENT_INCR
- IDENT_SEED

Common tasks

This section provides some sample code you can use to perform common tasks when you work with IDENTITY columns.

Column C1 is the IDENTITY in all the following tasks.

Find the highest allocated value for a table

Use the `MAX()` function to determine the highest value allocated for a distributed table:

```
SELECT MAX(C1)
FROM dbo.T1
```

Find the seed and increment for the IDENTITY property

You can use the catalog views to discover the identity increment and seed configuration values for a table by using the following query:

```
SELECT  sm.name
,       tb.name
,       co.name
,       ic.seed_value
,       ic.increment_value
FROM        sys.schemas AS sm
JOIN        sys.tables  AS tb           ON  sm.schema_id = tb.schema_id
JOIN        sys.columns AS co           ON  tb.object_id = co.object_id
JOIN        sys.identity_columns AS ic  ON  co.object_id = ic.object_id
                                        AND co.column_id = ic.column_id
WHERE   sm.name = 'dbo'
AND     tb.name = 'T1'
;
```

Next steps

- Table overview
- CREATE TABLE (Transact-SQL) IDENTITY (Property)
- DBCC CHECKIDENT

How to use OPENROWSET using serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 17 minutes to read • Edit Online

The `OPENROWSET(BULK...)` function allows you to access files in Azure Storage. `OPENROWSET` function reads content of a remote data source (for example file) and returns the content as a set of rows. Within the serverless SQL pool resource, the OPENROWSET bulk rowset provider is accessed by calling the OPENROWSET function and specifying the BULK option.

The `OPENROWSET` function can be referenced in the `FROM` clause of a query as if it were a table name `OPENROWSET`. It supports bulk operations through a built-in BULK provider that enables data from a file to be read and returned as a rowset.

> **NOTE**
> The OPENROWSET function is not supported in dedicated SQL pool.

Data source

OPENROWSET function in Synapse SQL reads the content of the file(s) from a data source. The data source is an Azure storage account and it can be explicitly referenced in the `OPENROWSET` function or can be dynamically inferred from URL of the files that you want to read. The `OPENROWSET` function can optionally contain a `DATA_SOURCE` parameter to specify the data source that contains files.

- `OPENROWSET` without `DATA_SOURCE` can be used to directly read the contents of the files from the URL location specified as `BULK` option:

```
SELECT *
FROM OPENROWSET(BULK 'http://<storage account>.dfs.core.windows.net/container/folder/*.parquet',
                FORMAT = 'PARQUET') AS [file]
```

This is a quick and easy way to read the content of the files without pre-configuration. This option enables you to use the basic authentication option to access the storage (Azure AD passthrough for Azure AD logins and SAS token for SQL logins).

- `OPENROWSET` with `DATA_SOURCE` can be used to access files on specified storage account:

```
SELECT *
FROM OPENROWSET(BULK '/folder/*.parquet',
                DATA_SOURCE='storage', --> Root URL is in LOCATION of DATA SOURCE
                FORMAT = 'PARQUET') AS [file]
```

  This option enables you to configure location of the storage account in the data source and specify the authentication method that should be used to access storage.

> **IMPORTANT**
>
> `OPENROWSET` without `DATA_SOURCE` provides quick and easy way to access the storage files but offers limited authentication options. As an example, Azure AD principals can access files only using their Azure AD identity or publicly available files. If you need more powerful authentication options, use `DATA_SOURCE` option and define credential that you want to use to access storage.

Security

A database user must have `ADMINISTER BULK OPERATIONS` permission to use the `OPENROWSET` function.

The storage administrator must also enable a user to access the files by providing valid SAS token or enabling Azure AD principal to access storage files. Learn more about storage access control in this article.

`OPENROWSET` use the following rules to determine how to authenticate to storage:

- In `OPENROWSET` without `DATA_SOURCE` authentication mechanism depends on caller type.
  - Any user can use `OPENROWSET` without `DATA_SOURCE` to read publicly available files on Azure storage.
  - Azure AD logins can access protected files using their own Azure AD identity if Azure storage allows the Azure AD user to access underlying files (for example, if the caller has `Storage Reader` permission on Azure storage).
  - SQL logins can also use `OPENROWSET` without `DATA_SOURCE` to access publicly available files, files protected using SAS token, or Managed Identity of Synapse workspace. You would need to create server-scoped credential to allow access to storage files.
- In `OPENROWSET` with `DATA_SOURCE` authentication mechanism is defined in database scoped credential assigned to the referenced data source. This option enables you to access publicly available storage, or access storage using SAS token, Managed Identity of workspace, or Azure AD identity of caller (if caller is Azure AD principal). If `DATA_SOURCE` references Azure storage that isn't public, you would need to create database-scoped credential and reference it in `DATA SOURCE` to allow access to storage files.

Caller must have `REFERENCES` permission on credential to use it to authenticate to storage.

Syntax

```
--OPENROWSET syntax for reading Parquet or Delta Lake (preview) files
OPENROWSET
( { BULK 'unstructured_data_path' , [DATA_SOURCE = <data source name>, ]
    FORMAT= ['PARQUET' | 'DELTA'] }
)
[WITH ({'column_name' 'column_type' }) ]
[AS] table_alias(column_alias,...n)

--OPENROWSET syntax for reading delimited text files
OPENROWSET
( { BULK 'unstructured_data_path' , [DATA_SOURCE = <data source name>, ]
    FORMAT = 'CSV'
    [ <bulk_options> ]
    [ , <reject_options> ] }
)
WITH ({'column_name' 'column_type' [ 'column_ordinal' | 'json_path'] })
[AS] table_alias(column_alias,...n)

<bulk_options> ::=
[ , FIELDTERMINATOR = 'char' ]
[ , ROWTERMINATOR = 'char' ]
[ , ESCAPECHAR = 'char' ]
[ , FIRSTROW = 'first_row' ]
[ , FIELDQUOTE = 'quote_characters' ]
[ , DATA_COMPRESSION = 'data_compression_method' ]
[ , PARSER_VERSION = 'parser_version' ]
[ , HEADER_ROW = { TRUE | FALSE } ]
[ , DATAFILETYPE = { 'char' | 'widechar' } ]
[ , CODEPAGE = { 'ACP' | 'OEM' | 'RAW' | 'code_page' } ]
[ , ROWSET_OPTIONS = '{"READ_OPTIONS":["ALLOW_INCONSISTENT_READS"]}' ]

<reject_options> ::=
{
    | MAXERRORS = reject_value,
    | ERRORFILE_DATA_SOURCE = <data source name>,
    | ERRORFILE_LOCATION = '/REJECT_Directory'
}
```

Arguments

You have two choices for input files that contain the target data for querying. Valid values are:

- 'CSV' - Includes any delimited text file with row/column separators. Any character can be used as a field separator, such as TSV: FIELDTERMINATOR = tab.
- 'PARQUET' - Binary file in Parquet format
- 'DELTA' - A set of Parquet files organized in Delta Lake (preview) format

'unstructured_data_path'

The unstructured_data_path that establishes a path to the data may be an absolute or relative path:

- Absolute path in the format `\<prefix>://\<storage_account_path>/\<storage_path>` enables a user to directly read the files.
- Relative path in the format `<storage_path>` that must be used with the `DATA_SOURCE` parameter and describes the file pattern within the <storage_account_path> location defined in `EXTERNAL DATA SOURCE`.

Below you'll find the relevant <storage account path> values that will link to your particular external data source.

EXTERNAL DATA SOURCE	PREFIX	STORAGE ACCOUNT PATH
Azure Blob Storage	http[s]	<storage_account>.blob.core.windows.net/path/file
Azure Blob Storage	wasb[s]	<container>@<storage_account>.blob.core.windows.net/path/file
Azure Data Lake Store Gen1	http[s]	<storage_account>.azuredatalakestore.net/webhdfs/v1
Azure Data Lake Store Gen2	http[s]	<storage_account>.dfs.core.windows.net /path/file
Azure Data Lake Store Gen2	abfs[s]	<file_system>@<account_name>.dfs.core.windows.net/path/file

'<storage_path>'

Specifies a path within your storage that points to the folder or file you want to read. If the path points to a container or folder, all files will be read from that particular container or folder. Files in subfolders won't be included.

You can use wildcards to target multiple files or folders. Usage of multiple nonconsecutive wildcards is allowed. Below is an example that reads all *csv* files starting with *population* from all folders starting with */csv/population*:

```
https://sqlondemandstorage.blob.core.windows.net/csv/population*/population*.csv
```

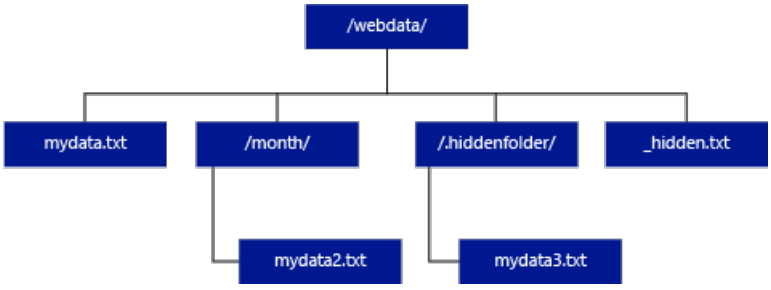
If you specify the unstructured_data_path to be a folder, a serverless SQL pool query will retrieve files from that folder.

You can instruct serverless SQL pool to traverse folders by specifying /* at the end of path as in example:

```
https://sqlondemandstorage.blob.core.windows.net/csv/population/**
```

> **NOTE**
> Unlike Hadoop and PolyBase, serverless SQL pool doesn't return subfolders unless you specify /** at the end of path. Just like Hadoop and PolyBase, it doesn't return files for which the file name begins with an underline (_) or a period (.).

In the example below, if the unstructured_data_path= `https://mystorageaccount.dfs.core.windows.net/webdata/`, a serverless SQL pool query will return rows from mydata.txt. It won't return mydata2.txt and mydata3.txt because they're located in a subfolder.




```
[WITH ({'column_name' 'column_type' [ 'column_ordinal'] }) ]
```

The WITH clause allows you to specify columns that you want to read from files.

- For CSV data files, to read all the columns, provide column names and their data types. If you want a subset of columns, use ordinal numbers to pick the columns from the originating data files by ordinal. Columns will be bound by the ordinal designation. If HEADER_ROW = TRUE is used, then column binding is done by column name instead of ordinal position.

  > **TIP**
  > You can omit WITH clause for CSV files also. Data types will be automatically inferred from file content. You can use HEADER_ROW argument to specify existence of header row in which case column names will be read from header row. For details check automatic schema discovery.

- For Parquet or Delta Lake files, provide column names that match the column names in the originating data files. Columns will be bound by name and is case-sensitive. If the WITH clause is omitted, all columns from Parquet files will be returned.

  > **IMPORTANT**
  > Column names in Parquet and Delta Lake files are case sensitive. If you specify column name with casing different from column name casing in the files, the `NULL` values will be returned for that column.

column_name = Name for the output column. If provided, this name overrides the column name in the source file and column name provided in JSON path if there is one. If json_path is not provided, it will be automatically added as '$.column_name'. Check json_path argument for behavior.

column_type = Data type for the output column. The implicit data type conversion will take place here.

column_ordinal = Ordinal number of the column in the source file(s). This argument is ignored for Parquet files since binding is done by name. The following example would return a second column only from a CSV file:

```
WITH (
    --[country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2 2
    --[year] smallint,
    --[population] bigint
)
```

json_path = JSON path expression to column or nested property. Default path mode is lax.

> **NOTE**
> In strict mode query will fail with error if provided path does not exist. In lax mode query will succeed and JSON path expression will evaluate to NULL.

<bulk_options>

FIELDTERMINATOR='field_terminator'

Specifies the field terminator to be used. The default field terminator is a comma (",").

ROWTERMINATOR='row_terminator'`

Specifies the row terminator to be used. If row terminator is not specified, one of default terminators will be used. Default terminators for PARSER_VERSION = '1.0' are \r\n, \n and \r. Default terminators for PARSER_VERSION = '2.0' are \r\n and \n.

> **NOTE**
> When you use PARSER_VERSION='1.0' and specify \n (newline) as the row terminator, it will be automatically prefixed with a \r (carriage return) character, which results in a row terminator of \r\n.

ESCAPE_CHAR = 'char'

Specifies the character in the file that is used to escape itself and all delimiter values in the file. If the escape character is followed by a value other than itself, or any of the delimiter values, the escape character is dropped when reading the value.

The ESCAPECHAR parameter will be applied regardless of whether the FIELDQUOTE is or isn't enabled. It won't be used to escape the quoting character. The quoting character must be escaped with another quoting character. Quoting character can appear within column value only if value is encapsulated with quoting characters.

FIRSTROW = 'first_row'

Specifies the number of the first row to load. The default is 1 and indicates the first row in the specified data file. The row numbers are determined by counting the row terminators. FIRSTROW is 1-based.

FIELDQUOTE = 'field_quote'

Specifies a character that will be used as the quote character in the CSV file. If not specified, the quote character (") will be used.

DATA_COMPRESSION = 'data_compression_method'

Specifies compression method. Supported in PARSER_VERSION='1.0' only. Following compression method is supported:

- GZIP

PARSER_VERSION = 'parser_version'

Specifies parser version to be used when reading files. Currently supported CSV parser versions are 1.0 and 2.0:

- PARSER_VERSION = '1.0'
- PARSER_VERSION = '2.0'

CSV parser version 1.0 is default and feature rich. Version 2.0 is built for performance and does not support all options and encodings.

CSV parser version 1.0 specifics:

- Following options aren't supported: HEADER_ROW.
- Default terminators are \r\n, \n and \r.
- If you specify \n (newline) as the row terminator, it will be automatically prefixed with a \r (carriage return) character, which results in a row terminator of \r\n.

CSV parser version 2.0 specifics:

- Not all data types are supported.
- Maximum character column length is 8000.
- Maximum row size limit is 8 MB.
- Following options aren't supported: DATA_COMPRESSION.
- Quoted empty string ("") is interpreted as empty string.
- DATEFORMAT SET option is not honored.
- Supported format for DATE data type: YYYY-MM-DD

- Supported format for TIME data type: HH:MM:SS[.fractional seconds]
- Supported format for DATETIME2 data type: YYYY-MM-DD HH:MM:SS[.fractional seconds]
- Default terminators are \r\n and \n.

HEADER_ROW = { TRUE | FALSE }

Specifies whether a CSV file contains header row. Default is `FALSE`. Supported in PARSER_VERSION='2.0'. If TRUE, the column names will be read from the first row according to FIRSTROW argument. If TRUE and schema is specified using WITH, binding of column names will be done by column name, not ordinal positions.

DATAFILETYPE = { 'char' | 'widechar' }

Specifies encoding: `char` is used for UTF8, `widechar` is used for UTF16 files.

CODEPAGE = { 'ACP' | 'OEM' | 'RAW' | 'code_page' }

Specifies the code page of the data in the data file. The default value is 65001 (UTF-8 encoding). See more details about this option here.

ROWSET_OPTIONS = '{"READ_OPTIONS":["ALLOW_INCONSISTENT_READS"]}'

This option will disable the file modification check during the query execution, and read the files that are updated while the query is running. This is useful option when you need to read append-only files that are appended while the query is running. In the appendable files, the existing content is not updated, and only new rows are added. Therefore, the probability of wrong results is minimized compared to the updateable files. This option might enable you to read the frequently appended files without handling the errors. See more information in querying appendable CSV files section.

Reject Options

> **NOTE**
> Rejected rows feature is in Public Preview. Please note that rejected rows feature works for delimited text files and PARSER_VERSION 1.0.

You can specify reject parameters that determine how service will handle *dirty* records it retrieves from the external data source. A data record is considered 'dirty' if actual data types don't match the column definitions of the external table.

When you don't specify or change reject options, service uses default values. Service will use the reject options to determine the number of rows that can be rejected before the actual query fails. The query will return (partial) results until the reject threshold is exceeded. It then fails with the appropriate error message.

MAXERRORS = *reject_value*

Specifies the number of rows that can be rejected before the query fails. MAXERRORS must be an integer between 0 and 2,147,483,647.

ERRORFILE_DATA_SOURCE = *data source*

Specifies data source where rejected rows and the corresponding error file should be written.

ERRORFILE_LOCATION = *Directory Location*

Specifies the directory within the DATA_SOURCE, or ERROR_FILE_DATASOURCE if specified, that the rejected rows and the corresponding error file should be written. If the specified path doesn't exist, service will create one on your behalf. A child directory is created with the name "*rejectedrows". The* '" character ensures that the directory is escaped for other data processing unless explicitly named in the location parameter. Within this

directory, there's a folder created based on the time of load submission in the format YearMonthDay_HourMinuteSecond_StatementID (Ex. 20180330-173205-559EE7D2-196D-400A-806D-3BF5D007F891). You can use statement id to correlate folder with query that generated it. In this folder, two files are written: error.json file and the data file.

error.json file contains json array with encountered errors related to rejected rows. Each element representing error contains following attributes:

ATTRIBUTE	DESCRIPTION
Error	Reason why row is rejected.
Row	Rejected row ordinal number in file.
Column	Rejected column ordinal number.
Value	Rejected column value. If the value is larger than 100 characters, only the first 100 characters will be displayed.
File	Path to file that row belongs to.

Fast delimited text parsing

There are two delimited text parser versions you can use. CSV parser version 1.0 is default and feature rich while parser version 2.0 is built for performance. Performance improvement in parser 2.0 comes from advanced parsing techniques and multi-threading. Difference in speed will be bigger as the file size grows.

Automatic schema discovery

You can easily query both CSV and Parquet files without knowing or specifying schema by omitting WITH clause. Column names and data types will be inferred from files.

Parquet files contain column metadata, which will be read, type mappings can be found in type mappings for Parquet. Check reading Parquet files without specifying schema for samples.

For the CSV files, column names can be read from header row. You can specify whether header row exists using HEADER_ROW argument. If HEADER_ROW = FALSE, generic column names will be used: C1, C2, ... Cn where n is number of columns in file. Data types will be inferred from first 100 data rows. Check reading CSV files without specifying schema for samples.

> **IMPORTANT**
> There are cases when appropriate data type cannot be inferred due to lack of information and larger data type will be used instead. This brings performance overhead and is particularly important for character columns which will be inferred as varchar(8000). For optimal performance, please check inferred data types and use appropriate data types.

Type mapping for Parquet

Parquet and Delta Lake files contain type descriptions for every column. The following table describes how Parquet types are mapped to SQL native types.

PARQUET TYPE	PARQUET LOGICAL TYPE (ANNOTATION)	SQL DATA TYPE
BOOLEAN		bit

PARQUET TYPE	PARQUET LOGICAL TYPE (ANNOTATION)	SQL DATA TYPE
BINARY / BYTE_ARRAY		varbinary
DOUBLE		float
FLOAT		real
INT32		int
INT64		bigint
INT96		datetime2
FIXED_LEN_BYTE_ARRAY		binary
BINARY	UTF8	varchar *(UTF8 collation)
BINARY	STRING	varchar *(UTF8 collation)
BINARY	ENUM	varchar *(UTF8 collation)
FIXED_LEN_BYTE_ARRAY	UUID	uniqueidentifier
BINARY	DECIMAL	decimal
BINARY	JSON	varchar(8000) *(UTF8 collation)
BINARY	BSON	Not supported
FIXED_LEN_BYTE_ARRAY	DECIMAL	decimal
BYTE_ARRAY	INTERVAL	Not supported
INT32	INT(8, true)	smallint
INT32	INT(16, true)	smallint
INT32	INT(32, true)	int
INT32	INT(8, false)	tinyint
INT32	INT(16, false)	int
INT32	INT(32, false)	bigint
INT32	DATE	date
INT32	DECIMAL	decimal
INT32	TIME (MILLIS)	time

PARQUET TYPE	PARQUET LOGICAL TYPE (ANNOTATION)	SQL DATA TYPE
INT64	INT(64, true)	bigint
INT64	INT(64, false)	decimal(20,0)
INT64	DECIMAL	decimal
INT64	TIME (MICROS)	time
INT64	TIME (NANOS)	Not supported
INT64	TIMESTAMP (normalized to utc) (MILLIS / MICROS)	datetime2
INT64	TIMESTAMP (not normalized to utc) (MILLIS / MICROS)	bigint - make sure that you explicitly adjust `bigint` value with the timezone offset before converting it to a datetime value.
INT64	TIMESTAMP (NANOS)	Not supported
Complex type	LIST	varchar(8000), serialized into JSON
Complex type	MAP	varchar(8000), serialized into JSON

Examples

Read CSV files without specifying schema

The following example reads CSV file that contains header row without specifying column names and data types:

```
SELECT
    *
FROM OPENROWSET(
    BULK 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-
19/ecdc_cases/latest/ecdc_cases.csv',
    FORMAT = 'CSV',
    PARSER_VERSION = '2.0',
    HEADER_ROW = TRUE) as [r]
```

The following example reads CSV file that doesn't contain header row without specifying column names and data types:

```
SELECT
    *
FROM OPENROWSET(
    BULK 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-
19/ecdc_cases/latest/ecdc_cases.csv',
    FORMAT = 'CSV',
    PARSER_VERSION = '2.0') as [r]
```

Read Parquet files without specifying schema

The following example returns all columns of the first row from the census data set, in Parquet format, and

without specifying column names and data types:

```
SELECT
    TOP 1 *
FROM
    OPENROWSET(
        BULK
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us_population_county/year=20
*/*.parquet',
        FORMAT='PARQUET'
    ) AS [r]
```

Read Delta Lake files without specifying schema

The following example returns all columns of the first row from the census data set, in Delta Lake format, and without specifying column names and data types:

```
SELECT
    TOP 1 *
FROM
    OPENROWSET(
        BULK
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us_population_county/year=20
*/*.parquet',
        FORMAT='DELTA'
    ) AS [r]
```

Read specific columns from CSV file

The following example returns only two columns with ordinal numbers 1 and 4 from the population*.csv files. Since there's no header row in the files, it starts reading from the first line:

```
SELECT
    *
FROM OPENROWSET(
        BULK 'https://sqlondemandstorage.blob.core.windows.net/csv/population/population*.csv',
        FORMAT = 'CSV',
        FIRSTROW = 1
    )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2 1,
    [population] bigint 4
) AS [r]
```

Read specific columns from Parquet file

The following example returns only two columns of the first row from the census data set, in Parquet format:

```
SELECT
    TOP 1 *
FROM
    OPENROWSET(
        BULK
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us_population_county/year=20
*/*.parquet',
        FORMAT='PARQUET'
    )
WITH (
    [stateName] VARCHAR (50),
    [population] bigint
) AS [r]
```

Specify columns using JSON paths

The following example shows how you can use JSON path expressions in WITH clause and demonstrates difference between strict and lax path modes:

```
SELECT
    TOP 1 *
FROM
    OPENROWSET(
        BULK
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us_population_county/year=20
*/*.parquet',
        FORMAT='PARQUET'
    )
WITH (
    --lax path mode samples
    [stateName] VARCHAR (50), -- this one works as column name casing is valid - it targets the same column
as the next one
    [stateName_explicit_path] VARCHAR (50) '$.stateName', -- this one works as column name casing is valid
    [COUNTYNAME] VARCHAR (50), -- STATEname column will contain NULLs only because of wrong casing - it
targets the same column as the next one
    [countyName_explicit_path] VARCHAR (50) '$.COUNTYNAME', -- STATEname column will contain NULLS only
because of wrong casing and default path mode being lax

    --strict path mode samples
    [population] bigint 'strict $.population' -- this one works as column name casing is valid
    --,[population2] bigint 'strict $.POPULATION' -- this one fails because of wrong casing and strict path
mode
)
AS [r]
```

Specify multiple files/folders in BULK path

The following example shows how you can use multiple file/folder paths in BULK parameter:

```
SELECT
    TOP 10 *
FROM
    OPENROWSET(
        BULK (

'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us_population_county/year=20
00/*.parquet',

'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us_population_county/year=20
10/*.parquet',
        ),
        FORMAT='PARQUET'
    )
AS [r]
```

Next steps

For more samples, see the query data storage quickstart to learn how to use `OPENROWSET` to read CSV, PARQUET, DELTA LAKE, and JSON file formats. Check best practices for achieving optimal performance. You can also learn how to save the results of your query to Azure Storage using CETAS.

Partitioning tables in dedicated SQL pool

8/19/2022 • 11 minutes to read • Edit Online

Recommendations and examples for using table partitions in dedicated SQL pool.

What are table partitions?

Table partitions enable you to divide your data into smaller groups of data. In most cases, table partitions are created on a date column. Partitioning is supported on all dedicated SQL pool table types; including clustered columnstore, clustered index, and heap. Partitioning is also supported on all distribution types, including both hash or round robin distributed.

Partitioning can benefit data maintenance and query performance. Whether it benefits both or just one is dependent on how data is loaded and whether the same column can be used for both purposes, since partitioning can only be done on one column.

Benefits to loads

The primary benefit of partitioning in dedicated SQL pool is to improve the efficiency and performance of loading data by use of partition deletion, switching and merging. In most cases data is partitioned on a date column that is closely tied to the order in which the data is loaded into the SQL pool. One of the greatest benefits of using partitions to maintain data is the avoidance of transaction logging. While simply inserting, updating, or deleting data can be the most straightforward approach, with a little thought and effort, using partitioning during your load process can substantially improve performance.

Partition switching can be used to quickly remove or replace a section of a table. For example, a sales fact table might contain just data for the past 36 months. At the end of every month, the oldest month of sales data is deleted from the table. This data could be deleted by using a delete statement to delete the data for the oldest month.

However, deleting a large amount of data row-by-row with a delete statement can take too much time, as well as create the risk of large transactions that take a long time to rollback if something goes wrong. A more optimal approach is to drop the oldest partition of data. Where deleting the individual rows could take hours, deleting an entire partition could take seconds.

Benefits to queries

Partitioning can also be used to improve query performance. A query that applies a filter to partitioned data can limit the scan to only the qualifying partitions. This method of filtering can avoid a full table scan and only scan a smaller subset of data. With the introduction of clustered columnstore indexes, the predicate elimination performance benefits are less beneficial, but in some cases there can be a benefit to queries.

For example, if the sales fact table is partitioned into 36 months using the sales date field, then queries that filter on the sale date can skip searching in partitions that don't match the filter.

Partition sizing

While partitioning can be used to improve performance some scenarios, creating a table with **too many** partitions can hurt performance under some circumstances. These concerns are especially true for clustered columnstore tables.

For partitioning to be helpful, it is important to understand when to use partitioning and the number of partitions to create. There is no hard fast rule as to how many partitions are too many, it depends on your data

and how many partitions you are loading simultaneously. A successful partitioning scheme usually has tens to hundreds of partitions, not thousands.

When creating partitions on **clustered columnstore** tables, it is important to consider how many rows belong to each partition. For optimal compression and performance of clustered columnstore tables, a minimum of 1 million rows per distribution and partition is needed. Before partitions are created, dedicated SQL pool already divides each table into 60 distributions.

Any partitioning added to a table is in addition to the distributions created behind the scenes. Using this example, if the sales fact table contained 36 monthly partitions, and given that a dedicated SQL pool has 60 distributions, then the sales fact table should contain 60 million rows per month, or 2.1 billion rows when all months are populated. If a table contains fewer than the recommended minimum number of rows per partition, consider using fewer partitions in order to increase the number of rows per partition.

For more information, see the Indexing article, which includes queries that can assess the quality of cluster columnstore indexes.

Syntax differences from SQL Server

Dedicated SQL pool introduces a way to define partitions that is simpler than SQL Server. Partitioning functions and schemes are not used in dedicated SQL pool as they are in SQL Server. Instead, all you need to do is identify partitioned column and the boundary points.

While the syntax of partitioning may be slightly different from SQL Server, the basic concepts are the same. SQL Server and dedicated SQL pool support one partition column per table, which can be ranged partition. To learn more about partitioning, see Partitioned Tables and Indexes.

The following example uses the CREATE TABLE statement to partition the `FactInternetSales` table on the `OrderDateKey` column:

```
CREATE TABLE [dbo].[FactInternetSales]
(
    [ProductKey]            int          NOT NULL
,   [OrderDateKey]          int          NOT NULL
,   [CustomerKey]           int          NOT NULL
,   [PromotionKey]          int          NOT NULL
,   [SalesOrderNumber]      nvarchar(20) NOT NULL
,   [OrderQuantity]         smallint     NOT NULL
,   [UnitPrice]             money        NOT NULL
,   [SalesAmount]           money        NOT NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = HASH([ProductKey])
,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                    (20000101,20010101,20020101
                    ,20030101,20040101,20050101
                    )
                )
);
```

Migrate partitions from SQL Server

To migrate SQL Server partition definitions to dedicated SQL pool simply:

- Eliminate the SQL Server partition scheme.
- Add the partition function definition to your CREATE TABLE.

If you are migrating a partitioned table from a SQL Server instance, the following SQL can help you to figure out

the number of rows in each partition. Keep in mind that if the same partitioning granularity is used in dedicated SQL pool, the number of rows per partition decreases by a factor of 60.

```
-- Partition information for a SQL Server Database
SELECT      s.[name]                        AS      [schema_name]
,           t.[name]                        AS      [table_name]
,           i.[name]                        AS      [index_name]
,           p.[partition_number]            AS      [partition_number]
,           SUM(a.[used_pages]*8.0)         AS      [partition_size_kb]
,           SUM(a.[used_pages]*8.0)/1024    AS      [partition_size_mb]
,           SUM(a.[used_pages]*8.0)/1048576 AS      [partition_size_gb]
,           p.[rows]                        AS      [partition_row_count]
,           rv.[value]                      AS      [partition_boundary_value]
,           p.[data_compression_desc]       AS      [partition_compression_desc]
FROM        sys.schemas s
JOIN        sys.tables t                    ON      t.[schema_id]         = s.[schema_id]
JOIN        sys.partitions p                ON      p.[object_id]         = t.[object_id]
JOIN        sys.allocation_units a          ON      a.[container_id]      = p.[partition_id]
JOIN        sys.indexes i                   ON      i.[object_id]         = p.[object_id]
                                            AND     i.[index_id]          = p.[index_id]
JOIN        sys.data_spaces ds              ON      ds.[data_space_id]    = i.[data_space_id]
LEFT JOIN   sys.partition_schemes ps        ON      ps.[data_space_id]    = ds.[data_space_id]
LEFT JOIN   sys.partition_functions pf      ON      pf.[function_id]      = ps.[function_id]
LEFT JOIN   sys.partition_range_values rv   ON      rv.[function_id]      = pf.[function_id]
                                            AND     rv.[boundary_id]      = p.[partition_number]
WHERE       p.[index_id] <=1
GROUP BY    s.[name]
,           t.[name]
,           i.[name]
,           p.[partition_number]
,           p.[rows]
,           rv.[value]
,           p.[data_compression_desc];
```

Partition switching

Dedicated SQL pool supports partition splitting, merging, and switching. Each of these functions is executed using the ALTER TABLE statement.

To switch partitions between two tables, you must ensure that the partitions align on their respective boundaries and that the table definitions match. As check constraints are not available to enforce the range of values in a table, the source table must contain the same partition boundaries as the target table. If the partition boundaries are not the same, then the partition switch will fail as the partition metadata will not be synchronized.

A partition split requires the respective partition (not necessarily the whole table) to be empty if the table has a clustered columnstore index (CCI). Other partitions in the same table can contain data. A partition that contains data cannot be split, it will result in error:

```
ALTER PARTITION statement failed because the partition is not empty. Only empty partitions can be split in
when a columnstore index exists on the table. Consider disabling the columnstore index before issuing the
ALTER PARTITION statement, then rebuilding the columnstore index after ALTER PARTITION is complete.
```

As a workaround to split a partition containing data, see How to split a partition that contains data.

How to split a partition that contains data

The most efficient method to split a partition that already contains data is to use a `CTAS` statement. If the partitioned table is a clustered columnstore, then the table partition must be empty before it can be split.

The following example creates a partitioned columnstore table. It inserts one row into each partition:

```
CREATE TABLE [dbo].[FactInternetSales]
(
        [ProductKey]            int          NOT NULL
    ,   [OrderDateKey]          int          NOT NULL
    ,   [CustomerKey]           int          NOT NULL
    ,   [PromotionKey]          int          NOT NULL
    ,   [SalesOrderNumber]      nvarchar(20) NOT NULL
    ,   [OrderQuantity]         smallint     NOT NULL
    ,   [UnitPrice]             money        NOT NULL
    ,   [SalesAmount]           money        NOT NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = HASH([ProductKey])
,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                    (20000101
                    )
                )
);

INSERT INTO dbo.FactInternetSales
VALUES (1,19990101,1,1,1,1,1,1);

INSERT INTO dbo.FactInternetSales
VALUES (1,20000101,1,1,1,1,1,1);
```

The following query finds the row count by using the `sys.partitions` catalog view:

```
SELECT  QUOTENAME(s.[name])+'.'+QUOTENAME(t.[name]) as Table_name
,       i.[name] as Index_name
,       p.partition_number as Partition_nmbr
,       p.[rows] as Row_count
,       p.[data_compression_desc] as Data_Compression_desc
FROM    sys.partitions p
JOIN    sys.tables     t    ON    p.[object_id]   = t.[object_id]
JOIN    sys.schemas    s    ON    t.[schema_id]   = s.[schema_id]
JOIN    sys.indexes    i    ON    p.[object_id]   = i.[object_Id]
                            AND   p.[index_Id]    = i.[index_Id]
WHERE t.[name] = 'FactInternetSales';
```

The following split command receives an error message:

```
ALTER TABLE FactInternetSales SPLIT RANGE (20010101);
```

```
Msg 35346, Level 15, State 1, Line 44
SPLIT clause of ALTER PARTITION statement failed because the partition is not empty. Only empty partitions
can be split in when a columnstore index exists on the table. Consider disabling the columnstore index
before issuing the ALTER PARTITION statement, then rebuilding the columnstore index after ALTER PARTITION is
complete.
```

However, you can use `CTAS` to create a new table to hold the data.

```
CREATE TABLE dbo.FactInternetSales_20000101
    WITH    (   DISTRIBUTION = HASH(ProductKey)
            ,   CLUSTERED COLUMNSTORE INDEX
            ,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                                (20000101
                                )
                            )
)
AS
SELECT *
FROM    FactInternetSales
WHERE   1=2;
```

As the partition boundaries are aligned, a switch is permitted. This will leave the source table with an empty partition that you can subsequently split.

```
ALTER TABLE FactInternetSales SWITCH PARTITION 2 TO FactInternetSales_20000101 PARTITION 2;

ALTER TABLE FactInternetSales SPLIT RANGE (20010101);
```

All that is left is to align the data to the new partition boundaries using `CTAS`, and then switch the data back into the main table.

```
CREATE TABLE [dbo].[FactInternetSales_20000101_20010101]
    WITH    (   DISTRIBUTION = HASH([ProductKey])
            ,   CLUSTERED COLUMNSTORE INDEX
            ,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                                (20000101,20010101
                                )
                            )
            )
AS
SELECT  *
FROM    [dbo].[FactInternetSales_20000101]
WHERE   [OrderDateKey] >= 20000101
AND     [OrderDateKey] <  20010101;

ALTER TABLE dbo.FactInternetSales_20000101_20010101 SWITCH PARTITION 2 TO dbo.FactInternetSales PARTITION 2;
```

Once you have completed the movement of the data, it is a good idea to refresh the statistics on the target table. Updating statistics ensures the statistics accurately reflect the new distribution of the data in their respective partitions.

```
UPDATE STATISTICS [dbo].[FactInternetSales];
```

Finally, in the case of a one-time partition switch to move data, you could drop the tables created for the partition switch, `FactInternetSales_20000101_20010101` and `FactInternetSales_20000101`. Alternatively, you may want to keep empty tables for regular, automated partition switches.

Load new data into partitions that contain data in one step

Loading data into partitions with partition switching is a convenient way to stage new data in a table that is not visible to users. It can be challenging on busy systems to deal with the locking contention associated with partition switching.

To clear out the existing data in a partition, an `ALTER TABLE` used to be required to switch out the data. Then another `ALTER TABLE` was required to switch in the new data.

In dedicated SQL pool, the `TRUNCATE_TARGET` option is supported in the `ALTER TABLE` command. With `TRUNCATE_TARGET` the `ALTER TABLE` command overwrites existing data in the partition with new data. Below is an example that uses `CTAS` to create a new table with the existing data, inserts new data, then switches all the data back into the target table, overwriting the existing data.

```
CREATE TABLE [dbo].[FactInternetSales_NewSales]
    WITH    (   DISTRIBUTION = HASH([ProductKey])
            ,   CLUSTERED COLUMNSTORE INDEX
            ,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                                (20000101,20010101
                                )
                            )
            )
AS
SELECT  *
FROM    [dbo].[FactInternetSales]
WHERE   [OrderDateKey] >= 20000101
AND     [OrderDateKey] <  20010101
;

INSERT INTO dbo.FactInternetSales_NewSales
VALUES (1,20000101,2,2,2,2,2,2);

ALTER TABLE dbo.FactInternetSales_NewSales SWITCH PARTITION 2 TO dbo.FactInternetSales PARTITION 2 WITH
(TRUNCATE_TARGET = ON);
```

Table partitioning source control

> **NOTE**
> If your source control tool is not configured to ignore partition schemas, altering a table's schema to update partitions may cause a table to be dropped and recreated as part of the deployment, which may be unfeasible. A custom solution to implement such a change, as described below, may be necessary. Check that your continuous integration/continuous deployment (CI/CD) tool allows for this. In SQL Server Data Tools (SSDT), look for the Advanced Publish Settings "Ignore partition schemes" to avoid a generated script that cause a table to be dropped and recreated.

This example is useful when updating partition schemas of an empty table. To continuously deploy partition changes on a table with data, follow the steps in How to split a partition that contains data alongside deployment to temporarily move data out of each partition before applying the partition SPLIT RANGE. This is necessary since the CI/CD tool isn't aware of which partitions have data.

To avoid your table definition from **rusting** in your source control system, you may want to consider the following approach:

1. Create the table as a partitioned table but with no partition values

```
CREATE TABLE [dbo].[FactInternetSales]
(
    [ProductKey]            int          NOT NULL
,   [OrderDateKey]          int          NOT NULL
,   [CustomerKey]           int          NOT NULL
,   [PromotionKey]          int          NOT NULL
,   [SalesOrderNumber]      nvarchar(20) NOT NULL
,   [OrderQuantity]         smallint     NOT NULL
,   [UnitPrice]             money        NOT NULL
,   [SalesAmount]           money        NOT NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = HASH([ProductKey])
,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES () )
);
```

2. `SPLIT` the table as part of the deployment process:

```
 -- Create a table containing the partition boundaries

CREATE TABLE #partitions
WITH
(
    LOCATION = USER_DB
,   DISTRIBUTION = HASH(ptn_no)
)
AS
SELECT  ptn_no
,       ROW_NUMBER() OVER (ORDER BY (ptn_no)) as seq_no
FROM    (
    SELECT CAST(20000101 AS INT) ptn_no
    UNION ALL
    SELECT CAST(20010101 AS INT)
    UNION ALL
    SELECT CAST(20020101 AS INT)
    UNION ALL
    SELECT CAST(20030101 AS INT)
    UNION ALL
    SELECT CAST(20040101 AS INT)
) a;

 -- Iterate over the partition boundaries and split the table

DECLARE @c INT = (SELECT COUNT(*) FROM #partitions)
,       @i INT = 1                              --iterator for while loop
,       @q NVARCHAR(4000)                       --query
,       @p NVARCHAR(20)     = N''               --partition_number
,       @s NVARCHAR(128)    = N'dbo'            --schema
,       @t NVARCHAR(128)    = N'FactInternetSales' --table;

WHILE @i <= @c
BEGIN
    SET @p = (SELECT ptn_no FROM #partitions WHERE seq_no = @i);
    SET @q = (SELECT N'ALTER TABLE '+@s+N'.'+@t+N' SPLIT RANGE ('+@p+N');');

    -- PRINT @q;
    EXECUTE sp_executesql @q;
    SET @i+=1;
END

 -- Code clean-up

DROP TABLE #partitions;
```

With this approach, the code in source control remains static and the partitioning boundary values are allowed to be dynamic; evolving with the SQL pool over time.

Next steps

For more information about developing tables, see the articles on Table Overview.

Design guidance for using replicated tables in Synapse SQL pool

8/19/2022 • 7 minutes to read • Edit Online

This article gives recommendations for designing replicated tables in your Synapse SQL pool schema. Use these recommendations to improve query performance by reducing data movement and query complexity.

Prerequisites

This article assumes you are familiar with data distribution and data movement concepts in SQL pool. For more information, see the architecture article.

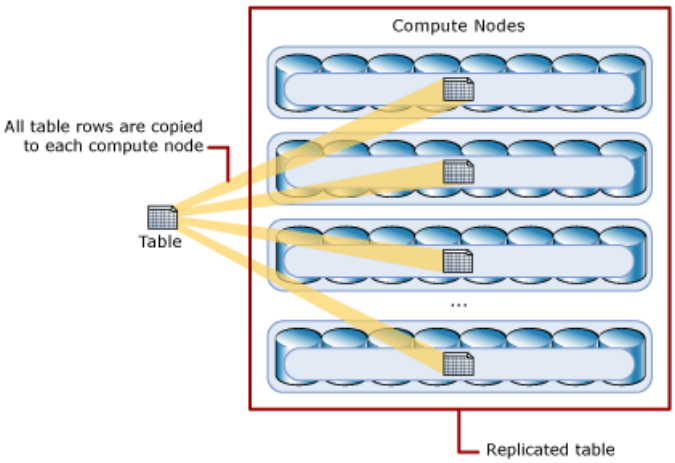
As part of table design, understand as much as possible about your data and how the data is queried. For example, consider these questions:

- How large is the table?
- How often is the table refreshed?
- Do I have fact and dimension tables in a SQL pool?

What is a replicated table?

A replicated table has a full copy of the table accessible on each Compute node. Replicating a table removes the need to transfer data among Compute nodes before a join or aggregation. Since the table has multiple copies, replicated tables work best when the table size is less than 2 GB compressed. 2 GB is not a hard limit. If the data is static and does not change, you can replicate larger tables.

The following diagram shows a replicated table that is accessible on each Compute node. In SQL pool, the replicated table is fully copied to a distribution database on each compute node.



Replicated tables work well for dimension tables in a star schema. Dimension tables are typically joined to fact tables, which are distributed differently than the dimension table. Dimensions are usually of a size that makes it feasible to store and maintain multiple copies. Dimensions store descriptive data that changes slowly, such as customer name and address, and product details. The slowly changing nature of the data leads to less maintenance of the replicated table.

Consider using a replicated table when:

- The table size on disk is less than 2 GB, regardless of the number of rows. To find the size of a table, you can

use the DBCC PDW_SHOWSPACEUSED command: `DBCC PDW_SHOWSPACEUSED('ReplTableCandidate')`.

- The table is used in joins that would otherwise require data movement. When joining tables that are not distributed on the same column, such as a hash-distributed table to a round-robin table, data movement is required to complete the query. If one of the tables is small, consider a replicated table. We recommend using replicated tables instead of round-robin tables in most cases. To view data movement operations in query plans, use sys.dm_pdw_request_steps. The BroadcastMoveOperation is the typical data movement operation that can be eliminated by using a replicated table.

Replicated tables may not yield the best query performance when:

- The table has frequent insert, update, and delete operations. The data manipulation language (DML) operations require a rebuild of the replicated table. Rebuilding frequently can cause slower performance.
- The SQL pool is scaled frequently. Scaling a SQL pool changes the number of Compute nodes, which incurs rebuilding the replicated table.
- The table has a large number of columns, but data operations typically access only a small number of columns. In this scenario, instead of replicating the entire table, it might be more effective to distribute the table, and then create an index on the frequently accessed columns. When a query requires data movement, SQL pool only moves data for the requested columns.

Use replicated tables with simple query predicates

Before you choose to distribute or replicate a table, think about the types of queries you plan to run against the table. Whenever possible,

- Use replicated tables for queries with simple query predicates, such as equality or inequality.
- Use distributed tables for queries with complex query predicates, such as LIKE or NOT LIKE.

CPU-intensive queries perform best when the work is distributed across all of the Compute nodes. For example, queries that run computations on each row of a table perform better on distributed tables than replicated tables. Since a replicated table is stored in full on each Compute node, a CPU-intensive query against a replicated table runs against the entire table on every Compute node. The extra computation can slow query performance.

For example, this query has a complex predicate. It runs faster when the data is in a distributed table instead of a replicated table. In this example, the data can be round-robin distributed.

```
SELECT EnglishProductName
FROM DimProduct
WHERE EnglishDescription LIKE '%frame%comfortable%';
```

Convert existing round-robin tables to replicated tables

If you already have round-robin tables, we recommend converting them to replicated tables if they meet the criteria outlined in this article. Replicated tables improve performance over round-robin tables because they eliminate the need for data movement. A round-robin table always requires data movement for joins.

This example uses CTAS to change the `DimSalesTerritory` table to a replicated table. This example works regardless of whether `DimSalesTerritory` is hash-distributed or round-robin.

```
CREATE TABLE [dbo].[DimSalesTerritory_REPLICATE]
WITH
  (
    HEAP,
    DISTRIBUTION = REPLICATE
  )
AS SELECT * FROM [dbo].[DimSalesTerritory]
OPTION  (LABEL  = 'CTAS : DimSalesTerritory_REPLICATE')

-- Switch table names
RENAME OBJECT [dbo].[DimSalesTerritory] to [DimSalesTerritory_old];
RENAME OBJECT [dbo].[DimSalesTerritory_REPLICATE] TO [DimSalesTerritory];

DROP TABLE [dbo].[DimSalesTerritory_old];
```

Query performance example for round-robin versus replicated

A replicated table does not require any data movement for joins because the entire table is already present on each Compute node. If the dimension tables are round-robin distributed, a join copies the dimension table in full to each Compute node. To move the data, the query plan contains an operation called BroadcastMoveOperation. This type of data movement operation slows query performance and is eliminated by using replicated tables. To view query plan steps, use the sys.dm_pdw_request_steps system catalog view.

For example, in following query against the `AdventureWorks` schema, the `FactInternetSales` table is hash-distributed. The `DimDate` and `DimSalesTerritory` tables are smaller dimension tables. This query returns the total sales in North America for fiscal year 2004:

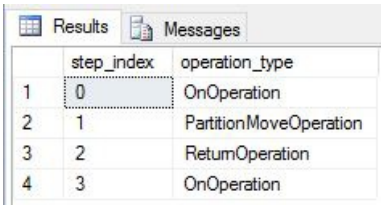
```
SELECT [TotalSalesAmount] = SUM(SalesAmount)
FROM dbo.FactInternetSales s
INNER JOIN dbo.DimDate d
  ON d.DateKey = s.OrderDateKey
INNER JOIN dbo.DimSalesTerritory t
  ON t.SalesTerritoryKey = s.SalesTerritoryKey
WHERE d.FiscalYear = 2004
  AND t.SalesTerritoryGroup = 'North America'
```

We re-created `DimDate` and `DimSalesTerritory` as round-robin tables. As a result, the query showed the following query plan, which has multiple broadcast move operations:

Results | Messages

	step_index	operation_type
1	0	RandomIDOperation
2	1	OnOperation
3	2	BroadcastMoveOperation
4	3	RandomIDOperation
5	4	OnOperation
6	5	BroadcastMoveOperation
7	6	OnOperation
8	7	PartitionMoveOperation
9	8	OnOperation
10	9	OnOperation
11	10	ReturnOperation
12	11	OnOperation

We re-created `DimDate` and `DimSalesTerritory` as replicated tables, and ran the query again. The resulting query plan is much shorter and does not have any broadcast moves.



Results | Messages

	step_index	operation_type
1	0	OnOperation
2	1	PartitionMoveOperation
3	2	ReturnOperation
4	3	OnOperation

Performance considerations for modifying replicated tables

SQL pool implements a replicated table by maintaining a master version of the table. It copies the master version to the first distribution database on each Compute node. When there is a change, the master version is updated first, then the tables on each Compute node are rebuilt. A rebuild of a replicated table includes copying the table to each Compute node and then building the indexes. For example, a replicated table on a DW2000c has five copies of the data. A master copy and a full copy on each Compute node. All data is stored in distribution databases. SQL pool uses this model to support faster data modification statements and flexible scaling operations.

Asynchronous rebuilds are triggered by the first query against the replicated table after:

- Data is loaded or modified
- The Synapse SQL instance is scaled to a different level
- Table definition is updated

Rebuilds are not required after:

- Pause operation
- Resume operation

The rebuild does not happen immediately after data is modified. Instead, the rebuild is triggered the first time a query selects from the table. The query that triggered the rebuild reads immediately from the master version of the table while the data is asynchronously copied to each Compute node. Until the data copy is complete, subsequent queries will continue to use the master version of the table. If any activity happens against the replicated table that forces another rebuild, the data copy is invalidated and the next select statement will trigger data to be copied again.

Use indexes conservatively

Standard indexing practices apply to replicated tables. SQL pool rebuilds each replicated table index as part of the rebuild. Only use indexes when the performance gain outweighs the cost of rebuilding the indexes.

Batch data load

When loading data into replicated tables, try to minimize rebuilds by batching loads together. Perform all the batched loads before running select statements.

For example, this load pattern loads data from four sources and invokes four rebuilds.

- Load from source 1.
- Select statement triggers rebuild 1.
- Load from source 2.
- Select statement triggers rebuild 2.
- Load from source 3.
- Select statement triggers rebuild 3.
- Load from source 4.
- Select statement triggers rebuild 4.

For example, this load pattern loads data from four sources, but only invokes one rebuild.

- Load from source 1.
- Load from source 2.
- Load from source 3.
- Load from source 4.
- Select statement triggers rebuild.

Rebuild a replicated table after a batch load

To ensure consistent query execution times, consider forcing the build of the replicated tables after a batch load. Otherwise, the first query will still use data movement to complete the query.

This query uses the sys.pdw_replicated_table_cache_state DMV to list the replicated tables that have been modified, but not rebuilt.

```
SELECT SchemaName = SCHEMA_NAME(t.schema_id)
 , [ReplicatedTable] = t.[name]
 , [RebuildStatement] = 'SELECT TOP 1 * FROM ' + '[' + SCHEMA_NAME(t.schema_id) + '].[' + t.[name] +']'
FROM sys.tables t
JOIN sys.pdw_replicated_table_cache_state c
  ON c.object_id = t.object_id
JOIN sys.pdw_table_distribution_properties p
  ON p.object_id = t.object_id
WHERE c.[state] = 'NotReady'
AND p.[distribution_policy_desc] = 'REPLICATE'
```

To trigger a rebuild, run the following statement on each table in the preceding output.

```
SELECT TOP 1 * FROM [ReplicatedTable]
```

Next steps

To create a replicated table, use one of these statements:

- CREATE TABLE (SQL pool)
- CREATE TABLE AS SELECT (SQL pool)

For an overview of distributed tables, see distributed tables.

Statistics in Synapse SQL

8/19/2022 • 23 minutes to read • Edit Online

Provided in this article are recommendations and examples for creating and updating query-optimization statistics using the Synapse SQL resources: dedicated SQL pool and serverless SQL pool.

Statistics in dedicated SQL pool

Why use statistics

The more dedicated SQL pool knows about your data, the faster it can execute queries. After loading data into a dedicated SQL pool, collecting statistics on your data is one of the most important things you can do for query optimization.

The dedicated SQL pool query optimizer is a cost-based optimizer. It compares the cost of various query plans, and then chooses the plan with the lowest cost. In most cases, it chooses the plan that will execute the fastest.

For example, if the optimizer estimates that the date your query is filtering on will return one row, it will choose one plan. If it estimates that the selected date will return 1 million rows, it will return a different plan.

Automatic creation of statistics

The dedicated SQL pool engine will analyze incoming user queries for missing statistics when the database AUTO_CREATE_STATISTICS option is set to `ON`. If statistics are missing, the query optimizer creates statistics on individual columns in the query predicate or join condition.

This function is used to improve cardinality estimates for the query plan.

> **IMPORTANT**
> Automatic creation of statistics is currently turned on by default.

You can check if your data warehouse has AUTO_CREATE_STATISTICS configured by running the following command:

```
SELECT name, is_auto_create_stats_on
FROM sys.databases
```

If your data warehouse doesn't have AUTO_CREATE_STATISTICS enabled, we recommend you enable this property by running the following command:

```
ALTER DATABASE <yourdatawarehousename>
SET AUTO_CREATE_STATISTICS ON
```

These statements will trigger the automatic creation of statistics:

- SELECT
- INSERT-SELECT
- CTAS
- UPDATE
- DELETE

- EXPLAIN when containing a join or the presence of a predicate is detected

> **NOTE**
> The automatic creation of statistics is not generated on temporary or external tables.

Automatic creation of statistics is done synchronously. So, you may incur slightly degraded query performance if your columns are missing statistics. The time to create statistics for a single column depends on the size of the table.

To avoid measurable performance degradation, you should ensure stats have been created first by executing the benchmark workload before profiling the system.

> **NOTE**
> The creation of stats is logged in sys.dm_pdw_exec_requests under a different user context.

When automatic statistics are created, they'll take the form: *WA_Sys*<8 digit column id in Hex>_<8 digit table id in Hex>. You can view already created stats by running the DBCC SHOW_STATISTICS command:

```
DBCC SHOW_STATISTICS (<table_name>, <target>)
```

The table_name is the name of the table that contains the statistics to display, which can't be an external table. The target is the name of the target index, statistics, or column for which to display statistics information.

Update statistics

One best practice is to update statistics on date columns each day as new dates are added. Each time new rows are loaded into the data warehouse, new load dates or transaction dates are added. These additions change the data distribution and make the statistics out of date.

Statistics on a country or region column in a customer table might never need to be updated because the distribution of values doesn't usually change. Assuming the distribution is constant between customers, adding new rows to the table variation isn't going to change the data distribution.

However, when your data warehouse only contains one country or region and you bring in data from a new country or region, then you need to update statistics on the country or region column.

The following are recommendations for updating statistics:

TYPE	RECOMMENDATION
Frequency of stats updates	Conservative: Daily After loading or transforming your data
Sampling	Less than 1 billion rows, use default sampling (20 percent). With more than 1 billion rows, use sampling of two percent.

Determine last statistics update

One of the first questions to ask when you're troubleshooting a query is, **"Are the statistics up to date?"**

This question isn't one that can be answered by the age of the data. An up-to-date statistics object might be old if there's been no material change to the underlying data. When the number of rows has changed substantially, or a material change in the distribution of values for a column occurs, *then* it's time to update statistics.

There isn't a dynamic management view available to determine if data within the table has changed since the last time statistics were updated. Knowing the age of your statistics can provide you with part of the picture.

You can use the following query to determine the last time your statistics were updated on each table.

> **NOTE**
> If there is a material change in the distribution of values for a column, you should update statistics regardless of the last time they were updated.

```
SELECT
    sm.[name] AS [schema_name],
    tb.[name] AS [table_name],
    co.[name] AS [stats_column_name],
    st.[name] AS [stats_name],
    STATS_DATE(st.[object_id],st.[stats_id]) AS [stats_last_updated_date]
FROM
    sys.objects ob
    JOIN sys.stats st
        ON  ob.[object_id] = st.[object_id]
    JOIN sys.stats_columns sc
        ON  st.[stats_id] = sc.[stats_id]
        AND st.[object_id] = sc.[object_id]
    JOIN sys.columns co
        ON  sc.[column_id] = co.[column_id]
        AND sc.[object_id] = co.[object_id]
    JOIN sys.types  ty
        ON  co.[user_type_id] = ty.[user_type_id]
    JOIN sys.tables tb
        ON  co.[object_id] = tb.[object_id]
    JOIN sys.schemas sm
        ON  tb.[schema_id] = sm.[schema_id]
WHERE
    st.[user_created] = 1;
```

Date columns in a data warehouse, for example, usually need frequent statistics updates. Each time new rows are loaded into the data warehouse, new load dates or transaction dates are added. These additions change the data distribution and make the statistics out of date.

Statistics on a gender column in a customer table might never need to be updated. Assuming the distribution is constant between customers, adding new rows to the table variation isn't going to change the data distribution.

But, if your data warehouse contains only one gender and a new requirement results in multiple genders, then you need to update statistics on the gender column.

For further information, review the Statistics article.

Implement statistics management

It's often a good idea to extend your data-loading process to ensure that statistics are updated at the end of the load. The data load is when tables most frequently change their size, distribution of values, or both. As such, the load process is a logical place to implement some management processes.

The following guiding principles are provided for updating your statistics during the load process:

- Ensure that each loaded table has at least one statistics object updated. This process updates the table size (row count and page count) information as part of the statistics update.
- Focus on columns participating in JOIN, GROUP BY, ORDER BY, and DISTINCT clauses.
- Consider updating "ascending key" columns such as transaction dates more frequently because these values won't be included in the statistics histogram.

- Consider updating static distribution columns less frequently.
- Remember, each statistic object is updated in sequence. Simply implementing `UPDATE STATISTICS <TABLE_NAME>` isn't always ideal, especially for wide tables with lots of statistics objects.

For more information, see Cardinality Estimation.

Examples: Create statistics

These examples show how to use various options for creating statistics. The options that you use for each column depend on the characteristics of your data and how the column will be used in queries.

Create single-column statistics with default options

To create statistics on a column, provide a name for the statistics object and the name of the column. This syntax uses all of the default options. By default, dedicated SQL pool samples **20 percent** of the table when it creates statistics.

```
CREATE STATISTICS [statistics_name]
    ON [schema_name].[table_name]([column_name]);
```

For example:

```
CREATE STATISTICS col1_stats
    ON dbo.table1 (col1);
```

Create single-column statistics by examining every row

The default sampling rate of 20 percent is sufficient for most situations. However, you can adjust the sampling rate. To sample the full table, use this syntax:

```
CREATE STATISTICS [statistics_name]
    ON [schema_name].[table_name]([column_name])
    WITH FULLSCAN;
```

For example:

```
CREATE STATISTICS col1_stats
    ON dbo.table1 (col1)
    WITH FULLSCAN;
```

Create single-column statistics by specifying the sample size

Another option you have is to specify the sample size as a percent:

```
CREATE STATISTICS col1_stats
    ON dbo.table1 (col1)
    WITH SAMPLE 50 PERCENT;
```

Create single-column statistics on only some of the rows

You can also create statistics on a portion of the rows in your table, which is called a filtered statistic.

For example, you can use filtered statistics when you plan to query a specific partition of a large partitioned table. By creating statistics on only the partition values, the accuracy of the statistics will improve. You'll also experience an improvement in query performance.

This example creates statistics on a range of values. The values can easily be defined to match the range of values in a partition.

```
CREATE STATISTICS stats_col1
    ON table1(col1)
    WHERE col1 > '2000101' AND col1 < '20001231';
```

> **NOTE**
>
> For the query optimizer to consider using filtered statistics when it chooses the distributed query plan, the query must fit inside the definition of the statistics object. Using the previous example, the query's WHERE clause needs to specify col1 values between 2000101 and 20001231.

Create single-column statistics with all the options

You can also combine the options together. The following example creates a filtered statistics object with a custom sample size:

```
CREATE STATISTICS stats_col1
    ON table1 (col1)
    WHERE col1 > '2000101' AND col1 < '20001231'
    WITH SAMPLE 50 PERCENT;
```

For the full reference, see CREATE STATISTICS.

Create multi-column statistics

To create a multi-column statistics object, use the previous examples, but specify more columns.

> **NOTE**
>
> The histogram, which is used to estimate the number of rows in the query result, is only available for the first column listed in the statistics object definition.

In this example, the histogram is on *product_category*. Cross-column statistics are calculated on *product_category* and *product_sub_category*.

```
CREATE STATISTICS stats_2cols
    ON table1 (product_category, product_sub_category)
    WHERE product_category > '2000101' AND product_category < '20001231'
    WITH SAMPLE 50 PERCENT;
```

Because a correlation exists between *product_category* and *product_sub_category*, a multi-column statistics object can be useful if these columns are accessed at the same time. When querying this table, the multi-column statistics will improve cardinality estimations for joins, GROUP BY aggregations, distinct counts, and WHERE filters (so long as the primary statistics column is a part of the filter).

Create statistics on all columns in a table

One way to create statistics is to issue CREATE STATISTICS commands after creating the table:

```
CREATE TABLE dbo.table1
(
   col1 int
,  col2 int
,  col3 int
)
WITH
  (
    CLUSTERED COLUMNSTORE INDEX
  )
;

CREATE STATISTICS stats_col1 on dbo.table1 (col1);
CREATE STATISTICS stats_col2 on dbo.table2 (col2);
CREATE STATISTICS stats_col3 on dbo.table3 (col3);
```

Use a stored procedure to create statistics on all columns in a database

SQL pool doesn't have a system stored procedure equivalent to `sp_create_stats` in SQL Server. This stored procedure creates a single column statistics object on every column of the database that doesn't already have statistics.

The following example will help you get started with your database design. Feel free to adapt it to your needs:

```
CREATE PROCEDURE    [dbo].[prc_sqldw_create_stats]
(   @create_type    tinyint -- 1 default, 2 Fullscan, 3 Sample
,   @sample_pct     tinyint
)
AS

IF @create_type IS NULL
BEGIN
    SET @create_type = 1;
END;

IF @create_type NOT IN (1,2,3)
BEGIN
    THROW 151000,'Invalid value for @stats_type parameter. Valid range 1 (default), 2 (fullscan) or 3
(sample).',1;
END;

IF @sample_pct IS NULL
BEGIN;
    SET @sample_pct = 20;
END;

IF OBJECT_ID('tempdb..#stats_ddl') IS NOT NULL
BEGIN;
    DROP TABLE #stats_ddl;
END;

CREATE TABLE #stats_ddl
WITH    (   DISTRIBUTION    = HASH([seq_nmbr])
        ,   LOCATION        = USER_DB
        )
AS
WITH T
AS
(
SELECT      t.[name]                        AS [table_name]
,           s.[name]                        AS [table_schema_name]
,           c.[name]                        AS [column_name]
,           c.[column_id]                   AS [column_id]
,           t.[object_id]                   AS [object_id]
,           ROW_NUMBER()
            OVER(ORDER BY (SELECT NULL))    AS [seq_nmbr]
```

```
FROM        sys.[tables] t
JOIN        sys.[schemas] s         ON  t.[schema_id]       = s.[schema_id]
JOIN        sys.[columns] c         ON  t.[object_id]       = c.[object_id]
LEFT JOIN   sys.[stats_columns] l   ON  l.[object_id]       = c.[object_id]
                                    AND l.[column_id]       = c.[column_id]
                                    AND l.[stats_column_id] = 1
LEFT JOIN    sys.[external_tables] e    ON    e.[object_id]        = t.[object_id]
WHERE       l.[object_id] IS NULL
AND            e.[object_id] IS NULL -- not an external table
)
SELECT  [table_schema_name]
,       [table_name]
,       [column_name]
,       [column_id]
,       [object_id]
,       [seq_nmbr]
,       CASE @create_type
        WHEN 1
        THEN    CAST('CREATE STATISTICS '+QUOTENAME('stat_'+table_schema_name+ '_' + table_name +
'_'+column_name)+' ON
'+QUOTENAME(table_schema_name)+'.'+QUOTENAME(table_name)+'('+QUOTENAME(column_name)+')' AS VARCHAR(8000))
        WHEN 2
        THEN    CAST('CREATE STATISTICS '+QUOTENAME('stat_'+table_schema_name+ '_' + table_name +
'_'+column_name)+' ON '+QUOTENAME(table_schema_name)+'.'+QUOTENAME(table_name)+'('+QUOTENAME(column_name)+')
WITH FULLSCAN' AS VARCHAR(8000))
        WHEN 3
        THEN    CAST('CREATE STATISTICS '+QUOTENAME('stat_'+table_schema_name+ '_' + table_name +
'_'+column_name)+' ON '+QUOTENAME(table_schema_name)+'.'+QUOTENAME(table_name)+'('+QUOTENAME(column_name)+')
WITH SAMPLE '+CONVERT(varchar(4),@sample_pct)+' PERCENT' AS VARCHAR(8000))
        END AS create_stat_ddl
FROM T
;

DECLARE @i INT              = 1
,       @t INT              = (SELECT COUNT(*) FROM #stats_ddl)
,       @s NVARCHAR(4000)   = N''
;

WHILE @i <= @t
BEGIN
    SET @s=(SELECT create_stat_ddl FROM #stats_ddl WHERE seq_nmbr = @i);

    PRINT @s
    EXEC sp_executesql @s
    SET @i+=1;
END

DROP TABLE #stats_ddl;
```

To create statistics on all columns in the table using the defaults, execute the stored procedure.

```
EXEC [dbo].[prc_sqldw_create_stats] 1, NULL;
```

To create statistics on all columns in the table using a fullscan, call this procedure:

```
EXEC [dbo].[prc_sqldw_create_stats] 2, NULL;
```

To create sampled statistics on all columns in the table, enter 3, and the sample percent. The procedure below uses a 20 percent sample rate.

```
EXEC [dbo].[prc_sqldw_create_stats] 3, 20;
```

Examples: Update statistics

To update statistics, you can:

- Update one statistics object. Specify the name of the statistics object you want to update.
- Update all statistics objects on a table. Specify the name of the table instead of one specific statistics object.

Update one specific statistics object

Use the following syntax to update a specific statistics object:

```
UPDATE STATISTICS [schema_name].[table_name]([stat_name]);
```

For example:

```
UPDATE STATISTICS [dbo].[table1] ([stats_col1]);
```

By updating specific statistics objects, you can minimize the time and resources required to manage statistics. This action requires some thought for selecting the best statistics objects to update.

Update all statistics on a table

A simple method for updating all the statistics objects on a table is:

```
UPDATE STATISTICS [schema_name].[table_name];
```

For example:

```
UPDATE STATISTICS dbo.table1;
```

The UPDATE STATISTICS statement is easy to use. Just remember that it updates *all* statistics on the table, prompting more work than is necessary.

If performance isn't an issue, this method is the easiest and most complete way to guarantee that statistics are up to date.

> **NOTE**
> When updating all statistics on a table, dedicated SQL pool does a scan to sample the table for each statistics object. If the table is large and has many columns and many statistics, it might be more efficient to update individual statistics based on need.

For an implementation of an `UPDATE STATISTICS` procedure, see Temporary tables. The implementation method is slightly different from the preceding `CREATE STATISTICS` procedure, but the result is the same. For the full syntax, see Update statistics.

Statistics metadata

There are several system views and functions that you can use to find information about statistics. For example, you can see if a statistics object might be out of date by using the STATS_DATE() function. STATS_DATE() allows you to see when statistics were last created or updated.

Catalog views for statistics

These system views provide information about statistics:

CATALOG VIEW	DESCRIPTION
sys.columns	One row for each column.
sys.objects	One row for each object in the database.
sys.schemas	One row for each schema in the database.
sys.stats	One row for each statistics object.
sys.stats_columns	One row for each column in the statistics object. Links back to sys.columns.
sys.tables	One row for each table (includes external tables).
sys.table_types	One row for each data type.

System functions for statistics

These system functions are useful for working with statistics:

SYSTEM FUNCTION	DESCRIPTION
STATS_DATE	Date the statistics object was last updated.
DBCC SHOW_STATISTICS	Summary level and detailed information about the distribution of values as understood by the statistics object.

Combine statistics columns and functions into one view

This view brings columns that relate to statistics and results from the STATS_DATE() function together.

```
CREATE VIEW dbo.vstats_columns
AS
SELECT
        sm.[name]                           AS [schema_name]
,       tb.[name]                           AS [table_name]
,       st.[name]                           AS [stats_name]
,       st.[filter_definition]              AS [stats_filter_definition]
,       st.[has_filter]                     AS [stats_is_filtered]
,       STATS_DATE(st.[object_id],st.[stats_id])
                                            AS [stats_last_updated_date]
,       co.[name]                           AS [stats_column_name]
,       ty.[name]                           AS [column_type]
,       co.[max_length]                     AS [column_max_length]
,       co.[precision]                      AS [column_precision]
,       co.[scale]                          AS [column_scale]
,       co.[is_nullable]                    AS [column_is_nullable]
,       co.[collation_name]                 AS [column_collation_name]
,       QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])
                                            AS two_part_name
,       QUOTENAME(DB_NAME())+'.'+QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])
                                            AS three_part_name
FROM    sys.objects                         AS ob
JOIN    sys.stats           AS st ON    ob.[object_id]      = st.[object_id]
JOIN    sys.stats_columns   AS sc ON    st.[stats_id]       = sc.[stats_id]
                            AND         st.[object_id]      = sc.[object_id]
JOIN    sys.columns         AS co ON    sc.[column_id]      = co.[column_id]
                            AND         sc.[object_id]      = co.[object_id]
JOIN    sys.types           AS ty ON    co.[user_type_id]   = ty.[user_type_id]
JOIN    sys.tables          AS tb ON    co.[object_id]      = tb.[object_id]
JOIN    sys.schemas         AS sm ON    tb.[schema_id]      = sm.[schema_id]
WHERE   1=1
AND     st.[user_created] = 1
;
```

DBCC SHOW_STATISTICS() examples

DBCC SHOW_STATISTICS() shows the data held within a statistics object. This data comes in three parts:

- Header
- Density vector
- Histogram

The header is the metadata about the statistics. The histogram displays the distribution of values in the first key column of the statistics object.

The density vector measures cross-column correlation. Dedicated SQL pool computes cardinality estimates with any of the data in the statistics object.

Show header, density, and histogram

This simple example shows all three parts of a statistics object:

```
DBCC SHOW_STATISTICS([<schema_name>.<table_name>],<stats_name>)
```

For example:

```
DBCC SHOW_STATISTICS (dbo.table1, stats_col1);
```

Show one or more parts of DBCC SHOW_STATISTICS()

If you're only interested in viewing specific parts, use the `WITH` clause and specify which parts you want to see:

```
DBCC SHOW_STATISTICS([<schema_name>.<table_name>],<stats_name>)
    WITH stat_header, histogram, density_vector
```

For example:

```
DBCC SHOW_STATISTICS (dbo.table1, stats_col1)
    WITH histogram, density_vector
```

DBCC SHOW_STATISTICS() differences

`DBCC SHOW_STATISTICS()` is more strictly implemented in dedicated SQL pool compared to SQL Server:

- Undocumented features aren't supported.
- Can't use Stats_stream.
- Can't join results for specific subsets of statistics data. For example, STAT_HEADER JOIN DENSITY_VECTOR.
- NO_INFOMSGS can't be set for message suppression.
- Square brackets around statistics names can't be used.
- Can't use column names to identify statistics objects.
- Custom error 2767 isn't supported.

Statistics in serverless SQL pool

Statistics are created per particular column for particular dataset (storage path).

> **NOTE**
> Statistics cannot be created for LOB columns.

Why use statistics

The more serverless SQL pool knows about your data, the faster it can execute queries against it. Collecting statistics on your data is one of the most important things you can do to optimize your queries.

The serverless SQL pool query optimizer is a cost-based optimizer. It compares the cost of various query plans, and then chooses the plan with the lowest cost. In most cases, it chooses the plan that will execute the fastest.

For example, if the optimizer estimates that the date your query is filtering on will return one row it will choose one plan. If it estimates that the selected date will return 1 million rows, it will return a different plan.

Automatic creation of statistics

Serverless SQL pool analyzes incoming user queries for missing statistics. If statistics are missing, the query optimizer creates statistics on individual columns in the query predicate or join condition to improve cardinality estimates for the query plan.

The SELECT statement will trigger automatic creation of statistics.

> **NOTE**
> Automatic creation of statistics is turned on for Parquet files. For CSV files, you need to create statistics manually until automatic creation of CSV files statistics is supported.

Automatic creation of statistics is done synchronously so you may incur slightly degraded query performance if your columns are missing statistics. The time to create statistics for a single column depends on the size of the files targeted.

Manual creation of statistics

Serverless SQL pool lets you create statistics manually. For CSV files, you have to create statistics manually because automatic creation of statistics isn't turned on for CSV files.

See the following examples for instructions on how to manually create statistics.

Update statistics

Changes to data in files, deleting, and adding files result in data distribution changes and makes statistics out of date. In that case, statistics needs to be updated.

Serverless SQL pool automatically recreates statistics if data is changed significantly. Every time statistics are automatically created, the current state of the dataset is also saved: file paths, sizes, last modification dates.

When statistics are stale, new ones will be created. The algorithm goes through the data and compares it to the current state of the dataset. If the size of the changes is greater than the specific threshold, then old stats are deleted and will be re-created over the new dataset.

Manual stats are never declared stale.

> **NOTE**
> Automatic recreation of statistics is turned on for Parquet files. For CSV files, you need to drop and create statistics manually until automatic creation of CSV files statistics is supported. Check the examples below on how to drop and create statistics.

One of the first questions to ask when you're troubleshooting a query is, **"Are the statistics up to date?"**

When the number of rows has changed substantially, or there's a material change in the distribution of values for a column, *then* it's time to update statistics.

> **NOTE**
> If there is a material change in the distribution of values for a column, you should update statistics regardless of the last time they were updated.

Implement statistics management

You may want to extend your data pipeline to ensure that statistics are updated when data is significantly changed through addition, deletion, or change of files.

The following guiding principles are provided for updating your statistics:

- Ensure that the dataset has at least one statistics object updated. This updates size (row count and page count) information as part of the statistics update.
- Focus on columns participating in WHERE, JOIN, GROUP BY, ORDER BY, and DISTINCT clauses.
- Update "ascending key" columns such as transaction dates more frequently because these values won't be included in the statistics histogram.
- Update static distribution columns less frequently.

For more information, see Cardinality Estimation.

Examples: Create statistics for column in OPENROWSET path

The following examples show you how to use various options for creating statistics in Azure Synapse serverless SQL pools. The options that you use for each column depend on the characteristics of your data and how the column will be used in queries. For more information on the stored procedures used in these examples, review sys.sp_create_openrowset_statistics and sys.sp_drop_openrowset_statistics, which apply to serverless SQL pools

only.

> **NOTE**
>
> You can create single-column statistics only at this moment.
>
> Following permissions are required to execute `sp_create_openrowset_statistics` and `sp_drop_openrowset_statistics`: ADMINISTER BULK OPERATIONS or ADMINISTER DATABASE BULK OPERATIONS.

The following stored procedure is used to create statistics:

```
sys.sp_create_openrowset_statistics [ @stmt = ] N'statement_text'
```

Arguments: [@stmt =] N'statement_text' - Specifies a Transact-SQL statement that will return column values to be used for statistics. You can use TABLESAMPLE to specify samples of data to be used. If TABLESAMPLE isn't specified, FULLSCAN will be used.

```
<tablesample_clause> ::= TABLESAMPLE ( sample_number PERCENT )
```

> **NOTE**
>
> CSV sampling does not work at this time, only FULLSCAN is supported for CSV.

Create single-column statistics by examining every row

To create statistics on a column, provide a query that returns the column for which you need statistics.

By default, if you don't specify otherwise, serverless SQL pool uses 100% of the data provided in the dataset when it creates statistics.

For example, to create statistics with default options (FULLSCAN) for a year column of the dataset based on the population.csv file:

```
/* make sure you have credentials for storage account access created
IF EXISTS (SELECT * FROM sys.credentials WHERE name =
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer')
DROP CREDENTIAL [https://azureopendatastorage.blob.core.windows.net/censusdatacontainer]
GO

CREATE CREDENTIAL [https://azureopendatastorage.blob.core.windows.net/censusdatacontainer]
WITH IDENTITY='SHARED ACCESS SIGNATURE',
SECRET = ''
GO
*/

EXEC sys.sp_create_openrowset_statistics N'SELECT year
FROM OPENROWSET(
        BULK ''https://sqlondemandstorage.blob.core.windows.net/csv/population/population.csv'',
        FORMAT = ''CSV'',
        FIELDTERMINATOR ='','',
        ROWTERMINATOR = ''\n''
    )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
) AS [r]
'
```

Create single-column statistics by specifying the sample size

You can specify the sample size as a percent:

```
/* make sure you have credentials for storage account access created
IF EXISTS (SELECT * FROM sys.credentials WHERE name =
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer')
DROP CREDENTIAL [https://azureopendatastorage.blob.core.windows.net/censusdatacontainer]
GO

CREATE CREDENTIAL [https://azureopendatastorage.blob.core.windows.net/censusdatacontainer]
WITH IDENTITY='SHARED ACCESS SIGNATURE',
SECRET = ''
GO
*/

EXEC sys.sp_create_openrowset_statistics N'SELECT payment_type
FROM OPENROWSET(
        BULK ''https://sqlondemandstorage.blob.core.windows.net/parquet/taxi/year=2018/month=6/*.parquet'',
         FORMAT = ''PARQUET''
    ) AS [nyc]
    TABLESAMPLE(5 PERCENT)
'
```

Examples: Update statistics

To update statistics, you need to drop and create statistics. For more information, review sys.sp_create_openrowset_statistics and sys.sp_drop_openrowset_statistics.

The `sys.sp_drop_openrowset_statistics` stored procedure is used to drop statistics:

```
sys.sp_drop_openrowset_statistics [ @stmt = ] N'statement_text'
```

> **NOTE**
>
> Following permissions are required to execute `sp_create_openrowset_statistics` and `sp_drop_openrowset_statistics`: ADMINISTER BULK OPERATIONS or ADMINISTER DATABASE BULK OPERATIONS.

Arguments: [@stmt =] N'statement_text' - Specifies the same Transact-SQL statement used when the statistics were created.

To update the statistics for the year column in the dataset, which is based on the `population.csv` file, you need to drop and create statistics:

```
EXEC sys.sp_drop_openrowset_statistics N'SELECT payment_type
FROM OPENROWSET(
        BULK ''https://sqlondemandstorage.blob.core.windows.net/parquet/taxi/year=2018/month=6/*.parquet'',
         FORMAT = ''PARQUET''
    ) AS [nyc]
    TABLESAMPLE(5 PERCENT)
'
GO

/* make sure you have credentials for storage account access created
IF EXISTS (SELECT * FROM sys.credentials WHERE name =
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer')
DROP CREDENTIAL [https://azureopendatastorage.blob.core.windows.net/censusdatacontainer]
GO

CREATE CREDENTIAL [https://azureopendatastorage.blob.core.windows.net/censusdatacontainer]
WITH IDENTITY='SHARED ACCESS SIGNATURE',
SECRET = ''
GO
*/

EXEC sys.sp_create_openrowset_statistics N'SELECT payment_type
FROM OPENROWSET(
        BULK ''https://sqlondemandstorage.blob.core.windows.net/parquet/taxi/year=2018/month=6/*.parquet'',
         FORMAT = ''PARQUET''
    ) AS [nyc]
    TABLESAMPLE(5 PERCENT)
'
```

Examples: Create statistics for external table column

The following examples show you how to use various options for creating statistics. The options that you use for each column depend on the characteristics of your data and how the column will be used in queries.

> **NOTE**
>
> You can create single-column statistics only at this moment.

To create statistics on a column, provide a name for the statistics object and the name of the column.

```
CREATE STATISTICS statistics_name
ON { external_table } ( column )
    WITH
        { FULLSCAN
          | [ SAMPLE number PERCENT ] }
        , { NORECOMPUTE }
```

Arguments: external_table Specifies external table that statistics should be created.

FULLSCAN Compute statistics by scanning all rows. FULLSCAN and SAMPLE 100 PERCENT have the same results. FULLSCAN can't be used with the SAMPLE option.

SAMPLE number PERCENT Specifies the approximate percentage or number of rows in the table or indexed view for the query optimizer to use when it creates statistics. Number can be from 0 through 100.

SAMPLE can't be used with the FULLSCAN option.

> **NOTE**
> CSV sampling does not work at this time, only FULLSCAN is supported for CSV.

Create single-column statistics by examining every row

```
CREATE STATISTICS sState
    on census_external_table (STATENAME)
    WITH FULLSCAN, NORECOMPUTE
```

Create single-column statistics by specifying the sample size

```
-- following sample creates statistics with sampling 20%
CREATE STATISTICS sState
    on census_external_table (STATENAME)
    WITH SAMPLE 5 percent, NORECOMPUTE
```

Examples: Update statistics

To update statistics, you need to drop and create statistics. Drop statistics first:

```
DROP STATISTICS census_external_table.sState
```

And create statistics:

```
CREATE STATISTICS sState
    on census_external_table (STATENAME)
    WITH FULLSCAN, NORECOMPUTE
```

Statistics metadata

There are several system views and functions that you can use to find information about statistics. For example, you can see if a statistics object might be out of date by using the STATS_DATE() function. STATS_DATE() allows you to see when statistics were last created or updated.

> **NOTE**
> Statistics metadata is available for external table columns only. Statistics metadata is not available for OPENROWSET columns.

Catalog views for statistics

These system views provide information about statistics:

CATALOG VIEW	DESCRIPTION
sys.columns	One row for each column.

CATALOG VIEW	DESCRIPTION
sys.objects	One row for each object in the database.
sys.schemas	One row for each schema in the database.
sys.stats	One row for each statistics object.
sys.stats_columns	One row for each column in the statistics object. Links back to sys.columns.
sys.tables	One row for each table (includes external tables).
sys.table_types	One row for each data type.

System functions for statistics

These system functions are useful for working with statistics:

SYSTEM FUNCTION	DESCRIPTION
STATS_DATE	Date the statistics object was last updated.

Combine statistics columns and functions into one view

This view brings columns that relate to statistics and results from the STATS_DATE() function together.

```
CREATE VIEW dbo.vstats_columns
AS
SELECT
        sm.[name]                           AS [schema_name]
,       tb.[name]                           AS [table_name]
,       st.[name]                           AS [stats_name]
,       st.[filter_definition]              AS [stats_filter_definition]
,       st.[has_filter]                     AS [stats_is_filtered]
,       STATS_DATE(st.[object_id],st.[stats_id])
                                            AS [stats_last_updated_date]
,       co.[name]                           AS [stats_column_name]
,       ty.[name]                           AS [column_type]
,       co.[max_length]                     AS [column_max_length]
,       co.[precision]                      AS [column_precision]
,       co.[scale]                          AS [column_scale]
,       co.[is_nullable]                    AS [column_is_nullable]
,       co.[collation_name]                 AS [column_collation_name]
,       QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])
                                            AS two_part_name
,       QUOTENAME(DB_NAME())+'.'+QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])
                                            AS three_part_name
FROM    sys.objects                         AS ob
JOIN    sys.stats           AS st ON    ob.[object_id]      = st.[object_id]
JOIN    sys.stats_columns   AS sc ON    st.[stats_id]       = sc.[stats_id]
                            AND         st.[object_id]      = sc.[object_id]
JOIN    sys.columns         AS co ON    sc.[column_id]      = co.[column_id]
                            AND         sc.[object_id]      = co.[object_id]
JOIN    sys.types           AS ty ON    co.[user_type_id]   = ty.[user_type_id]
JOIN    sys.tables          AS tb ON    co.[object_id]      = tb.[object_id]
JOIN    sys.schemas         AS sm ON    tb.[schema_id]      = sm.[schema_id]
WHERE   st.[user_created] = 1
;
```

Next steps

To further improve query performance for dedicated SQL pool, see Monitor your workload and Best practices for dedicated SQL pool.

To further improve query performance for serverless SQL pool, see Best practices for serverless SQL pool.

Temporary tables in Synapse SQL

8/19/2022 • 4 minutes to read • Edit Online

This article contains essential guidance for using temporary tables and highlights the principles of session level temporary tables within Synapse SQL.

Both the dedicated SQL pool and serverless SQL pool resources can utilize temporary tables. Serverless SQL pool has limitations that are discussed at the end of this article.

Temporary tables

Temporary tables are useful when processing data, especially during transformation where the intermediate results are transient. With Synapse SQL, temporary tables exist at the session level. They're only visible to the session in which they were created. As such, they're automatically dropped when that session ends.

Temporary tables in dedicated SQL pool

In the dedicated SQL pool resource, temporary tables offer a performance benefit because their results are written to local rather than remote storage.

Create a temporary table

Temporary tables are created by prefixing your table name with a `#`. For example:

```
CREATE TABLE #stats_ddl
(
    [schema_name]        NVARCHAR(128) NOT NULL
,    [table_name]            NVARCHAR(128) NOT NULL
,    [stats_name]            NVARCHAR(128) NOT NULL
,    [stats_is_filtered]     BIT           NOT NULL
,    [seq_nmbr]              BIGINT        NOT NULL
,    [two_part_name]         NVARCHAR(260) NOT NULL
,    [three_part_name]       NVARCHAR(400) NOT NULL
)
WITH
(
    DISTRIBUTION = HASH([seq_nmbr])
,    HEAP
)
```

Temporary tables can also be created with a `CTAS` using exactly the same approach:

```
CREATE TABLE #stats_ddl
WITH
(
    DISTRIBUTION = HASH([seq_nmbr])
,    HEAP
)
AS
(
SELECT
        sm.[name]                                                                   AS [schema_name]
,        tb.[name]                                                                   AS [table_name]
,        st.[name]                                                                   AS [stats_name]
,        st.[has_filter]                                                               AS [stats_is_filtered]
,       ROW_NUMBER()
        OVER(ORDER BY (SELECT NULL))                                               AS [seq_nmbr]
,                               QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])  AS [two_part_name]
,        QUOTENAME(DB_NAME())+'.'+QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])  AS [three_part_name]
FROM    sys.objects            AS ob
JOIN    sys.stats           AS st    ON    ob.[object_id]        = st.[object_id]
JOIN    sys.stats_columns    AS sc    ON    st.[stats_id]        = sc.[stats_id]
                                AND st.[object_id]        = sc.[object_id]
JOIN    sys.columns            AS co    ON    sc.[column_id]        = co.[column_id]
                                AND    sc.[object_id]        = co.[object_id]
JOIN    sys.tables            AS tb    ON    co.[object_id]        = tb.[object_id]
JOIN    sys.schemas            AS sm    ON    tb.[schema_id]        = sm.[schema_id]
WHERE    1=1
AND        st.[user_created]   = 1
GROUP BY
        sm.[name]
,        tb.[name]
,        st.[name]
,        st.[filter_definition]
,        st.[has_filter]
)
;
```

NOTE

`CTAS` is a powerful command and has the added advantage of being efficient in its use of transaction log space.

Drop temporary tables

When a new session is created, no temporary tables should exist. However, if you're calling the same stored procedure that creates a temporary with the same name, to ensure that your `CREATE TABLE` statements are successful, use a simple pre-existence check with `DROP`:

```
IF OBJECT_ID('tempdb..#stats_ddl') IS NOT NULL
BEGIN
    DROP TABLE #stats_ddl
END
```

For coding consistency, it's a good practice to use this pattern for both tables and temporary tables. It's also a good idea to use `DROP TABLE` to remove temporary tables when you're finished with them.

In stored procedure development, it's common to see the drop commands bundled together at the end of a procedure to ensure these objects are cleaned up.

```
DROP TABLE #stats_ddl
```

Modularize code

Temporary tables can be used anywhere in a user session. This capability can then be exploited to help you modularize your application code. To demonstrate, the following stored procedure generates DDL to update all statistics in the database by statistic name:

```
CREATE PROCEDURE    [dbo].[prc_sqldw_update_stats]
(   @update_type    tinyint -- 1 default 2 fullscan 3 sample 4 resample
    ,@sample_pct     tinyint
)
AS

IF @update_type NOT IN (1,2,3,4)
BEGIN;
    THROW 151000,'Invalid value for @update_type parameter. Valid range 1 (default), 2 (fullscan), 3
(sample) or 4 (resample).',1;
END;

IF @sample_pct IS NULL
BEGIN;
    SET @sample_pct = 20;
END;

IF OBJECT_ID('tempdb..#stats_ddl') IS NOT NULL
BEGIN
    DROP TABLE #stats_ddl
END

CREATE TABLE #stats_ddl
WITH
(
    DISTRIBUTION = HASH([seq_nmbr])
)
AS
(
SELECT
        sm.[name]                                                                   AS [schema_name]
,        tb.[name]                                                                   AS [table_name]
,        st.[name]                                                                   AS [stats_name]
,        st.[has_filter]                                                               AS [stats_is_filtered]
,       ROW_NUMBER()
        OVER(ORDER BY (SELECT NULL))                                               AS [seq_nmbr]
,                                QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])  AS [two_part_name]
,        QUOTENAME(DB_NAME())+'.'+QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])  AS [three_part_name]
FROM    sys.objects            AS ob
JOIN    sys.stats            AS st    ON    ob.[object_id]        = st.[object_id]
JOIN    sys.stats_columns    AS sc    ON    st.[stats_id]        = sc.[stats_id]
                                AND st.[object_id]        = sc.[object_id]
JOIN    sys.columns            AS co    ON    sc.[column_id]        = co.[column_id]
                                AND    sc.[object_id]        = co.[object_id]
JOIN    sys.tables            AS tb    ON    co.[object_id]        = tb.[object_id]
JOIN    sys.schemas            AS sm    ON    tb.[schema_id]        = sm.[schema_id]
WHERE    1=1
AND        st.[user_created]   = 1
GROUP BY
        sm.[name]
,        tb.[name]
,        st.[name]
,        st.[filter_definition]
,        st.[has_filter]
)
SELECT
    CASE @update_type
    WHEN 1
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+');'
    WHEN 2
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+') WITH FULLSCAN;'
    WHEN 3
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+') WITH SAMPLE '+CAST(@sample_pct AS
```

```
VARCHAR(20))+' PERCENT;'
    WHEN 4
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+') WITH RESAMPLE;'
    END AS [update_stats_ddl]
,   [seq_nmbr]
FROM    #stats_ddl
;
GO
```

At this stage, the only action that has occurred is the creation of a stored procedure that generates the #stats_ddl temporary table. The stored procedure drops #stats_ddl if it already exists. This drop ensures it doesn't fail if run more than once within a session.

Since there isn't a `DROP TABLE` at the end of the stored procedure, when the stored procedure completes, the created table remains and can be read outside of the stored procedure.

In contrast to other SQL Server databases, Synapse SQL allows you to use the temporary table outside of the procedure that created it. The temporary tables created via dedicated SQL pool can be used **anywhere** inside the session. As a result, you'll have more modular and manageable code, as demonstrated in the sample below:

```
EXEC [dbo].[prc_sqldw_update_stats] @update_type = 1, @sample_pct = NULL;

DECLARE @i INT              = 1
,       @t INT              = (SELECT COUNT(*) FROM #stats_ddl)
,       @s NVARCHAR(4000)   = N''

WHILE @i <= @t
BEGIN
    SET @s=(SELECT update_stats_ddl FROM #stats_ddl WHERE seq_nmbr = @i);

    PRINT @s
    EXEC sp_executesql @s
    SET @i+=1;
END

DROP TABLE #stats_ddl;
```

Temporary table limitations

Dedicated SQL pool does have a few implementation limitations for temporary tables:

- Only session scoped temporary tables are supported. Global Temporary Tables aren't supported.
- Views can't be created on temporary tables.
- Temporary tables can only be created with hash or round robin distribution. Replicated temporary table distribution isn't supported.

Temporary tables in serverless SQL pool

Temporary tables in serverless SQL pool are supported but their usage is limited. They can't be used in queries which target files.

For example, you can't join a temporary table with data from files in storage. The number of temporary tables is limited to 100, and their total size is limited to 100 MB.

Next steps

To learn more about developing tables, see the Designing tables using the Synapse SQL resources article.

Use T-SQL loops with Synapse SQL in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

This article provides you with essential tips for using T-SQL loops, replacing cursors, and developing related solutions with Synapse SQL.

Purpose of WHILE loops

Synapse SQL supports the WHILE loop for repeatedly executing statement blocks. This WHILE loop continues for as long as the specified conditions are true or until the code specifically terminates the loop using the BREAK keyword.

Loops in Synapse SQL are useful for replacing cursors defined in SQL code. Fortunately, almost all cursors that are written in SQL code are of the fast forward, read-only variety. So, WHILE loops are a great alternative for replacing cursors.

Replace cursors in Synapse SQL

Before diving in, the following question should be considered: "Could this cursor be rewritten to use set-based operations?" In many cases, the answer is yes and is frequently the best approach. A set-based operation often executes faster than an iterative, row by row approach.

Fast forward read-only cursors are easily replaced with a looping construct. The following code is a simple example. This code example updates the statistics for every table in the database. By iterating over the tables in the loop, each command executes in sequence.

First, create a temporary table containing a unique row number used to identify the individual statements:

```
CREATE TABLE #tbl
WITH
( DISTRIBUTION = ROUND_ROBIN
)
AS
SELECT  ROW_NUMBER() OVER(ORDER BY (SELECT NULL)) AS Sequence
,       [name]
,       'UPDATE STATISTICS '+QUOTENAME([name]) AS sql_code
FROM    sys.tables
;
```

Second, initialize the variables required to execute the loop:

```
DECLARE @nbr_statements INT = (SELECT COUNT(*) FROM #tbl)
,       @i INT = 1
;
```

Now loop over statements executing them one at a time:

```
WHILE   @i <= @nbr_statements
BEGIN
    DECLARE @sql_code NVARCHAR(4000) = (SELECT sql_code FROM #tbl WHERE Sequence = @i);
    EXEC    sp_executesql @sql_code;
    SET     @i +=1;
END
```

Finally drop the temporary table created in the first step

```
DROP TABLE #tbl;
```

Next steps

For more development tips, see development overview.

Stored procedures using Synapse SQL in Azure Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

Synapse SQL provisioned and serverless pools enable you to place complex data processing logic into SQL stored procedures. Stored procedures are a great way for encapsulating your SQL code and storing it close to your data in the data warehouse. Stored procedures help developers modularize their solutions by encapsulating the code into manageable units, and facilitating greater reusability of code. Each stored procedure can also accept parameters to make them even more flexible. In this article you will find some tips for implementing stored procedures in Synapse SQL pool for developing solutions.

What to expect

Synapse SQL supports many of the T-SQL features that are used in SQL Server. More importantly, there are scale-out specific features that you can use to maximize the performance of your solution. In this article, you will learn about the features that you can place in stored procedures.

> **NOTE**
>
> In the procedure body you can use only the features that are supported in Synapse SQL surface area. Review this article to identify objects, statement that can be used in stored procedures. The examples in these articles use generic features that are available both in serverless and dedicated surface area. See additional limitations in provisioned and serverless Synapse SQL pools at the end of this article.

To maintain the scale and performance of SQL pool, there are also some features and functionality that have behavioral differences and others that aren't supported.

Stored procedures in Synapse SQL

In the following example, you can see the procedures that drop external objects if they exist in the database:

```
CREATE PROCEDURE drop_external_table_if_exists @name SYSNAME
AS BEGIN
    IF (0 <> (SELECT COUNT(*) FROM sys.external_tables WHERE name = @name))
    BEGIN
        DECLARE @drop_stmt NVARCHAR(200) = N'DROP EXTERNAL TABLE ' + @name;
        EXEC sp_executesql @tsql = @drop_stmt;
    END
END
GO
CREATE PROCEDURE drop_external_file_format_if_exists @name SYSNAME
AS BEGIN
    IF (0 <> (SELECT COUNT(*) FROM sys.external_file_formats WHERE name = @name))
    BEGIN
        DECLARE @drop_stmt NVARCHAR(200) = N'DROP EXTERNAL FILE FORMAT ' + @name;
        EXEC sp_executesql @tsql = @drop_stmt;
    END
END
GO
CREATE PROCEDURE drop_external_data_source_if_exists @name SYSNAME
AS BEGIN
    IF (0 <> (SELECT COUNT(*) FROM sys.external_data_sources WHERE name = @name))
    BEGIN
        DECLARE @drop_stmt NVARCHAR(200) = N'DROP EXTERNAL DATA SOURCE ' + @name;
        EXEC sp_executesql @tsql = @drop_stmt;
    END
END
```

These procedures can be executed using `EXEC` statement where you can specify the procedure name and parameters:

```
EXEC drop_external_table_if_exists 'mytest';
EXEC drop_external_file_format_if_exists 'mytest';
EXEC drop_external_data_source_if_exists 'mytest';
```

Synapse SQL provides a simplified and streamlined stored procedure implementation. The biggest difference compared to SQL Server is that the stored procedure is not pre-compiled code. In data warehouses, the compilation time is small in comparison to the time it takes to run queries against large data volumes. It is more important to ensure the stored procedure code is correctly optimized for large queries. The goal is to save hours, minutes, and seconds, not milliseconds. It is therefore more helpful to think of stored procedures as containers for SQL logic.

When Synapse SQL executes your stored procedure, the SQL statements are parsed, translated, and optimized at run time. During this process, each statement is converted into distributed queries. The SQL code that is executed against the data is different than the query submitted.

Encapsulate validation rules

Stored procedures enable you to locate validation logic in a single module stored in SQL database. In the following example, you can see how to validate the values of parameters and change their default values.

```
CREATE PROCEDURE count_objects_by_date_created
                            @start_date DATETIME2,
                            @end_date DATETIME2
AS BEGIN

    IF( @start_date >= GETUTCDATE() )
    BEGIN
        THROW 51000, 'Invalid argument @start_date. Value should be in past.', 1;
    END

    IF( @end_date IS NULL )
    BEGIN
        SET @end_date = GETUTCDATE();
    END

    IF( @start_date >= @end_date )
    BEGIN
        THROW 51000, 'Invalid argument @end_date. Value should be greater than @start_date.', 2;
    END

    SELECT
         year = YEAR(create_date),
         month = MONTH(create_date),
         objects_created = COUNT(*)
    FROM
        sys.objects
    WHERE
        create_date BETWEEN @start_date AND @end_date
    GROUP BY
        YEAR(create_date), MONTH(create_date);
END
```

The logic in the sql procedure will validate the input parameters when the procedure is called.

```
EXEC count_objects_by_date_created '2020-08-01', '2020-09-01'

EXEC count_objects_by_date_created '2020-08-01', NULL

EXEC count_objects_by_date_created '2020-09-01', '2020-08-01'
-- Error
-- Invalid argument @end_date. Value should be greater than @start_date.

EXEC count_objects_by_date_created '2120-09-01', NULL
-- Error
-- Invalid argument @start_date. Value should be in past.
```

Nesting stored procedures

When stored procedures call other stored procedures, or execute dynamic SQL, then the inner stored procedure or code invocation is said to be nested. An example of nested procedure is shown in the following code:

```
CREATE PROCEDURE clean_up @name SYSNAME
AS BEGIN
    EXEC drop_external_table_if_exists @name;
    EXEC drop_external_file_format_if_exists @name;
    EXEC drop_external_data_source_if_exists @name;
END
```

This procedure accepts a parameter that represents some name and then calls other procedures to drop the objects with this name. Synapse SQL pool supports a maximum of eight nesting levels. This capability is slightly

different than SQL Server. The nest level in SQL Server is 32.

The top-level stored procedure call equates to nest level 1.

```
EXEC clean_up 'mytest'
```

If the stored procedure also makes another EXEC call, the nest level increases to two.

```
CREATE PROCEDURE clean_up @name SYSNAME
AS
    EXEC drop_external_table_if_exists @name  -- This call is nest level 2
GO
EXEC clean_up 'mytest'  -- This call is nest level 1
```

If the second procedure then executes some dynamic SQL, the nest level increases to three.

```
CREATE PROCEDURE drop_external_table_if_exists @name SYSNAME
AS BEGIN
    /* See full code in the previous example */
    EXEC sp_executesql @tsql = @drop_stmt;  -- This call is nest level 3
END
GO
CREATE PROCEDURE clean_up @name SYSNAME
AS
    EXEC drop_external_table_if_exists @name  -- This call is nest level 2
GO
EXEC clean_up 'mytest'  -- This call is nest level 1
```

> **NOTE**
> Synapse SQL does not currently support @@NESTLEVEL. You need to track the nest level. It is unlikely for you to exceed the eight nest level limit, but if you do, you need to rework your code to fit the nesting levels within this limit.

INSERT..EXECUTE

Provisioned Synapse SQL pool doesn't permit you to consume the result set of a stored procedure with an INSERT statement. There's an alternative approach you can use. For an example, see the article on temporary tables for provisioned Synapse SQL pool.

Limitations

There are some aspects of Transact-SQL stored procedures that aren't implemented in Synapse SQL, such as:

FEATURE/OPTION	PROVISIONED	SERVERLESS
Temporary stored procedures	No	Yes
Numbered stored procedures	No	No
Extended stored procedures	No	No
CLR stored procedures	No	No
Encryption option	No	Yes

FEATURE/OPTION	PROVISIONED	SERVERLESS
Replication option	No	No
Table-valued parameters	No	No
Read-only parameters	No	No
Default parameters	No	Yes
Execution contexts	No	No
Return statement	No	Yes
INSERT INTO .. EXEC	No	Yes

Next steps

For more development tips, see development overview.

Use transactions with dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

Tips for implementing transactions with dedicated SQL pool in Azure Synapse Analytics for developing solutions.

What to expect

As you would expect, dedicated SQL pool supports transactions as part of the data warehouse workload. However, to ensure the performance of dedicated SQL pool is maintained at scale some features are limited when compared to SQL Server. This article highlights the differences and lists the others.

Transaction isolation levels

Dedicated SQL pool implements ACID transactions. The isolation level of the transactional support is default to READ UNCOMMITTED. You can change it to READ COMMITTED SNAPSHOT ISOLATION by turning ON the READ_COMMITTED_SNAPSHOT database option for a user database when connected to the master database.

Once enabled, all transactions in this database are executed under READ COMMITTED SNAPSHOT ISOLATION and setting READ UNCOMMITTED on session level will not be honored. Check ALTER DATABASE SET options (Transact-SQL) for details.

Transaction size

A single data modification transaction is limited in size. The limit is applied per distribution. As such, the total allocation can be calculated by multiplying the limit by the distribution count.

To approximate the maximum number of rows in the transaction divide the distribution cap by the total size of each row. For variable length columns, consider taking an average column length rather than using the maximum size.

In the table below the following assumptions have been made:

- An even distribution of data has occurred
- The average row length is 250 bytes

Gen2

DWU	CAP PER DISTRIBUTION (GB)	NUMBER OF DISTRIBUTIONS	MAX TRANSACTION SIZE (GB)	# ROWS PER DISTRIBUTION	MAX ROWS PER TRANSACTION
DW100c	1	60	60	4,000,000	240,000,000
DW200c	1.5	60	90	6,000,000	360,000,000
DW300c	2.25	60	135	9,000,000	540,000,000
DW400c	3	60	180	12,000,000	720,000,000

DWU	CAP PER DISTRIBUTION (GB)	NUMBER OF DISTRIBUTIONS	MAX TRANSACTION SIZE (GB)	# ROWS PER DISTRIBUTION	MAX ROWS PER TRANSACTION
DW500c	3.75	60	225	15,000,000	900,000,000
DW1000c	7.5	60	450	30,000,000	1,800,000,000
DW1500c	11.25	60	675	45,000,000	2,700,000,000
DW2000c	15	60	900	60,000,000	3,600,000,000
DW2500c	18.75	60	1125	75,000,000	4,500,000,000
DW3000c	22.5	60	1,350	90,000,000	5,400,000,000
DW5000c	37.5	60	2,250	150,000,000	9,000,000,000
DW6000c	45	60	2,700	180,000,000	10,800,000,000
DW7500c	56.25	60	3,375	225,000,000	13,500,000,000
DW10000c	75	60	4,500	300,000,000	18,000,000,000
DW15000c	112.5	60	6,750	450,000,000	27,000,000,000
DW30000c	225	60	13,500	900,000,000	54,000,000,000

Gen1

DWU	CAP PER DISTRIBUTION (GB)	NUMBER OF DISTRIBUTIONS	MAX TRANSACTION SIZE (GB)	# ROWS PER DISTRIBUTION	MAX ROWS PER TRANSACTION
DW100	1	60	60	4,000,000	240,000,000
DW200	1.5	60	90	6,000,000	360,000,000
DW300	2.25	60	135	9,000,000	540,000,000
DW400	3	60	180	12,000,000	720,000,000
DW500	3.75	60	225	15,000,000	900,000,000
DW600	4.5	60	270	18,000,000	1,080,000,000
DW1000	7.5	60	450	30,000,000	1,800,000,000
DW1200	9	60	540	36,000,000	2,160,000,000
DW1500	11.25	60	675	45,000,000	2,700,000,000
DW2000	15	60	900	60,000,000	3,600,000,000

DWU	CAP PER DISTRIBUTION (GB)	NUMBER OF DISTRIBUTIONS	MAX TRANSACTION SIZE (GB)	# ROWS PER DISTRIBUTION	MAX ROWS PER TRANSACTION
DW3000	22.5	60	1,350	90,000,000	5,400,000,000
DW6000	45	60	2,700	180,000,000	10,800,000,000

The transaction size limit is applied per transaction or operation. It is not applied across all concurrent transactions. Therefore each transaction is permitted to write this amount of data to the log.

To optimize and minimize the amount of data written to the log, refer to the Transactions best practices article.

> **WARNING**
> The maximum transaction size can only be achieved for HASH or ROUND_ROBIN distributed tables where the spread of the data is even. If the transaction is writing data in a skewed fashion to the distributions then the limit is likely to be reached prior to the maximum transaction size.

Transaction state

Dedicated SQL pool uses the XACT_STATE() function to report a failed transaction using the value -2. This value means the transaction has failed and is marked for rollback only.

> **NOTE**
> The use of -2 by the XACT_STATE function to denote a failed transaction represents different behavior to SQL Server. SQL Server uses the value -1 to represent an uncommittable transaction. SQL Server can tolerate some errors inside a transaction without it having to be marked as uncommittable. For example `SELECT 1/0` would cause an error but not force a transaction into an uncommittable state. SQL Server also permits reads in the uncommittable transaction. However, dedicated SQL pool does not let you do this. If an error occurs inside a dedicated SQL pool transaction it will automatically enter the -2 state and you will not be able to make any further select statements until the statement has been rolled back. It is therefore important to check that your application code to see if it uses XACT_STATE() as you may need to make code modifications.

For example, in SQL Server you might see a transaction that looks like the following:

```
SET NOCOUNT ON;
DECLARE @xact_state smallint = 0;

BEGIN TRAN
    BEGIN TRY
        DECLARE @i INT;
        SET     @i = CONVERT(INT,'ABC');
    END TRY
    BEGIN CATCH
        SET @xact_state = XACT_STATE();

        SELECT  ERROR_NUMBER()    AS ErrNumber
        ,       ERROR_SEVERITY()  AS ErrSeverity
        ,       ERROR_STATE()     AS ErrState
        ,       ERROR_PROCEDURE() AS ErrProcedure
        ,       ERROR_MESSAGE()   AS ErrMessage
        ;

        IF @@TRANCOUNT > 0
        BEGIN
            ROLLBACK TRAN;
            PRINT 'ROLLBACK';
        END

    END CATCH;

IF @@TRANCOUNT >0
BEGIN
    PRINT 'COMMIT';
    COMMIT TRAN;
END

SELECT @xact_state AS TransactionState;
```

The preceding code gives the following error message:

Msg 111233, Level 16, State 1, Line 1 111233; The current transaction has aborted, and any pending changes have been rolled back. Cause: A transaction in a rollback-only state wasn't explicitly rolled back before a DDL, DML, or SELECT statement.

You won't get the output of the ERROR_* functions.

In dedicated SQL pool, the code needs to be slightly altered:

```
SET NOCOUNT ON;
DECLARE @xact_state smallint = 0;

BEGIN TRAN
    BEGIN TRY
        DECLARE @i INT;
        SET     @i = CONVERT(INT,'ABC');
    END TRY
    BEGIN CATCH
        SET @xact_state = XACT_STATE();

        IF @@TRANCOUNT > 0
        BEGIN
            ROLLBACK TRAN;
            PRINT 'ROLLBACK';
        END

        SELECT  ERROR_NUMBER()    AS ErrNumber
        ,       ERROR_SEVERITY()  AS ErrSeverity
        ,       ERROR_STATE()     AS ErrState
        ,       ERROR_PROCEDURE() AS ErrProcedure
        ,       ERROR_MESSAGE()   AS ErrMessage
        ;
    END CATCH;

IF @@TRANCOUNT >0
BEGIN
    PRINT 'COMMIT';
    COMMIT TRAN;
END

SELECT @xact_state AS TransactionState;
```

The expected behavior is now observed. The error in the transaction is managed and the ERROR_* functions provide values as expected.

All that has changed is that the ROLLBACK of the transaction had to happen before the read of the error information in the CATCH block.

Error_Line() function

It is also worth noting that dedicated SQL pool does not implement or support the ERROR_LINE() function. If you have this function in your code, you need to remove it to be compliant with dedicated SQL pool. Use query labels in your code instead to implement equivalent functionality. For more information, see the LABEL article.

Use of THROW and RAISERROR

THROW is the more modern implementation for raising exceptions in dedicated SQL pool but RAISERROR is also supported. There are a few differences that are worth paying attention to however.

- User-defined error messages numbers can't be in the 100,000 - 150,000 range for THROW
- RAISERROR error messages are fixed at 50,000
- Use of sys.messages is not supported

Limitations

Dedicated SQL pool does have a few other restrictions that relate to transactions. They are as follows:

- No distributed transactions
- No nested transactions permitted

- No save points allowed
- No named transactions
- No marked transactions
- No support for DDL such as CREATE TABLE inside a user-defined transaction

Next steps

To learn more about optimizing transactions, see Transactions best practices. Additional best practices guides are also provided for Dedicated SQL pool and serverless SQL pool.

Optimize transactions with dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 9 minutes to read • Edit Online

Learn how to optimize the performance of your transactional code in dedicated SQL pool while minimizing risk for long rollbacks.

Transactions and logging

Transactions are an important component of a relational database engine. Dedicated SQL pool uses transactions during data modification. These transactions can be explicit or implicit. Single INSERT, UPDATE, and DELETE statements are all examples of implicit transactions. Explicit transactions use BEGIN TRAN, COMMIT TRAN, or ROLLBACK TRAN. Explicit transactions are typically used when multiple modification statements need to be tied together into a single atomic unit.

Dedicated SQL pool commits changes to the database using transaction logs. Each distribution has its own transaction log. Transaction log writes are automatic. There is no configuration required. However, while this process guarantees the write it does introduce an overhead in the system. You can minimize this impact by writing transactionally efficient code. Transactionally efficient code broadly falls into two categories.

- Use minimal logging constructs whenever possible
- Process data using scoped batches to avoid singular long running transactions
- Adopt a partition switching pattern for large modifications to a given partition

Minimal vs. full logging

Unlike fully logged operations, which use the transaction log to keep track of every row change, minimally logged operations keep track of extent allocations and meta-data changes only. Therefore, minimal logging involves logging only the information that is required to roll back the transaction after a failure, or for an explicit request (ROLLBACK TRAN). As much less information is tracked in the transaction log, a minimally logged operation performs better than a similarly sized fully logged operation. Furthermore, because fewer writes go the transaction log, a much smaller amount of log data is generated and so is more I/O efficient.

The transaction safety limits only apply to fully logged operations.

> **NOTE**
> Minimally logged operations can participate in explicit transactions. As all changes in allocation structures are tracked, it is possible to roll back minimally logged operations.

Minimally logged operations

The following operations are capable of being minimally logged:

- CREATE TABLE AS SELECT (CTAS)
- INSERT..SELECT
- CREATE INDEX
- ALTER INDEX REBUILD
- DROP INDEX

- TRUNCATE TABLE
- DROP TABLE
- ALTER TABLE SWITCH PARTITION

> **NOTE**
> Internal data movement operations (such as BROADCAST and SHUFFLE) are not affected by the transaction safety limit.

Minimal logging with bulk load

CTAS and INSERT...SELECT are both bulk load operations. However, both are influenced by the target table definition and depend on the load scenario. The following table explains when bulk operations are fully or minimally logged:

PRIMARY INDEX	LOAD SCENARIO	LOGGING MODE
Heap	Any	**Minimal**
Clustered Index	Empty target table	**Minimal**
Clustered Index	Loaded rows do not overlap with existing pages in target	**Minimal**
Clustered Index	Loaded rows overlap with existing pages in target	Full
Clustered Columnstore Index	Batch size >= 102,400 per partition aligned distribution	**Minimal**
Clustered Columnstore Index	Batch size < 102,400 per partition aligned distribution	Full

It is worth noting that any writes to update secondary or non-clustered indexes will always be fully logged operations.

> **IMPORTANT**
> Dedicated SQL pool has 60 distributions. Therefore, assuming all rows are evenly distributed and landing in a single partition, your batch will need to contain 6,144,000 rows or larger to be minimally logged when writing to a Clustered Columnstore Index. If the table is partitioned and the rows being inserted span partition boundaries, then you will need 6,144,000 rows per partition boundary assuming even data distribution. Each partition in each distribution must independently exceed the 102,400 row threshold for the insert to be minimally logged into the distribution.

Loading data into a non-empty table with a clustered index can often contain a mixture of fully logged and minimally logged rows. A clustered index is a balanced tree (b-tree) of pages. If the page being written to already contains rows from another transaction, then these writes will be fully logged. However, if the page is empty then the write to that page will be minimally logged.

Optimize deletes

DELETE is a fully logged operation. If you need to delete a large amount of data in a table or a partition, it often makes more sense to `SELECT` the data you wish to keep, which can be run as a minimally logged operation. To select the data, create a new table with CTAS. Once created, use RENAME to swap out your old table with the

newly created table.

```
-- Delete all sales transactions for Promotions except PromotionKey 2.

--Step 01. Create a new table select only the records we want to kep (PromotionKey 2)
CREATE TABLE [dbo].[FactInternetSales_d]
WITH
(    CLUSTERED COLUMNSTORE INDEX
,    DISTRIBUTION = HASH([ProductKey])
,     PARTITION     (    [OrderDateKey] RANGE RIGHT
                                    FOR VALUES    (    20000101, 20010101, 20020101, 20030101, 20040101,
20050101
                                                ,    20060101, 20070101, 20080101, 20090101, 20100101,
20110101
                                                ,    20120101, 20130101, 20140101, 20150101, 20160101,
20170101
                                                ,    20180101, 20190101, 20200101, 20210101, 20220101,
20230101
                                                ,    20240101, 20250101, 20260101, 20270101, 20280101,
20290101
                                                )
)
AS
SELECT     *
FROM     [dbo].[FactInternetSales]
WHERE    [PromotionKey] = 2
OPTION (LABEL = 'CTAS : Delete')
;

--Step 02. Rename the Tables to replace the
RENAME OBJECT [dbo].[FactInternetSales]   TO [FactInternetSales_old];
RENAME OBJECT [dbo].[FactInternetSales_d] TO [FactInternetSales];
```

Optimize updates

UPDATE is a fully logged operation. If you need to update a large number of rows in a table or a partition, it can often be far more efficient to use a minimally logged operation such as CTAS to do so.

In the example below a full table update has been converted to a CTAS so that minimal logging is possible.

In this case, we are retrospectively adding a discount amount to the sales in the table:

```
--Step 01. Create a new table containing the "Update".
CREATE TABLE [dbo].[FactInternetSales_u]
WITH
(    CLUSTERED INDEX
,    DISTRIBUTION = HASH([ProductKey])
,     PARTITION     (    [OrderDateKey] RANGE RIGHT
                                    FOR VALUES    (    20000101, 20010101, 20020101, 20030101, 20040101,
20050101
                                                ,    20060101, 20070101, 20080101, 20090101, 20100101,
20110101
                                                ,    20120101, 20130101, 20140101, 20150101, 20160101,
20170101
                                                ,    20180101, 20190101, 20200101, 20210101, 20220101,
20230101
                                                ,    20240101, 20250101, 20260101, 20270101, 20280101,
20290101
                                                )
                )
)
AS
SELECT
    [ProductKey]
,    [OrderDateKey]
,    [DueDateKey]
,    [ShipDateKey]
,    [CustomerKey]
,    [PromotionKey]
,    [CurrencyKey]
,    [SalesTerritoryKey]
,    [SalesOrderNumber]
,    [SalesOrderLineNumber]
,    [RevisionNumber]
,    [OrderQuantity]
,    [UnitPrice]
,    [ExtendedAmount]
,    [UnitPriceDiscountPct]
,    ISNULL(CAST(5 as float),0) AS [DiscountAmount]
,    [ProductStandardCost]
,    [TotalProductCost]
,    ISNULL(CAST(CASE WHEN [SalesAmount] <=5 THEN 0
         ELSE [SalesAmount] - 5
         END AS MONEY),0) AS [SalesAmount]
,    [TaxAmt]
,    [Freight]
,    [CarrierTrackingNumber]
,    [CustomerPONumber]
FROM    [dbo].[FactInternetSales]
OPTION (LABEL = 'CTAS : Update')
;

--Step 02. Rename the tables
RENAME OBJECT [dbo].[FactInternetSales]   TO [FactInternetSales_old];
RENAME OBJECT [dbo].[FactInternetSales_u] TO [FactInternetSales];

--Step 03. Drop the old table
DROP TABLE [dbo].[FactInternetSales_old]
```

NOTE

Re-creating large tables can benefit from using dedicated SQL pool workload management features. For more information, see Resource classes for workload management.

Optimize with partition switching

If faced with large-scale modifications inside a table partition, then a partition switching pattern makes sense. If the data modification is significant and spans multiple partitions, then iterating over the partitions achieves the same result.

The steps to perform a partition switch are as follows:

1. Create an empty out partition
2. Perform the 'update' as a CTAS
3. Switch out the existing data to the out table
4. Switch in the new data
5. Clean up the data

However, to help identify the partitions to switch, create the following helper procedure.

```
CREATE PROCEDURE dbo.partition_data_get
    @schema_name           NVARCHAR(128)
,    @table_name               NVARCHAR(128)
,    @boundary_value           INT
AS
IF OBJECT_ID('tempdb..#ptn_data') IS NOT NULL
BEGIN
    DROP TABLE #ptn_data
END
CREATE TABLE #ptn_data
WITH    (    DISTRIBUTION = ROUND_ROBIN
        ,    HEAP
        )
AS
WITH CTE
AS
(
SELECT     s.name                            AS [schema_name]
,        t.name                              AS [table_name]
,         p.partition_number                 AS [ptn_nmbr]
,        p.[rows]                          AS [ptn_rows]
,        CAST(r.[value] AS INT)              AS [boundary_value]
FROM        sys.schemas                      AS s
JOIN        sys.tables                      AS t    ON  s.[schema_id]        = t.[schema_id]
JOIN        sys.indexes                      AS i    ON     t.[object_id]        = i.[object_id]
JOIN        sys.partitions                  AS p    ON     i.[object_id]        = p.[object_id]
                                                AND i.[index_id]        = p.[index_id]
JOIN        sys.partition_schemes        AS h    ON     i.[data_space_id]    = h.[data_space_id]
JOIN        sys.partition_functions        AS f    ON     h.[function_id]        = f.[function_id]
LEFT JOIN    sys.partition_range_values    AS r     ON     f.[function_id]        = r.[function_id]
                                                AND r.[boundary_id]        = p.[partition_number]
WHERE i.[index_id] <= 1
)
SELECT    *
FROM    CTE
WHERE    [schema_name]        = @schema_name
AND        [table_name]        = @table_name
AND        [boundary_value]    = @boundary_value
OPTION (LABEL = 'dbo.partition_data_get : CTAS : #ptn_data')
;
GO
```

This procedure maximizes code reuse and keeps the partition switching example more compact.

The following code demonstrates the steps mentioned previously to achieve a full partition switching routine.

```
--Create a partitioned aligned empty table to switch out the data
IF OBJECT_ID('[dbo].[FactInternetSales_out]') IS NOT NULL
BEGIN
```

```
    DROP TABLE [dbo].[FactInternetSales_out]
END

CREATE TABLE [dbo].[FactInternetSales_out]
WITH
(    DISTRIBUTION = HASH([ProductKey])
,    CLUSTERED COLUMNSTORE INDEX
,     PARTITION     (    [OrderDateKey] RANGE RIGHT
                                    FOR VALUES    (    20020101, 20030101
                                                )
                )
)
AS
SELECT *
FROM    [dbo].[FactInternetSales]
WHERE 1=2
OPTION (LABEL = 'CTAS : Partition Switch IN : UPDATE')
;

--Create a partitioned aligned table and update the data in the select portion of the CTAS
IF OBJECT_ID('[dbo].[FactInternetSales_in]') IS NOT NULL
BEGIN
    DROP TABLE [dbo].[FactInternetSales_in]
END

CREATE TABLE [dbo].[FactInternetSales_in]
WITH
(    DISTRIBUTION = HASH([ProductKey])
,    CLUSTERED COLUMNSTORE INDEX
,     PARTITION     (    [OrderDateKey] RANGE RIGHT
                                    FOR VALUES    (    20020101, 20030101
                                                )
                )
)
AS
SELECT
    [ProductKey]
,    [OrderDateKey]
,    [DueDateKey]
,    [ShipDateKey]
,    [CustomerKey]
,    [PromotionKey]
,    [CurrencyKey]
,    [SalesTerritoryKey]
,    [SalesOrderNumber]
,    [SalesOrderLineNumber]
,    [RevisionNumber]
,    [OrderQuantity]
,    [UnitPrice]
,    [ExtendedAmount]
,    [UnitPriceDiscountPct]
,    ISNULL(CAST(5 as float),0) AS [DiscountAmount]
,    [ProductStandardCost]
,    [TotalProductCost]
,    ISNULL(CAST(CASE WHEN [SalesAmount] <=5 THEN 0
         ELSE [SalesAmount] - 5
         END AS MONEY),0) AS [SalesAmount]
,    [TaxAmt]
,    [Freight]
,    [CarrierTrackingNumber]
,    [CustomerPONumber]
FROM    [dbo].[FactInternetSales]
WHERE    OrderDateKey BETWEEN 20020101 AND 20021231
OPTION (LABEL = 'CTAS : Partition Switch IN : UPDATE')
;

--Use the helper procedure to identify the partitions
--The source table
EXEC dbo.partition_data_get 'dbo','FactInternetSales',20030101
```

```
DECLARE @ptn_nmbr_src INT = (SELECT ptn_nmbr FROM #ptn_data)
SELECT @ptn_nmbr_src

--The "in" table
EXEC dbo.partition_data_get 'dbo','FactInternetSales_in',20030101
DECLARE @ptn_nmbr_in INT = (SELECT ptn_nmbr FROM #ptn_data)
SELECT @ptn_nmbr_in

--The "out" table
EXEC dbo.partition_data_get 'dbo','FactInternetSales_out',20030101
DECLARE @ptn_nmbr_out INT = (SELECT ptn_nmbr FROM #ptn_data)
SELECT @ptn_nmbr_out

--Switch the partitions over
DECLARE @SQL NVARCHAR(4000) = '
ALTER TABLE [dbo].[FactInternetSales]    SWITCH PARTITION '+CAST(@ptn_nmbr_src AS VARCHAR(20))    +' TO
[dbo].[FactInternetSales_out] PARTITION '    +CAST(@ptn_nmbr_out AS VARCHAR(20))+';
ALTER TABLE [dbo].[FactInternetSales_in] SWITCH PARTITION '+CAST(@ptn_nmbr_in AS VARCHAR(20))    +' TO
[dbo].[FactInternetSales] PARTITION '        +CAST(@ptn_nmbr_src AS VARCHAR(20))+';'
EXEC sp_executesql @SQL

--Perform the clean-up
TRUNCATE TABLE dbo.FactInternetSales_out;
TRUNCATE TABLE dbo.FactInternetSales_in;

DROP TABLE dbo.FactInternetSales_out
DROP TABLE dbo.FactInternetSales_in
DROP TABLE #ptn_data
```

Minimize logging with small batches

For large data modification operations, it may make sense to divide the operation into chunks or batches to scope the unit of work.

A following code is a working example. The batch size has been set to a trivial number to highlight the technique. In reality, the batch size would be significantly larger.

```
SET NO_COUNT ON;
IF OBJECT_ID('tempdb..#t') IS NOT NULL
BEGIN
    DROP TABLE #t;
    PRINT '#t dropped';
END

CREATE TABLE #t
WITH    (    DISTRIBUTION = ROUND_ROBIN
        ,    HEAP
        )
AS
SELECT    ROW_NUMBER() OVER(ORDER BY (SELECT NULL)) AS seq_nmbr
,        SalesOrderNumber
,        SalesOrderLineNumber
FROM    dbo.FactInternetSales
WHERE    [OrderDateKey] BETWEEN 20010101 and 20011231
;

DECLARE    @seq_start        INT = 1
,        @batch_iterator    INT = 1
,        @batch_size        INT = 50
,        @max_seq_nmbr    INT = (SELECT MAX(seq_nmbr) FROM dbo.#t)
;

DECLARE    @batch_count    INT = (SELECT CEILING((@max_seq_nmbr*1.0)/@batch_size))
,        @seq_end        INT = @batch_size
;

SELECT COUNT(*)
FROM    dbo.FactInternetSales f

PRINT 'MAX_seq_nmbr '+CAST(@max_seq_nmbr AS VARCHAR(20))
PRINT 'MAX_Batch_count '+CAST(@batch_count AS VARCHAR(20))

WHILE    @batch_iterator <= @batch_count
BEGIN
    DELETE
    FROM    dbo.FactInternetSales
    WHERE EXISTS
    (
            SELECT    1
            FROM    #t t
            WHERE    seq_nmbr BETWEEN  @seq_start AND @seq_end
            AND        FactInternetSales.SalesOrderNumber        = t.SalesOrderNumber
            AND        FactInternetSales.SalesOrderLineNumber    = t.SalesOrderLineNumber
    )
    ;

    SET @seq_start = @seq_end
    SET @seq_end = (@seq_start+@batch_size);
    SET @batch_iterator +=1;
END
```

Pause and scaling guidance

Azure Synapse Analytics lets you pause, resume, and scale your dedicated SQL pool on demand.

When you pause or scale your dedicated SQL pool, it is important to understand that any in-flight transactions are terminated immediately; causing any open transactions to be rolled back.

If your workload had issued a long running and incomplete data modification prior to the pause or scale operation, then this work will need to be undone. This undoing might impact the time it takes to pause or scale your dedicated SQL pool.

> **IMPORTANT**
> Both `UPDATE` and `DELETE` are fully logged operations and so these undo/redo operations can take significantly longer than equivalent minimally logged operations.

The best scenario is to let in flight data modification transactions complete prior to pausing or scaling your dedicated SQL pool. However, this scenario might not always be practical. To mitigate the risk of a long rollback, consider one of the following options:

- Rewrite long running operations using CTAS
- Break the operation into chunks; operating on a subset of the rows

Next steps

See Transactions in dedicated SQL pool to learn more about isolation levels and transactional limits. For an overview of other best practices, see Dedicated SQL pool best practices.

User-defined schemas within Synapse SQL

8/19/2022 • 3 minutes to read • Edit Online

In the sections below, you'll find various tips for using T-SQL user-defined schemas to develop solutions within Synapse SQL.

Schemas for application boundaries

Traditional analytics architecture often uses separate databases to create application boundaries based on workload, domain, or security. For example, a traditional SQL Server analytics infrastructure might include a staging database, an analytics database, and data mart databases. In this topology, each database operates as a workload and security boundary in the architecture.

Instead, Synapse SQL runs the entire analytics workload within one database. Cross database joins aren't permitted. Synapse SQL expects all tables used by the warehouse to be stored within the one database.

> **NOTE**
> Dedicated SQL pools do not support cross database queries of any kind. Consequently, analytics implementations that leverage this pattern will need to be revised. Serverless SQL pool supports cross database queries.

User-defined schema recommendations

Included are recommendations for consolidating workloads, security, domain, and functional boundaries by using user-defined schemas:

- Use one database to run your entire analytics workload.
- Consolidate your existing analytics environment to use one database.
- Leverage **user-defined schemas** to provide the boundary previously implemented using databases.

If user-defined schemas haven't been used previously, then you have a clean slate. Use the old database name as the basis for your user-defined schemas in the Synapse SQL database.

If schemas have already been used, then you have a few options:

- Remove the legacy schema names and start fresh
- Keep the legacy schema names by pre-pending the legacy schema name to the table name
- Retain the legacy schema names by implementing views over the table in an extra schema, which re-creates the old schema structure.

> **NOTE**
> On first inspection, option 3 may seem like the most appealing choice. Views are read only in Synapse SQL. Any data or table modification would need to be performed against the base table. Option 3 also introduces a layer of views into your system. You might want to give this some additional thought if you are already using views in your architecture.

Examples

Implement user-defined schemas based on database names.

```
CREATE SCHEMA [stg]; -- stg previously database name for staging database
GO
CREATE SCHEMA [edw]; -- edw previously database name for the analytics
GO
CREATE TABLE [stg].[customer] -- create staging tables in the stg schema
(       CustKey BIGINT NOT NULL
,       ...
);
GO
CREATE TABLE [edw].[customer] -- create analytics tables in the edw schema
(       CustKey BIGINT NOT NULL
,       ...
);
```

Keep the legacy schema names by pre-pending them to the table name. Use schemas for the workload boundary.

```
CREATE SCHEMA [stg]; -- stg defines the staging boundary
GO
CREATE SCHEMA [edw]; -- edw defines the analytics boundary
GO
CREATE TABLE [stg].[dim_customer] --pre-pend the old schema name to the table and create in the staging
boundary
(       CustKey BIGINT NOT NULL
,       ...
);
GO
CREATE TABLE [edw].[dim_customer] --pre-pend the old schema name to the table and create in the analytics
boundary
(       CustKey BIGINT NOT NULL
,       ...
);
```

Retain the legacy schema names using views.

```
CREATE SCHEMA [stg]; -- stg defines the staging boundary
GO
CREATE SCHEMA [edw]; -- stg defines the analytics boundary
GO
CREATE SCHEMA [dim]; -- edw defines the legacy schema name boundary
GO
CREATE TABLE [stg].[customer] -- create the base staging tables in the staging boundary
(       CustKey    BIGINT NOT NULL
,       ...
)
GO
CREATE TABLE [edw].[customer] -- create the base analytics tables in the analytics boundary
(       CustKey    BIGINT NOT NULL
,       ...
)
GO
CREATE VIEW [dim].[customer] -- create a view in the legacy schema name boundary for presentation
consistency purposes only
AS
SELECT  CustKey
,       ...
FROM    [edw].customer
;
```

> **NOTE**
>
> Any change in schema strategy requires a review of the security model for the database. In many cases, you might be able to simplify the security model by assigning permissions at the schema level.

If more granular permissions are required, you can use database roles. For more information about database roles, see the Manage database roles and users article.

Next steps

For more development tips, see Synapse SQL development overview.

Assign variables with Synapse SQL

8/19/2022 • 2 minutes to read • Edit Online

In this article, you'll find tips for assigning T-SQL variables with Synapse SQL.

Set variables with DECLARE

Variables in Synapse SQL are set using the `DECLARE` statement or the `SET` statement. Initializing variables with DECLARE is one of the most flexible ways to set a variable value in Synapse SQL.

```
DECLARE @v  int = 0
;
```

You can also use DECLARE to set more than one variable at a time. You can't use SELECT or UPDATE to do the following:

```
DECLARE @v  INT = (SELECT TOP 1 c_customer_sk FROM Customer where c_last_name = 'Smith')
,       @v1 INT = (SELECT TOP 1 c_customer_sk FROM Customer where c_last_name = 'Jones')
;
```

You can't initialize and use a variable in the same DECLARE statement. To illustrate, the following example isn't allowed since *@p1* is both initialized and used in the same DECLARE statement. The following example gives an error.

```
DECLARE @p1 int = 0
,       @p2 int = (SELECT COUNT (*) FROM sys.types where is_user_defined = @p1 )
;
```

Set values with SET

SET is a common method for setting a single variable.

The following statements are all valid ways to set a variable with SET:

```
SET     @v = (Select max(database_id) from sys.databases);
SET     @v = 1;
SET     @v = @v+1;
SET     @v +=1;
```

You can only set one variable at a time with SET. However, compound operators are permissible.

Limitations

You can't use UPDATE for variable assignment.

Next steps

For more development tips, see the Synapse SQL development overview article.

T-SQL views with dedicated SQL pool and serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

In this article, you'll find tips for using T-SQL views and developing solutions with dedicated SQL pool and serverless SQL pool in Azure Synapse Analytics.

Why use views

Views can be used in a number of different ways to improve the quality of your solution. This article highlights a few examples of how to enrich your solution with views and includes the limitations that need to be considered.

SQL pool - create view

> **NOTE**
> Syntax for CREATE VIEW is not discussed in this article. For more information, see the CREATE VIEW documentation.

Architectural abstraction

A common application pattern is to re-create tables using CREATE TABLE AS SELECT (CTAS), which is followed by an object renaming pattern while loading data.

The following example adds new date records to a date dimension. Note how a new table, DimDate_New, is first created and then renamed to replace the original version of the table.

```
CREATE TABLE dbo.DimDate_New
WITH (DISTRIBUTION = ROUND_ROBIN
, CLUSTERED INDEX (DateKey ASC)
)
AS
SELECT *
FROM   dbo.DimDate  AS prod
UNION ALL
SELECT *
FROM   dbo.DimDate_stg AS stg
;

RENAME OBJECT DimDate TO DimDate_Old;
RENAME OBJECT DimDate_New TO DimDate;
```

Keep in mind that this approach can result in tables appearing and disappearing from a user's view, and prompts "table does not exist" error messages. Views can be used to provide users with a consistent presentation layer while the underlying objects are renamed.

By providing access to data through views, users don't need visibility to the underlying tables. In addition to a consistent user experience, this layer ensures that analytics designers can evolve the data model. The ability to evolve the underlying tables means designers can use CTAS to maximize performance during the data loading process.

Performance optimization

Views can also be used to enforce performance optimized joins between tables. For example, a view can incorporate a redundant distribution key as part of the joining criteria to minimize data movement.

Forcing a specific query or joining hint is another benefit of using T-SQL views. As such, the views capability ensures that joins are always performed in an optimal fashion. You'll avoid the need for users to remember the correct construct for their joins.

Limitations

Views in Synapse SQL are only stored as metadata. Consequently, the following options aren't available:

- There isn't a schema binding option
- Base tables can't be updated through the view
- Views can't be created over temporary tables
- There's no support for the EXPAND / NOEXPAND hints
- There are no indexed views in Synapse SQL

Next steps

For more development tips, see Synapse SQL development overview.

Dynamic SQL in Synapse SQL

8/19/2022 • 2 minutes to read • Edit Online

In this article, you'll find tips for using dynamic SQL and developing solutions using Synapse SQL.

Dynamic SQL Example

When developing application code, you may need to use dynamic SQL to help deliver flexible, generic, and modular solutions.

> **NOTE**
>
> Dedicated SQL pool does not support blob data types at this time. Not supporting blob data types might limit the size of your strings since blob data types include both varchar(max) and nvarchar(max) types. If you have used these types in your application code to build large strings, you need to break the code into chunks and use the EXEC statement instead.

A simple example:

```
DECLARE @sql_fragment1 VARCHAR(8000)=' SELECT name '
,       @sql_fragment2 VARCHAR(8000)=' FROM sys.system_views '
,       @sql_fragment3 VARCHAR(8000)=' WHERE name like ''%table%''';

EXEC( @sql_fragment1 + @sql_fragment2 + @sql_fragment3);
```

If the string is short, you can use sp_executesql as normal.

> **NOTE**
>
> Statements executed as dynamic SQL will still be subject to all T-SQL validation rules.

Next steps

For more development tips, see development overview.

GROUP BY options in Synapse SQL

8/19/2022 • 3 minutes to read • Edit Online

Synapse SQL allows for developing solutions by implementing different GROUP BY options.

What GROUP BY does

The GROUP BY T-SQL clause aggregates data to a summary set of rows.

Serverless SQL pool doesn't support GROUP BY options. Dedicated SQL pool supports a limited number of GROUP BY options.

GROUP BY options supported in dedicated SQL pool

GROUP BY has some options that dedicated SQL pool doesn't support. These options have workarounds, which are as follows:

- GROUP BY with ROLLUP
- GROUPING SETS
- GROUP BY with CUBE

Rollup and grouping sets options

The simplest option here is to use UNION ALL to execute the rollup rather than relying on the explicit syntax. The result is exactly the same

The following example uses the GROUP BY statement with the ROLLUP option:

```
SELECT [SalesTerritoryCountry]
,      [SalesTerritoryRegion]
,      SUM(SalesAmount)             AS TotalSalesAmount
FROM  dbo.factInternetSales s
JOIN  dbo.DimSalesTerritory t       ON s.SalesTerritoryKey       = t.SalesTerritoryKey
GROUP BY ROLLUP (
                        [SalesTerritoryCountry]
                ,       [SalesTerritoryRegion]
                )
;
```

By using ROLLUP, the preceding example requests the following aggregations:

- Country and Region
- Country
- Grand Total

To replace ROLLUP and return the same results, you can use UNION ALL and explicitly specify the required aggregations:

```
SELECT [SalesTerritoryCountry]
,      [SalesTerritoryRegion]
,      SUM(SalesAmount) AS TotalSalesAmount
FROM  dbo.factInternetSales s
JOIN  dbo.DimSalesTerritory t     ON s.SalesTerritoryKey       = t.SalesTerritoryKey
GROUP BY
       [SalesTerritoryCountry]
,      [SalesTerritoryRegion]
UNION ALL
SELECT [SalesTerritoryCountry]
,      NULL
,      SUM(SalesAmount) AS TotalSalesAmount
FROM  dbo.factInternetSales s
JOIN  dbo.DimSalesTerritory t     ON s.SalesTerritoryKey       = t.SalesTerritoryKey
GROUP BY
       [SalesTerritoryCountry]
UNION ALL
SELECT NULL
,      NULL
,      SUM(SalesAmount) AS TotalSalesAmount
FROM  dbo.factInternetSales s
JOIN  dbo.DimSalesTerritory t     ON s.SalesTerritoryKey       = t.SalesTerritoryKey;
```

To replace GROUPING SETS, the sample principle applies. You only need to create UNION ALL sections for the aggregation levels you want to see.

Cube options

It's possible to create a GROUP BY WITH CUBE using the UNION ALL approach. The problem is that the code can quickly become cumbersome and unwieldy. To mitigate this issue, you can use this more advanced approach.

The first step is to define the 'cube' that defines all the levels of aggregation that we want to create. Take note of the CROSS JOIN of the two derived tables as it generates all levels. The rest of the code is there for formatting.

```
CREATE TABLE #Cube
WITH
(   DISTRIBUTION = ROUND_ROBIN
,   LOCATION = USER_DB
)
AS
WITH GrpCube AS
(SELECT    CAST(ISNULL(Country,'NULL')+','+ISNULL(Region,'NULL') AS NVARCHAR(50)) as 'Cols'
,          CAST(ISNULL(Country+',','')+ISNULL(Region,'') AS NVARCHAR(50))  as 'GroupBy'
,          ROW_NUMBER() OVER (ORDER BY Country) as 'Seq'
FROM       ( SELECT 'SalesTerritoryCountry' as Country
             UNION ALL
             SELECT NULL
           ) c
CROSS JOIN ( SELECT 'SalesTerritoryRegion' as Region
             UNION ALL
             SELECT NULL
           ) r
)
SELECT Cols
,      CASE WHEN SUBSTRING(GroupBy,LEN(GroupBy),1) = ','
            THEN SUBSTRING(GroupBy,1,LEN(GroupBy)-1)
            ELSE GroupBy
       END AS GroupBy  --Remove Trailing Comma
,Seq
FROM GrpCube;
```

The following image shows the results of CREATE TABLE AS SELECT:

	Cols	GroupBy	Seq
1	NULL,SalesTerritoryRegion	SalesTerritoryRegion	1
2	NULL,NULL		2
3	SalesTerritoryCountry,SalesTerritoryRegion	SalesTerritoryCountry,SalesTerritoryRegion	3
4	SalesTerritoryCountry,NULL	SalesTerritoryCountry	4

The second step is to specify a target table for storing interim results:

```
DECLARE
 @SQL NVARCHAR(4000)
,@Columns NVARCHAR(4000)
,@GroupBy NVARCHAR(4000)
,@i INT = 1
,@nbr INT = 0
;
CREATE TABLE #Results
(
 [SalesTerritoryCountry] NVARCHAR(50)
,[SalesTerritoryRegion]  NVARCHAR(50)
,[TotalSalesAmount]      MONEY
)
WITH
(   DISTRIBUTION = ROUND_ROBIN
,   LOCATION = USER_DB
)
;
```

The third step is to loop over the cube of columns performing the aggregation. The query will run once for every row in the #Cube temporary table. Results are stored in the #Results temp table:

```
SET @nbr =(SELECT MAX(Seq) FROM #Cube);

WHILE @i<=@nbr
BEGIN
    SET @Columns = (SELECT Cols    FROM #Cube where seq = @i);
    SET @GroupBy = (SELECT GroupBy FROM #Cube where seq = @i);

    SET @SQL ='INSERT INTO #Results
               SELECT '+@Columns+'
               ,      SUM(SalesAmount) AS TotalSalesAmount
               FROM  dbo.factInternetSales s
               JOIN  dbo.DimSalesTerritory t
               ON s.SalesTerritoryKey = t.SalesTerritoryKey
               '+CASE WHEN @GroupBy <>''
                      THEN 'GROUP BY '+@GroupBy ELSE '' END

    EXEC sp_executesql @SQL;
    SET @i +=1;
END
```

Lastly, you can return the results by reading from the #Results temporary table:

```
SELECT *
FROM #Results
ORDER BY 1,2,3
;
```

By breaking up the code into sections and generating a looping construct, the code becomes more manageable and maintainable.

Next steps

For more development tips, see development overview.

Use query labels in Synapse SQL

8/19/2022 • 2 minutes to read • Edit Online

Included in this article are essential tips for using query labels in Synapse SQL.

> **NOTE**
> Serverless SQL pool doesn't support labelling queries.

What are query labels

Dedicated SQL pool supports a concept called query labels. Before going into any depth, let's look at an example:

```
SELECT *
FROM sys.tables
OPTION (LABEL = 'My Query Label')
;
```

The last line tags the string 'My Query Label' to the query. This tag is helpful since the label is query-able through the DMVs. Querying for labels provides a mechanism for locating problem queries and helps to identify progress through an ELT run.

Good naming conventions are most helpful. For example, starting the label with PROJECT, PROCEDURE, STATEMENT, or COMMENT uniquely identifies the query among all the code in source control.

The following query uses a dynamic management view to search by label:

```
SELECT  *
FROM    sys.dm_pdw_exec_requests r
WHERE   r.[label] = 'My Query Label'
;
```

> **NOTE**
> It is essential to put square brackets or double quotes around the word label when querying. Label is a reserved word and causes an error when it is not delimited.

Next steps

For more development tips, see development overview.

Access external storage using serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

This article describes how users can read data from the files stored on Azure Storage in serverless SQL pool. Users have the following options to access storage:

- OPENROWSET function that enables ad-hoc queries over the files in Azure Storage.
- External table that is a predefined data structure built on top of set of external files.

User can use different authentication methods such as Azure AD passthrough authentication (default for Azure AD principals) and SAS authentication (default for SQL principals).

Query files using OPENROWSET

OPENROWSET enables users to query external files on Azure storage if they have access to the storage. A user who is connected to serverless SQL pool should use the following query to read the content of the files on Azure storage:

```
SELECT * FROM
 OPENROWSET(BULK 'https://<storage_account>.dfs.core.windows.net/<container>/<path>/*.parquet', format=
'parquet') as rows
```

User can access storage using the following access rules:

- Azure AD user - `OPENROWSET` will use Azure AD identity of caller to access Azure Storage or access storage with anonymous access.
- SQL user – `OPENROWSET` will access storage with anonymous access or can be impersonated using SAS token or Managed identity of workspace.

- Impersonation
- Direct access

SQL principals can also use OPENROWSET to directly query files protected with SAS tokens or Managed Identity of the workspace. If a SQL user executes this function, a power user with `ALTER ANY CREDENTIAL` permission must create a server-scoped credential that matches URL in the function (using storage name and container) and granted REFERENCES permission for this credential to the caller of OPENROWSET function:

```
EXECUTE AS somepoweruser

CREATE CREDENTIAL [https://<storage_account>.dfs.core.windows.net/<container>]
 WITH IDENTITY = 'SHARED ACCESS SIGNATURE', SECRET = 'sas token';

GRANT REFERENCES ON CREDENTIAL::[https://<storage_account>.dfs.core.windows.net/<container>] TO sqluser
```

If there's no server-level CREDENTIAL that matches the URL, or the SQL user doesn't have references permission for this credential, the error will be returned. SQL principals can't impersonate using some Azure AD identity.

> **NOTE**
> This version of OPENROWSET is designed for quick-and-easy data exploration using default authentication. To leverage impersonation or Managed Identity, use OPENROWSET with DATA_SOURCE described in the next section.

Query data sources using OPENROWSET

OPENROWSET enables user to query the files placed on some external data source:

```
SELECT * FROM
 OPENROWSET(BULK 'file/path/*.parquet',
 DATA_SOURCE = MyAzureInvoices,
 FORMAT= 'parquet') as rows
```

The user that executes this query must be able to access the files. The users must be impersonated using SAS token or Managed Identity of workspace if they can't directly access the files using their Azure AD identity or anonymous access.

- Impersonation
- Direct access

`DATABASE SCOPED CREDENTIAL` specifies how to access files on the referenced data source (currently SAS and Managed Identity). Power user with `CONTROL DATABASE` permission would need to create `DATABASE SCOPED CREDENTIAL` that will be used to access storage and `EXTERNAL DATA SOURCE` that specifies URL of data source and credential that should be used:

```
EXECUTE AS somepoweruser;

-- Create MASTER KEY if it doesn't exists in database
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'some very strong password';

CREATE DATABASE SCOPED CREDENTIAL AccessAzureInvoices
 WITH IDENTITY = 'SHARED ACCESS SIGNATURE',
 SECRET = '******srt=sco&amp;sp=rwac&amp;se=2017-02-01T00:55:34Z&amp;st=201********' ;

CREATE EXTERNAL DATA SOURCE MyAzureInvoices
 WITH ( LOCATION = 'https://<storage_account>.dfs.core.windows.net/<container>/<path>/' ,
 CREDENTIAL = AccessAzureInvoices) ;
```

Caller must have one of the following permissions to execute OPENROWSET function:

- One of the permissions to execute OPENROWSET:
  - `ADMINISTER BULK OPERATIONS` enables login to execute OPENROWSET function.
  - `ADMINISTER DATABASE BULK OPERATIONS` enables database scoped user to execute OPENROWSET function.
- `REFERENCES DATABASE SCOPED CREDENTIAL` to the credential that is referenced in `EXTERNAL DATA SOURCE`.

EXTERNAL TABLE

User with the permissions to read table can access external files using an EXTERNAL TABLE created on top of set of Azure Storage folders and files.

User that has permissions to create external table (for example CREATE TABLE and ALTER ANY CREDENTIAL or REFERENCES DATABASE SCOPED CREDENTIAL) can use the following script to create a table on top of Azure Storage data source:

```
CREATE EXTERNAL TABLE [dbo].[DimProductexternal]
( ProductKey int, ProductLabel nvarchar, ProductName nvarchar )
WITH
(
LOCATION='/DimProduct/year=*/month=*' ,
DATA_SOURCE = AzureDataLakeStore ,
FILE_FORMAT = TextFileFormat
) ;
```

User that reads data from this table must be able to access the files. The users must be impersonated using SAS token or Managed Identity of workspace if they cannot directly access the files using their Azure AD identity or anonymous access.

- Impersonation
- Direct access

DATABASE SCOPED CREDENTIAL specifies how to access files on the referenced data source. User with CONTROL DATABASE permission would need to create DATABASE SCOPED CREDENTIAL that will be used to access storage and EXTERNAL DATA SOURCE that specifies URL of data source and credential that should be used:

```
EXECUTE AS somepoweruser;

-- Create MASTER KEY if it doesn't exists in database
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'some very strong password';

CREATE DATABASE SCOPED CREDENTIAL cred
 WITH IDENTITY = 'SHARED ACCESS SIGNATURE',
 SECRET = '******srt=sco&sp=rwac&se=2017-02-01T00:55:34Z&st=201********' ;

CREATE EXTERNAL DATA SOURCE AzureDataLakeStore
 WITH ( LOCATION = 'https://<storage_account>.dfs.core.windows.net/<container>/<path>' ,
 CREDENTIAL = cred
 ) ;
```

Read external files with EXTERNAL TABLE

EXTERNAL TABLE enables you to read data from the files that are referenced via data source using standard SQL SELECT statement:

```
SELECT *
FROM dbo.DimProductsExternal
```

Caller must have the following permissions to read data:

- `SELECT` permission ON external table
- `REFERENCES DATABASE SCOPED CREDENTIAL` permission if `DATA SOURCE` has `CREDENTIAL`

Permissions

The following table lists required permissions for the operations listed above.

QUERY	REQUIRED PERMISSIONS

QUERY	REQUIRED PERMISSIONS
OPENROWSET(BULK) without datasource	`ADMINISTER BULK OPERATIONS`, `ADMINISTER DATABASE BULK OPERATIONS`, or SQL login must have REFERENCES CREDENTIAL::<URL> for SAS-protected storage
OPENROWSET(BULK) with datasource without credential	`ADMINISTER BULK OPERATIONS` or `ADMINISTER DATABASE BULK OPERATIONS`,
OPENROWSET(BULK) with datasource with credential	`REFERENCES DATABASE SCOPED CREDENTIAL` and one of `ADMINISTER BULK OPERATIONS` or `ADMINISTER DATABASE BULK OPERATIONS`
CREATE EXTERNAL DATA SOURCE	`ALTER ANY EXTERNAL DATA SOURCE` and `REFERENCES DATABASE SCOPED CREDENTIAL`
CREATE EXTERNAL TABLE	`CREATE TABLE`, `ALTER ANY SCHEMA`, `ALTER ANY EXTERNAL FILE FORMAT`, and `ALTER ANY EXTERNAL DATA SOURCE`
SELECT FROM EXTERNAL TABLE	`SELECT TABLE` and `REFERENCES DATABASE SCOPED CREDENTIAL`
CETAS	To create table - `CREATE TABLE`, `ALTER ANY SCHEMA`, `ALTER ANY DATA SOURCE`, and `ALTER ANY EXTERNAL FILE FORMAT`. To read data: `ADMINISTER BULK OPERATIONS` or `REFERENCES CREDENTIAL` or `SELECT TABLE` per each table/view/function in query + R/W permission on storage

Next steps

You're now ready to continue on with the following How To articles:

- Query data on storage
- Query CSV file
- Query Parquet files
- Query JSON files
- Query folders and multiple files
- Use partitioning and metadata functions
- Query nested types
- Creating and using views

Control storage account access for serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 16 minutes to read • Edit Online

A serverless SQL pool query reads files directly from Azure Storage. Permissions to access the files on Azure storage are controlled at two levels:

- **Storage level** - User should have permission to access underlying storage files. Your storage administrator should allow Azure AD principal to read/write files, or generate SAS key that will be used to access storage.
- **SQL service level** - User should have granted permission to read data using external table or to execute the `OPENROWSET` function. Read more about the required permissions in this section.

This article describes the types of credentials you can use and how credential lookup is enacted for SQL and Azure AD users.

Storage permissions

A serverless SQL pool in Synapse Analytics workspace can read the content of files stored in Azure Data Lake storage. You need to configure permissions on storage to enable a user who executes a SQL query to read the files. There are three methods for enabling the access to the files:

- **Role based access control (RBAC)** enables you to assign a role to some Azure AD user in the tenant where your storage is placed. A reader must have `Storage Blob Data Reader`, `Storage Blob Data Contributor`, or `Storage Blob Data Owner` RBAC role on storage account. A user who writes data in the Azure storage must have `Storage Blob Data Contributor` or `Storage Blob Data Owner` role. Note that `Storage Owner` role does not imply that a user is also `Storage Data Owner`.
- **Access Control Lists (ACL)** enable you to define a fine grained Read(R), Write(W), and Execute(X) permissions on the files and directories in Azure storage. ACL can be assigned to Azure AD users. If readers want to read a file on a path in Azure Storage, they must have Execute(X) ACL on every folder in the file path, and Read(R) ACL on the file. Learn more how to set ACL permissions in storage layer.
- **Shared access signature (SAS)** enables a reader to access the files on the Azure Data Lake storage using the time-limited token. The reader doesn't even need to be authenticated as Azure AD user. SAS token contains the permissions granted to the reader as well as the period when the token is valid. SAS token is good choice for time-constrained access to any user that doesn't even need to be in the same Azure AD tenant. SAS token can be defined on the storage account or on specific directories. Learn more about granting limited access to Azure Storage resources using shared access signatures.

As an alternative, you can make your files publicly available by allowing anonymous access. This approach should NOT be used if you have non-public data.

Supported storage authorization types

A user that has logged into a serverless SQL pool must be authorized to access and query the files in Azure Storage if the files aren't publicly available. You can use four authorization types to access non-public storage - User Identity, Shared access signature, Service Principal and Managed Identity.

> **NOTE**
> **Azure AD pass-through** is the default behavior when you create a workspace.

- User Identity
- Shared access signature
- Service Principal
- Managed Identity
- Anonymous access

User Identity, also known as "Azure AD pass-through", is an authorization type where the identity of the Azure AD user that logged into serverless SQL pool is used to authorize data access. Before accessing the data, the Azure Storage administrator must grant permissions to the Azure AD user. As indicated in the table below, it's not supported for the SQL user type.

> **IMPORTANT**
> AAD authentication token might be cached by the client applications. For example PowerBI caches AAD token and reuses the same token for an hour. The long running queries might fail if the token expires in the middle of the query execution. If you are experiencing query failures caused by the AAD access token that expires in the middle of the query, consider switching to Service Principal, Managed identity or Shared access signature.

You need to have a Storage Blob Data Owner/Contributor/Reader role to use your identity to access the data. As an alternative, you can specify fine-grained ACL rules to access files and folders. Even if you are an Owner of a Storage Account, you still need to add yourself into one of the Storage Blob Data roles. To learn more about access control in Azure Data Lake Store Gen2, review the Access control in Azure Data Lake Storage Gen2 article.

Cross-tenant scenarios

In cases when Azure Storage is in a different tenant from the Synapse serverless SQL pool, authorization via **Service Principal** is the recommended method. **SAS** authorization is also possible, while **Managed Identity** is not supported.

Supported authorization types for databases users

In the table below you can find the available authorization types for different login methods into Synapse Serverless SQL endpoint:

AUTHORIZATION TYPE	*SQL USER*	*AZURE AD USER*	*SERVICE PRINCIPAL*
User Identity	Not Supported	Supported	Supported
SAS	Supported	Supported	Supported
Service Principal	Supported	Supported	Supported
Managed Identity	Supported	Supported	Supported

Supported storages and authorization types

You can use the following combinations of authorization and Azure Storage types:

AUTHORIZATION TYPE	BLOB STORAGE	ADLS GEN1	ADLS GEN2
SAS	Supported	Not supported	Supported

AUTHORIZATION TYPE	BLOB STORAGE	ADLS GEN1	ADLS GEN2
Service Principal	Supported	Supported	Supported
Managed Identity	Supported	Supported	Supported
User Identity	Supported	Supported	Supported

Firewall protected storage

You can configure storage accounts to allow access to specific serverless SQL pool by creating a resource instance rule. When accessing storage that is protected with the firewall, you can use **User Identity** or **Managed Identity**.

> **NOTE**
> The firewall feature on Storage is in public preview and is available in all public cloud regions.

In the table below you can find the available authorization types for different login methods into Synapse Serverless SQL endpoint:

AUTHORIZATION TYPE	*SQL USER*	*AZURE AD USER*	*SERVICE PRINCIPAL*
User Identity	Not Supported	Supported	Supported
SAS	Not Supported	Not Supported	Not Supported
Service Principal	Not Supported	Not Supported	Not Supported
Managed Identity	Supported	Supported	Supported

- User Identity
- Shared access signature
- Service Principal
- Managed Identity
- Anonymous access

To access storage that is protected with the firewall via User Identity, you can use Azure portal UI or PowerShell module Az.Storage.

Configuration via Azure portal

1. Search for your Storage Account in Azure portal.
2. Go to Networking under section Settings.
3. In Section "Resource instances" add an exception for your Synapse workspace.
4. Select Microsoft.Synapse/workspaces as a Resource type.
5. Select name of your workspace as an Instance name.
6. Click Save.

Configuration via PowerShell

Follow these steps to configure your storage account firewall and add an exception for Synapse workspace.

1. Open PowerShell or install PowerShell

2. Install the Az.Storage 3.4.0 module and Az.Synapse 0.7.0:

```
Install-Module -Name Az.Storage -RequiredVersion 3.4.0
Install-Module -Name Az.Synapse -RequiredVersion 0.7.0
```

> **IMPORTANT**
>
> Make sure that you use **version 3.4.0**. You can check your Az.Storage version by running this command:
>
> ```
> Get-Module -ListAvailable -Name  Az.Storage | select Version
> ```

3. Connect to your Azure Tenant:

```
Connect-AzAccount
```

4. Define variables in PowerShell:

   - Resource group name - you can find this in Azure portal in overview of Storage account.
   - Account Name - name of storage account that is protected by firewall rules.
   - Tenant ID - you can find this in Azure portal in Azure Active Directory in tenant information.
   - Workspace Name - Name of the Synapse workspace.

```
    $resourceGroupName = "<resource group name>"
    $accountName = "<storage account name>"
    $tenantId = "<tenant id>"
    $workspaceName = "<synapse workspace name>"

    $workspace = Get-AzSynapseWorkspace -Name $workspaceName
    $resourceId = $workspace.Id
    $index = $resourceId.IndexOf("/resourceGroups/", 0)
    # Replace G with g - /resourceGroups/ to /resourcegroups/
    $resourceId = $resourceId.Substring(0,$index) + "/resourcegroups/" + $resourceId.Substring($index
+ "/resourceGroups/".Length)
    $resourceId
```

> **IMPORTANT**
>
> Make sure that resource id matches this template in the print of the resourceId variable.
>
> It's important to write **resourcegroups** in lower case. Example of one resource id:
>
> ```
> /subscriptions/{subscription-id}/resourcegroups/{resource-
> group}/providers/Microsoft.Synapse/workspaces/{name-of-workspace}
> ```

5. Add Storage Network rule:

```
    Add-AzStorageAccountNetworkRule -ResourceGroupName $resourceGroupName -Name $accountName -
TenantId $tenantId -ResourceId $resourceId
```

6. Verify that rule was applied in your storage account:

```
    $rule = Get-AzStorageAccountNetworkRuleSet -ResourceGroupName $resourceGroupName -Name
$accountName
    $rule.ResourceAccessRules | ForEach-Object {
        if ($_.ResourceId -cmatch "\/subscriptions\/(\w\-*)+\/resourcegroups\/(.)+") {
            Write-Host "Storage account network rule is successfully configured." -ForegroundColor
Green
            $rule.ResourceAccessRules
        } else {
            Write-Host "Storage account network rule is not configured correctly. Remove this rule
and follow the steps in detail." -ForegroundColor Red
            $rule.ResourceAccessRules
        }
    }
```

Credentials

To query a file located in Azure Storage, your serverless SQL pool end point needs a credential that contains the authentication information. Two types of credentials are used:

- Server-level CREDENTIAL is used for ad-hoc queries executed using `OPENROWSET` function. Credential name must match the storage URL.
- DATABASE SCOPED CREDENTIAL is used for external tables. External table references `DATA SOURCE` with the credential that should be used to access storage.

To allow a user to create or drop a credential, admin can GRANT/DENY ALTER ANY CREDENTIAL permission to a user:

```
GRANT ALTER ANY CREDENTIAL TO [user_name];
```

Database users who access external storage must have permission to use credentials.

Grant permissions to use credential

To use the credential, a user must have `REFERENCES` permission on a specific credential. To grant a `REFERENCES` permission ON a storage_credential for a specific_user, execute:

```
GRANT REFERENCES ON CREDENTIAL::[storage_credential] TO [specific_user];
```

Server-scoped credential

Server-scoped credentials are used when SQL login calls `OPENROWSET` function without `DATA_SOURCE` to read files on some storage account. The name of server-scoped credential **must** match the base URL of Azure storage (optionally followed by a container name). A credential is added by running CREATE CREDENTIAL. You'll need to provide a CREDENTIAL NAME argument.

> **NOTE**
> The `FOR CRYPTOGRAPHIC PROVIDER` argument is not supported.

Server-level CREDENTIAL name must match the full path to the storage account (and optionally container) in the following format: `<prefix>://<storage_account_path>[/<container_name>]`. Storage account paths are described in the following table:

EXTERNAL DATA SOURCE	PREFIX	STORAGE ACCOUNT PATH
Azure Blob Storage	https	<storage_account>.blob.core.windows.net
Azure Data Lake Storage Gen1	https	<storage_account>.azuredatalakestore.net/webhdfs/v1
Azure Data Lake Storage Gen2	https	<storage_account>.dfs.core.windows.net

Server-scoped credentials enable access to Azure storage using the following authentication types:

- User Identity
- Shared access signature
- Service Principal
- Managed Identity
- Public access

Azure AD users can access any file on Azure storage if they have `Storage Blob Data Owner`, `Storage Blob Data Contributor`, or `Storage Blob Data Reader` role. Azure AD users don't need credentials to access storage.

SQL users can't use Azure AD authentication to access storage.

Database-scoped credential

Database-scoped credentials are used when any principal calls `OPENROWSET` function with `DATA_SOURCE` or selects data from external table that don't access public files. The database scoped credential doesn't need to match the name of storage account. It will be explicitly used in DATA SOURCE that defines the location of storage.

Database-scoped credentials enable access to Azure storage using the following authentication types:

- Azure AD Identity
- Shared access signature
- Service Principal
- Managed Identity
- Public access

Azure AD users can access any file on Azure storage if they have at least `Storage Blob Data Owner`, `Storage Blob Data Contributor`, or `Storage Blob Data Reader` role. Azure AD users don't need credentials to access storage.

```
CREATE EXTERNAL DATA SOURCE mysample
WITH (    LOCATION   = 'https://<storage_account>.dfs.core.windows.net/<container>/<path>'
)
```

SQL users can't use Azure AD authentication to access storage.

Database scoped credentials are used in external data sources to specify what authentication method will be used to access this storage:

```
CREATE EXTERNAL DATA SOURCE mysample
WITH (    LOCATION   = 'https://<storage_account>.dfs.core.windows.net/<container>/<path>',
          CREDENTIAL = <name of database scoped credential>
)
```

Examples

Access a publicly available data source

Use the following script to create a table that accesses publicly available data source.

```
CREATE EXTERNAL FILE FORMAT [SynapseParquetFormat]
       WITH ( FORMAT_TYPE = PARQUET)
GO
CREATE EXTERNAL DATA SOURCE publicData
WITH (    LOCATION   = 'https://<storage_account>.dfs.core.windows.net/<public_container>/<path>' )
GO

CREATE EXTERNAL TABLE dbo.userPublicData ( [id] int, [first_name] varchar(8000), [last_name] varchar(8000) )
WITH ( LOCATION = 'parquet/user-data/*.parquet',
       DATA_SOURCE = [publicData],
       FILE_FORMAT = [SynapseParquetFormat] )
```

Database user can read the content of the files from the data source using external table or OPENROWSET function that references the data source:

```
SELECT TOP 10 * FROM dbo.userPublicData;
GO
SELECT TOP 10 * FROM OPENROWSET(BULK 'parquet/user-data/*.parquet',
                                DATA_SOURCE = 'mysample',
                                FORMAT='PARQUET') as rows;
GO
```

Access a data source using credentials

Modify the following script to create an external table that accesses Azure storage using SAS token, Azure AD identity of user, or managed identity of workspace.

```
-- Create master key in databases with some password (one-off per database)
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Y*********0'
GO

-- Create databases scoped credential that use Managed Identity, SAS token or Service Principal. User needs
to create only database-scoped credentials that should be used to access data source:

CREATE DATABASE SCOPED CREDENTIAL WorkspaceIdentity
WITH IDENTITY = 'Managed Identity'
GO
CREATE DATABASE SCOPED CREDENTIAL SasCredential
WITH IDENTITY = 'SHARED ACCESS SIGNATURE', SECRET = 'sv=2019-10-1********ZVsTOL0ltEGhf54N8KhDCRfLRI%3D'
GO
CREATE DATABASE SCOPED CREDENTIAL SPNCredential WITH
IDENTITY = '**44e*****8f6-ag44-1890-34u4-22r23r771098@https://login.microsoftonline.com/**do99dd-87f3-33da-
33gf-3d3rh133ee33/oauth2/token'
, SECRET = '.7OaaU_454azar9WWzLL.Ea9ePPZWzQee~'
GO
-- Create data source that one of the credentials above, external file format, and external tables that
reference this data source and file format:

CREATE EXTERNAL FILE FORMAT [SynapseParquetFormat] WITH ( FORMAT_TYPE = PARQUET)
GO

CREATE EXTERNAL DATA SOURCE mysample
WITH (    LOCATION   = 'https://<storage_account>.dfs.core.windows.net/<container>/<path>'
-- Uncomment one of these options depending on authentication method that you want to use to access data
source:
--,CREDENTIAL = WorkspaceIdentity
--,CREDENTIAL = SasCredential
--,CREDENTIAL = SPNCredential
)

CREATE EXTERNAL TABLE dbo.userData ( [id] int, [first_name] varchar(8000), [last_name] varchar(8000) )
WITH ( LOCATION = 'parquet/user-data/*.parquet',
       DATA_SOURCE = [mysample],
       FILE_FORMAT = [SynapseParquetFormat] );
```

Database user can read the content of the files from the data source using external table or OPENROWSET function that references the data source:

```
SELECT TOP 10 * FROM dbo.userdata;
GO
SELECT TOP 10 * FROM OPENROWSET(BULK 'parquet/user-data/*.parquet', DATA_SOURCE = 'mysample',
FORMAT='PARQUET') as rows;
GO
```

Next steps

The articles listed below will help you learn how query different folder types, file types, and create and use views:

- Query single CSV file
- Query folders and multiple CSV files
- Query specific files
- Query Parquet files
- Create and use views
- Query JSON files
- Query Parquet nested types

Synchronize Apache Spark for Azure Synapse external table definitions in serverless SQL pool

8/19/2022 • 2 minutes to read • Edit Online

Serverless SQL pool can automatically synchronize metadata from Apache Spark. A serverless SQL pool database will be created for each database existing in serverless Apache Spark pools.

For each Spark external table based on Parquet or CSV and located in Azure Storage, an external table is created in a serverless SQL pool database. As such, you can shut down your Spark pools and still query Spark external tables from serverless SQL pool.

When a table is partitioned in Spark, files in storage are organized by folders. Serverless SQL pool will use partition metadata and only target relevant folders and files for your query.

Metadata synchronization is automatically configured for each serverless Apache Spark pool provisioned in the Azure Synapse workspace. You can start querying Spark external tables instantly.

Each Spark Parquet or CSV external table located in Azure Storage is represented with an external table in a dbo schema that corresponds to a serverless SQL pool database.

For Spark external table queries, run a query that targets an external [spark_table]. Before running the following example, make sure you have correct access to the storage account where the files are located.

```
SELECT * FROM [db].dbo.[spark_table]
```

Apache Spark data types to SQL data types mapping

For more information on mapping Apache Spark data types to SQL data types, see Azure Synapse Analytics shared metadata tables.

Next steps

Advance to the Storage Access Control article to learn more about storage access control.

Manageability and monitoring with dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

Synapse SQL allows you to provision a data warehouse via dedicated SQL pool. The articles that follow will help you to manage and monitor your data warehouse. You'll also learn ways to optimize the data warehouse's usage and performance.

Overview

- Learn about compute management and elasticity
- Understand what metrics and logs are available in the Azure portal
- Learn about backup and restore capabilities
- Learn about built-in intelligence and recommendations
- Learn about maintenance periods and what is available to minimize downtime of your data warehouse
- Find common troubleshooting guidance

Next steps

For How-to guides, see Monitor and tune your dedicated SQL pool.

Monitoring resource utilization and query activity in Azure Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

Azure Synapse Analytics provides a rich monitoring experience within the Azure portal to surface insights regarding your data warehouse workload. The Azure portal is the recommended tool when monitoring your data warehouse as it provides configurable retention periods, alerts, recommendations, and customizable charts and dashboards for metrics and logs. The portal also enables you to integrate with other Azure monitoring services such as Azure Monitor (logs) with Log analytics to provide a holistic monitoring experience for not only your data warehouse but also your entire Azure analytics platform for an integrated monitoring experience. This documentation describes what monitoring capabilities are available to optimize and manage your analytics platform with Synapse SQL.

Resource utilization

The following metrics are available in the Azure portal for Synapse SQL. These metrics are surfaced through Azure Monitor.

METRIC NAME	DESCRIPTION	AGGREGATION TYPE
CPU percentage	CPU utilization across all nodes for the data warehouse	Avg, Min, Max
Data IO percentage	IO Utilization across all nodes for the data warehouse	Avg, Min, Max
Memory percentage	Memory utilization (SQL Server) across all nodes for the data warehouse	Avg, Min, Max
Active Queries	Number of active queries executing on the system	Sum
Queued Queries	Number of queued queries waiting to start executing	Sum
Successful Connections	Number of successful connections (logins) against the database	Sum, Count
Failed Connections	Number of failed connections (logins) against the database	Sum, Count
Blocked by Firewall	Number of logins to the data warehouse which was blocked	Sum, Count
DWU limit	Service level objective of the data warehouse	Avg, Min, Max
DWU percentage	Maximum between CPU percentage and Data IO percentage	Avg, Min, Max

METRIC NAME	DESCRIPTION	AGGREGATION TYPE
DWU used	DWU limit * DWU percentage	Avg, Min, Max
Cache hit percentage	(cache hits / (cache hits + cache miss)) * 100, where cache hits are the sum of all columnstore segments hits in the local SSD cache and cache miss is the columnstore segments misses in the local SSD cache summed across all nodes	Avg, Min, Max
Cache used percentage	(cache used / cache capacity) * 100 where cache used is the sum of all bytes in the local SSD cache across all nodes and cache capacity is the sum of the storage capacity of the local SSD cache across all nodes	Avg, Min, Max
Local tempdb percentage	Local tempdb utilization across all compute nodes - values are emitted every five minutes	Avg, Min, Max

Things to consider when viewing metrics and setting alerts:

- DWU used represents only a **high-level representation of usage** across the SQL pool and is not meant to be a comprehensive indicator of utilization. To determine whether to scale up or down, consider all factors which can be impacted by DWU such as concurrency, memory, tempdb, and adaptive cache capacity. We recommend running your workload at different DWU settings to determine what works best to meet your business objectives.
- Failed and successful connections are reported for a particular data warehouse - not for the server itself.
- Memory percentage reflects utilization even if the data warehouse is in idle state - it does not reflect active workload memory consumption. Use and track this metric along with others (`tempdb`, gen2 cache) to make a holistic decision on if scaling for additional cache capacity will increase workload performance to meet your requirements.

Query activity

For a programmatic experience when monitoring Synapse SQL via T-SQL, the service provides a set of Dynamic Management Views (DMVs). These views are useful when actively troubleshooting and identifying performance bottlenecks with your workload.

To view the list of DMVs that apply to Synapse SQL, review dedicated SQL pool DMVs.

Metrics and diagnostics logging

Both metrics and logs can be exported to Azure Monitor, specifically the Azure Monitor logs component and can be programmatically accessed through log queries. The log latency for Synapse SQL is about 10-15 minutes.

Next steps

The following How-to guide describes common scenarios and use cases when monitoring and managing your data warehouse:

- Monitor your data warehouse workload with DMVs
- Use Azure Monitor with your Azure Synapse Analytics workspace

Azure Advisor recommendations for dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

This article describes the dedicated SQL pool recommendations available in Azure Advisor.

Dedicated SQL pool provides recommendations to ensure your data warehouse workload is consistently optimized for performance. Recommendations are tightly integrated with Azure Advisor to provide you with best practices directly within the Azure portal. Dedicated SQL pool collects telemetry and surfaces recommendations for your active workload on a daily cadence. The supported recommendation scenarios are outlined below along with how to apply recommended actions.

You can check your recommendations today!

Data skew

Data skew can cause additional data movement or resource bottlenecks when running your workload. The following documentation describes show to identify data skew and prevent it from happening by selecting an optimal distribution key.

- Identify and remove skew

No or outdated statistics

Having suboptimal statistics can severely impact query performance as it can cause the SQL query optimizer to generate suboptimal query plans. The following documentation describes the best practices around creating and updating statistics:

- Creating and updating table statistics

To see the list of impacted tables by these recommendations, run the following T-SQL script. Advisor continuously runs the same T-SQL script to generate these recommendations.

Replicate tables

For replicated table recommendations, Advisor detects table candidates based on the following physical characteristics:

- Replicated table size
- Number of columns
- Table distribution type
- Number of partitions

Advisor continuously leverages workload-based heuristics such as table access frequency, rows returned on average, and thresholds around data warehouse size and activity to ensure high-quality recommendations are generated.

The following section describes workload-based heuristics you may find in the Azure portal for each replicated table recommendation:

- Scan avg- the average percent of rows that were returned from the table for each table access over the past seven days

- Frequent read, no update - indicates that the table has not been updated in the past seven days while showing access activity
- Read/update ratio - the ratio of how frequent the table was accessed relative to when it gets updated over the past seven days
- Activity - measures the usage based on access activity. This activity compares the table access activity relative to the average table access activity across the data warehouse over the past seven days.

Currently Advisor will only show at most four replicated table candidates at once with clustered columnstore indexes prioritizing the highest activity.

> **IMPORTANT**
> The replicated table recommendation is not full proof and does not take into account data movement operations. We are working on adding this as a heuristic but in the meantime, you should always validate your workload after applying the recommendation. To learn more about replicated tables, visit the following documentation.

Adaptive (Gen2) cache utilization

When you have a large working set, you can experience a low cache hit percentage and high cache utilization. For this scenario, you should scale up to increase cache capacity and rerun your workload. For more information visit the following documentation.

Tempdb contention

Query performance can degrade when there is high tempdb contention. Tempdb contention can occur via user-defined temporary tables or when there is a large amount of data movement. For this scenario, you can scale for more tempdb allocation and configure resource classes and workload management to provide more memory to your queries.

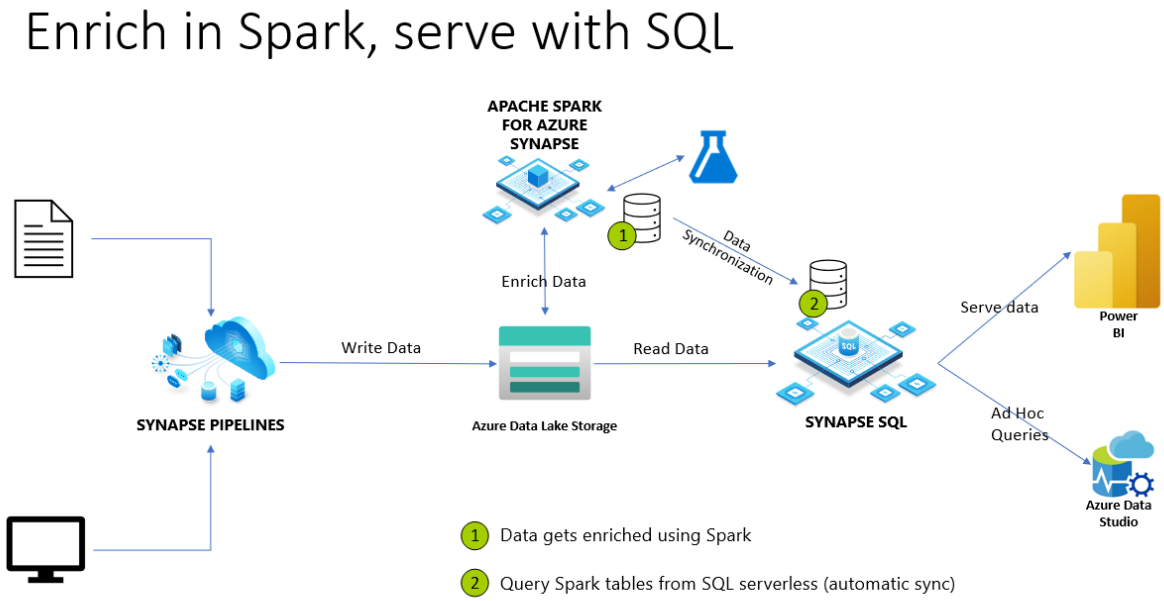
Data loading misconfiguration

You should always load data from a storage account in the same region as your dedicated SQL pool to minimize latency. Use the COPY statement for high throughput data ingestion and split your staged files in your storage account to maximize throughput. If you can't use the COPY statement, you can use the SqlBulkCopy API or bcp with a high batch size for better throughput. See Best practices for data loading for additional data loading guidance.

How to set up access control on synchronized objects in serverless SQL pool

8/19/2022 • 3 minutes to read • Edit Online

In Azure Synapse Analytics, Spark databases and tables are shared with serverless SQL pool. Lake databases, Parquet, and CSV backed tables created with Spark are automatically available in serverless SQL pool. This feature allows using serverless SQL pool to explore and query data prepared by using Spark pools. On below diagram, you can see a high-level architecture overview to utilize this feature. First, Azure Synapse Pipelines are moving data from on-premise (or other) storage to Azure Data Lake Storage. Spark can now enrich the data, and create databases, and tables that are getting synchronized to serverless Synapse SQL. Later, user can execute ad-hoc queries on top of the enriched data or serve it to Power BI for example.



Full administrator access(sysadmin)

Once these databases and tables are synchronized from Spark to serverless SQL pool, these external tables in serverless SQL pool can be used to access the same data. However, objects in serverless SQL pool are read-only because of keeping consistency with the Spark pools objects. The limitation makes only users with Synapse SQL Administrator or Synapse Administrator roles can access these objects in serverless SQL pool. If a non-admin user tries to execute a query on the synchronized database/table, they'll receive error like: `External table '<table>' is not accessible because content of directory cannot be listed.` despite them having access to data on the underlying storage account(s).

Since synchronized databases in serverless SQL pool are read-only, they can't be modified. Creating a user, or giving other permissions will fail if attempted. To read synchronized databases, one must have privileged server-level permissions (like sysadmin). This limitation is also present on external tables in serverless SQL pool when using Azure Synapse Link for Dataverse and lake databases tables.

Non-admin access to synchronized databases

A user who needs to read data and create reports usually doesn't have full administrator access (sysadmin). This

user is usually data analyst who just needs to read and analyze data using the existing tables. They don't need to create new objects.

A user with minimal permission should be able to:

- Connect to a database that is replicated from Spark
- Select data via external tables and access the underlying ADLS data.

After executing the code script below, it will allow non-admin users to have server-level permissions to connect to any database. It will also allow users to view data from all schema-level objects, such as tables or views. Data access security can be managed on the storage layer.

```
-- Creating Azure AD login (same can be achieved for Azure AD app)
    CREATE LOGIN [login@contoso.com] FROM EXTERNAL PROVIDER;
    go;

    GRANT CONNECT ANY DATABASE to [login@contoso.com];
    GRANT SELECT ALL USER SECURABLES to [login@contoso.com];
    GO;
```

`CONNECT ANY DATABASE` permission will allow a user to access connection to any database, but it doesn't grant any permission in any database beyond connect. When `SELECT ALL USER SECURABLES` permission is granted, a login can view data from all schema-level objects, such as external tables and views (any schema except sys and INFORMATION_SCHEMA). This permission has effect in all databases that the user can connect to. Read more about GRANT SERVER permissions.

After creating a login and granting permissions, users can run queries on top of the synchronized external tables. This mitigation can also be applied to Azure AD security groups.

More security on the objects can be managed through specific schemas and lock access to a specific schema. The workaround requires extra DDL. For this scenario, you can create new serverless database, schemas, and views that will point to the Spark tables data on ADLS.

Access to the data on storage account can be managed via ACL or regular Storage Blob Data Owner/Reader/Contributor roles for Azure AD users/groups. For Service Principals (Azure AD apps), make sure you use ACL setup.

> **NOTE**
>
> - If you want to forbid using OPENROWSET on top of the data, you can use `DENY ADMINISTER BULK OPERATIONS to [login@contoso.com];` For more information, visit DENY Server permissions.
> - If you want to forbid using specific schemas, you can use `DENY SELECT ON SCHEMA::[schema_name] TO [login@contoso.com];` For more information, visit DENY Schema Permissions.

Next steps

For more information, see SQL Authentication.

Column-level security

8/19/2022 • 2 minutes to read • Edit Online

Column-Level security allows customers to control access to table columns based on the user's execution context or group membership.

Since this video was posted Row level Security became available for dedicated SQL pool in Azure Synapse.

Column-level security simplifies the design and coding of security in your application, allowing you to restrict column access to protect sensitive data. For example, ensuring that specific users can access only certain columns of a table pertinent to their department. The access restriction logic is located in the database tier rather than away from the data in another application tier. The database applies the access restrictions every time data access is attempted from any tier. This restriction makes your security more reliable and robust by reducing the surface area of your overall security system. In addition, column-level security also eliminates the need for introducing views to filter out columns for imposing access restrictions on the users.

You can implement column-level security with the GRANT T-SQL statement. With this mechanism, both SQL and Azure Active Directory (Azure AD) authentication are supported.

Syntax

```
GRANT <permission> [ ,...n ] ON
    [ OBJECT :: ][ schema_name ]. object_name [ ( column [ ,...n ] ) ]
    TO <database_principal> [ ,...n ]
    [ WITH GRANT OPTION ]
    [ AS <database_principal> ]
<permission> ::=
    SELECT
  | UPDATE
<database_principal> ::=
      Database_user
    | Database_role
    | Database_user_mapped_to_Windows_User
    | Database_user_mapped_to_Windows_Group
```

Example

The following example shows how to restrict `TestUser` from accessing the `SSN` column of the `Membership` table:

Create `Membership` table with SSN column used to store social security numbers:

```
CREATE TABLE Membership
  (MemberID int IDENTITY,
   FirstName varchar(100) NULL,
   SSN char(9) NOT NULL,
   LastName varchar(100) NOT NULL,
   Phone varchar(12) NULL,
   Email varchar(100) NULL);
```

Allow `TestUser` to access all columns except for the SSN column, which has the sensitive data:

```
GRANT SELECT ON Membership(MemberID, FirstName, LastName, Phone, Email) TO TestUser;
```

Queries executed as `TestUser` will fail if they include the SSN column:

```
SELECT * FROM Membership;

-- Msg 230, Level 14, State 1, Line 12
-- The SELECT permission was denied on the column 'SSN' of the object 'Membership', database 'CLS_TestDW',
schema 'dbo'.
```

Use Cases

Some examples of how column-level security is being used today:

- A financial services firm allows only account managers to have access to customer social security numbers (SSN), phone numbers, and other personal data.
- A health care provider allows only doctors and nurses to have access to sensitive medical records while preventing members of the billing department from viewing this data.

What is workload management?

8/19/2022 • 2 minutes to read • Edit Online

Running mixed workloads can pose resource challenges on busy systems. Solution Architects seek ways to separate classic data warehousing activities (such as loading, transforming, and querying data) to ensure that enough resources exist to hit SLAs.

Physical server isolation can lead to pockets of infrastructure that are underutilized, overbooked or in a state where caches are constantly being primed with hardware starting and stopping. A successful workload management scheme effectively manages resources, ensures highly efficient resource utilization, and maximizes return on investment (ROI).

A data warehouse workload refers to all operations that transpire in relation to a data warehouse. The depth and breadth of these components depend on the maturity level of the data warehouse. The data warehouse workload encompasses:

- The entire process of loading data into the warehouse
- Performing data warehouse analysis and reporting
- Managing data in the data warehouse
- Exporting data from the data warehouse

The performance capacity of a data warehouse is determined by the data warehouse units.

- To view the resources allocated for all the performance profiles, see Memory and concurrency limits.
- To adjust capacity, you can scale up or down.

Workload management concepts

In the past, for Synapse SQL in Azure Synapse you managed the query performance through resource classes. Resource classes allowed for assigning memory to a query based on role membership. The primary challenge with resources classes is that, once configured, there was no governance or ability to control the workload.

For example, granting an ad-hoc user role membership to smallrc allowed that user to consume 100% of the memory on the system. With resource classes, there is no way to reserve and ensure resources are available for critical workloads.

Dedicated SQL pool workload management in Azure Synapse consists of three high-level concepts: Workload Classification, Workload Importance, and Workload Isolation. These capabilities give you more control over how your workload utilizes system resources.

Workload classification is the concept of assigning a request to a workload group and setting importance levels. Historically, this assignment was done via role membership using sp_addrolemember. This action can now be done via the CREATE WORKLOAD CLASSIFER. The classification capability provides a richer set of options such as label, session, and time to classify requests.

Workload importance influences the order in which a request gets access to resources. On a busy system, a request with higher importance has first access to resources. Importance can also ensure ordered access to locks.

Workload isolation reserves resources for a workload group. Resources reserved in a workload group are held exclusively for that workload group to ensure execution. Workload groups also allow you to define the amount of resources that are assigned per request, much like resource classes do. Workload groups give you the ability

to reserve or cap the amount of resources a set of requests can consume. Finally, workload groups are a mechanism to apply rules, such as query timeout, to requests.

Next steps

- For more information about workload classification, see Workload Classification.
- For more information about workload isolation, see Workload Isolation.
- For more information about workload importance, see Workload Importance.
- For more information about workload management monitoring, see Workload Management Portal Monitoring.

Workload classification for dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

This article explains the workload classification process of assigning a workload group and importance to incoming requests with dedicated SQL pools in Azure Synapse.

Classification

Workload management classification allows workload policies to be applied to requests through assigning resource classes and importance.

While there are many ways to classify data warehousing workloads, the simplest and most common classification is load and query. You load data with insert, update, and delete statements. You query the data using selects. A data warehousing solution will often have a workload policy for load activity, such as assigning a higher resource class with more resources. A different workload policy could apply to queries, such as lower importance compared to load activities.

You can also subclassify your load and query workloads. Subclassification gives you more control of your workloads. For example, query workloads can consist of cube refreshes, dashboard queries or ad-hoc queries. You can classify each of these query workloads with different resource classes or importance settings. Load can also benefit from subclassification. Large transformations can be assigned to larger resource classes. Higher importance can be used to ensure key sales data is loader before weather data or a social data feed.

Not all statements are classified as they do not require resources or need importance to influence execution. DBCC commands, BEGIN, COMMIT, and ROLLBACK TRANSACTION statements are not classified.

Classification process

Classification for dedicated SQL pool is achieved today by assigning users to a role that has a corresponding resource class assigned to it using sp_addrolemember. The ability to characterize requests beyond a login to a resource class is limited with this capability. A richer method for classification is now available with the CREATE WORKLOAD CLASSIFIER syntax. With this syntax, dedicated SQL pool users can assign importance and how much system resources are assigned to a request via the `workload_group` parameter.

Classification weighting

As part of the classification process, weighting is in place to determine which workload group is assigned. The weighting goes as follows:

CLASSIFIER PARAMETER	WEIGHT
MEMBERNAME:USER	64
MEMBERNAME:ROLE	32
WLM_LABEL	16

CLASSIFIER PARAMETER	WEIGHT
WLM_CONTEXT	8
START_TIME/END_TIME	4

The `membername` parameter is mandatory. However, if the membername specified is a database user instead of a database role, the weighting for user is higher and thus that classifier is chosen.

If a user is a member of multiple roles with different resource classes assigned or matched in multiple classifiers, the user is given the highest resource class assignment. This behavior is consistent with existing resource class assignment behavior.

> **NOTE**
> Classifying managed identities (MI) behavior differs between the dedicated SQL pool in Azure Synapse workspaces and the standalone dedicated SQL pool (formerly SQL DW). While the standalone dedicated SQL pool MI maintains the assigned identity, for Azure Synapse workspaces the MI runs as **dbo**. This cannot be changed. The dbo role, by default, is classified to smallrc. Creating a classifier for the dbo role allows for assigning requests to a workload group other than smallrc. If dbo alone is too generic for classification and has broader impacts, consider using label, session or time-based classification in conjunction with the dbo role classification.

System classifiers

Workload classification has system workload classifiers. The system classifiers map existing resource class role memberships to resource class resource allocations with normal importance. System classifiers can't be dropped. To view system classifiers, you can run the below query:

```
SELECT * FROM sys.workload_management_workload_classifiers where classifier_id <= 12
```

Mixing resource class assignments with classifiers

System classifiers created on your behalf provide an easy path to migrate to workload classification. Using resource class role mappings with classification precedence, can lead to misclassification as you start to create new classifiers with importance.

Consider the following scenario:

- An existing data warehouse has a database user DBAUser assigned to the largerc resource class role. The resource class assignment was done with sp_addrolemember.
- The data warehouse is now updated with workload management.
- To test the new classification syntax, the database role DBARole (which DBAUser is a member of), has a classifier created for them mapping them to mediumrc and high importance.
- When DBAUser logs in and runs a query, the query will be assigned to largerc. Because a user takes precedence over a role membership.

To simplify troubleshooting misclassification, we recommended you remove resource class role mappings as you create workload classifiers. The code below returns existing resource class role memberships. Run sp_droprolemember for each member name returned from the corresponding resource class.

```
SELECT  r.name AS [Resource Class]
,       m.name AS membername
FROM    sys.database_role_members rm
JOIN    sys.database_principals AS r ON rm.role_principal_id = r.principal_id
JOIN    sys.database_principals AS m ON rm.member_principal_id = m.principal_id
WHERE   r.name IN
('mediumrc','largerc','xlargerc','staticrc10','staticrc20','staticrc30','staticrc40','staticrc50','staticrc6
0','staticrc70','staticrc80');

--for each row returned run
sp_droprolemember '[Resource Class]', membername
```

Next steps

- For more information on creating a classifier, see the CREATE WORKLOAD CLASSIFIER (Transact-SQL).
- See the Quickstart on how to create a workload classifier Create a workload classifier.
- See the how-to articles to Configure Workload Importance and how to manage and monitor Workload Management.
- See sys.dm_pdw_exec_requests to view queries and the importance assigned.

Azure Synapse Analytics workload importance

8/19/2022 • 2 minutes to read • Edit Online

This article explains how workload importance can influence the order of execution for dedicated SQL pool requests in Azure Synapse.

Importance

Business needs can require data warehousing workloads to be more important than others. Consider a scenario where mission critical sales data is loaded before the fiscal period close. Data loads for other sources such as weather data don't have strict SLAs. Setting high importance for a request to load sales data and low importance to a request to load weather data ensures the sales data load gets first access to resources and completes quicker.

Importance levels

There are five levels of importance: low, below_normal, normal, above_normal, and high. Requests that don't set importance are assigned the default level of normal. Requests that have the same importance level have the same scheduling behavior that exists today.

Importance scenarios

Beyond the basic importance scenario described above with sales and weather data, there are other scenarios where workload importance helps meet data processing and querying needs.

Locking

Access to locks for read and write activity is one area of natural contention. Activities such as partition switching or RENAME OBJECT require elevated locks. Without workload importance, dedicated SQL pool in Azure Synapse optimizes for throughput. Optimizing for throughput means that when running and queued requests have the same locking needs and resources are available, the queued requests can bypass requests with higher locking needs that arrived in the request queue earlier. Once workload importance is applied to requests with higher locking needs. Request with higher importance will be run before request with lower importance.

Consider the following example:

- Q1 is actively running and selecting data from SalesFact.
- Q2 is queued waiting for Q1 to complete. It was submitted at 9am and is attempting to partition switch new data into SalesFact.
- Q3 is submitted at 9:01am and wants to select data from SalesFact.

If Q2 and Q3 have the same importance and Q1 is still executing, Q3 will begin executing. Q2 will continue to wait for an exclusive lock on SalesFact. If Q2 has higher importance than Q3, Q3 will wait until Q2 is finished before it can begin execution.

Non-uniform requests

Another scenario where importance can help meet querying demands is when requests with different resource classes are submitted. As was previously mentioned, under the same importance, dedicated SQL pool in Azure Synapse optimizes for throughput. When mixed size requests (such as smallrc or mediumrc) are queued, dedicated SQL pool will choose the earliest arriving request that fits within the available resources. If workload

importance is applied, the highest importance request is scheduled next.

Consider the following example on DW500c:

- Q1, Q2, Q3, and Q4 are running smallrc queries.
- Q5 is submitted with the mediumrc resource class at 9am.
- Q6 is submitted with smallrc resource class at 9:01am.

Because Q5 is mediumrc, it requires two concurrency slots. Q5 needs to wait for two of the running queries to complete. However, when one of the running queries (Q1-Q4) completes, Q6 is scheduled immediately because the resources exist to execute the query. If Q5 has higher importance than Q6, Q6 waits until Q5 is running before it can begin executing.

Next steps

- For more information on creating a classifier, see the CREATE WORKLOAD CLASSIFIER (Transact-SQL).
- For more information about workload classification, see Workload Classification.
- See the Quickstart Create workload classifier for how to create a workload classifier.
- See the how-to articles to Configure Workload Importance and how to Manage and monitor Workload Management.
- See sys.dm_pdw_exec_requests to view queries and the importance assigned.

Azure Synapse Analytics workload group isolation

8/19/2022 • 6 minutes to read • Edit Online

This article explains how workload groups can be used to configure workload isolation, contain resources, and apply runtime rules for query execution.

Workload groups

Workload groups are containers for a set of requests and are the basis for how workload management, including workload isolation, is configured on a system. Workload groups are created using the CREATE WORKLOAD GROUP syntax. A simple workload management configuration can manage data loads and user queries. For example, a workload group named `wgDataLoads` will define workload aspects for data being loaded into the system. Also, a workload group named `wgUserQueries` will define workload aspects for users running queries to read data from the system.

The following sections will highlight how workload groups provide the ability to define isolation, containment, request resource definition, and adhere to execution rules.

Resource governance

Workload groups govern memory and CPU resources. Disk and network IO as well as tempdb are not governed. Resource governance for memory and CPU is as follows:

Memory is governed at the request level and held throughout the duration of the request. See Resources per request definition for further details on how to configure the amount of memory per request. The MIN_PERCENTAGE_RESOURCE parameter for the workload group dedicates memory to that workload group exclusively. The CAP_PERCENTAGE_RESOURCE parameter for the workload group is a hard limit on the memory a workload group can consume.

CPU resources are governed at the workload group level and shared by all requests within a workload group. CPU resources are fluid compared to memory which is dedicated to a request for the duration of execution. Given CPU is a fluid resource, unused CPU resources can be consumed by all workload groups. This means that CPU utilization can exceed the CAP_PERCENTAGE_RESOURCE parameter for the workload group. This also means that the MIN_PERCENTAGE_RESOURCE parameter for the workload group is not a hard reservation like memory is. When CPU resources are under contention, utilization will align to the CAP_PERCENTAGE_RESOURCE definition for workload groups.

Workload isolation

Workload isolation means resources are reserved, exclusively, for a workload group. Workload isolation is achieved by configuring the MIN_PERCENTAGE_RESOURCE parameter to greater than zero in the CREATE WORKLOAD GROUP syntax. For continuous execution workloads that need to adhere to tight SLAs, isolation ensures resources are always available for the workload group.

Configuring workload isolation implicitly defines a guaranteed level of concurrency. For example, a workload group with a MIN_PERCENTAGE_RESOURCE set to 30% and REQUEST_MIN_RESOURCE_GRANT_PERCENT set to 2% is guaranteed 15 concurrency. The level of concurrency is guaranteed because 15-2% slots of resources are reserved within the workload group at all times (regardless of how REQUEST_*MAX*_RESOURCE_GRANT_PERCENT is configured). If REQUEST_MAX_RESOURCE_GRANT_PERCENT is greater than REQUEST_MIN_RESOURCE_GRANT_PERCENT and CAP_PERCENTAGE_RESOURCE is greater than MIN_PERCENTAGE_RESOURCE additional resources can be added per request (based on resource availability). If

REQUEST_MAX_RESOURCE_GRANT_PERCENT and REQUEST_MIN_RESOURCE_GRANT_PERCENT are equal and CAP_PERCENTAGE_RESOURCE is greater than MIN_PERCENTAGE_RESOURCE, additional concurrency is possible. Consider the below method for determining guaranteed concurrency:

[Guaranteed Concurrency] = [`MIN_PERCENTAGE_RESOURCE`] / [`REQUEST_MIN_RESOURCE_GRANT_PERCENT`]

> **NOTE**
> There are specific service level minimum values for min_percentage_resource. For more information, see Effective Values for further details.

In the absence of workload isolation, requests operate in the shared pool of resources. Access to resources in the shared pool is not guaranteed and is assigned on an importance basis.

Configuring workload isolation should be done with caution as the resources are allocated to the workload group even if there are no active requests in the workload group. Over-configuring isolation can lead to diminished overall system utilization.

Users should avoid a workload management solution that configures 100% workload isolation: 100% isolation is achieved when the sum of min_percentage_resource configured across all workload groups equals 100%. This type of configuration is overly restrictive and rigid, leaving little room for resource requests that are accidentally mis-classified. There is a provision to allow one request to execute from workload groups not configured for isolation. The resources allocated to this request will show up as a zero in the systems DMVs and borrow a smallrc level of resource grant from system reserved resources.

> **NOTE**
> To ensure optimal resource utilization, consider a workload management solution that leverages some isolation to ensure SLAs are met and mixed with shared resources that are accessed based on workload importance.

Workload containment

Workload containment refers to limiting the amount of resources a workload group can consume. Workload containment is achieved by configuring the CAP_PERCENTAGE_RESOURCE parameter to less than 100 in the CREATE WORKLOAD GROUP syntax. Consider the scenario whereby users need read access to the system so they can run a what-if analysis via ad-hoc queries. These types of requests could have a negative impact on other workloads that are running on the system. Configuring containment ensures the amount of resources is limited.

Configuring workload containment implicitly defines a maximum level of concurrency. With a CAP_PERCENTAGE_RESOURCE set to 60% and a REQUEST_MIN_RESOURCE_GRANT_PERCENT set to 1%, up to a 60-concurrency level is allowed for the workload group. Consider the method included below for determining the maximum concurrency:

[Max Concurrency] = [`CAP_PERCENTAGE_RESOURCE`] / [`REQUEST_MIN_RESOURCE_GRANT_PERCENT`]

> **NOTE**
> The effective CAP_PERCENTAGE_RESOURCE of a workload group will not reach 100% when workload groups with MIN_PERCENTAGE_RESOURCE at a level greater than zero are created. See sys.dm_workload_management_workload_groups_stats for effective runtime values.

Resources per request definition

Workload groups provide a mechanism to define the min and max amount of resources that are allocated per request with the REQUEST_MIN_RESOURCE_GRANT_PERCENT and REQUEST_MAX_RESOURCE_GRANT_PERCENT parameters in the CREATE WORKLOAD GROUP syntax. Resource in this case is memory. CPU resource governance is covered in the Resource governance section.

> **NOTE**
> REQUEST_MAX_RESOURCE_GRANT_PERCENT is an optional parameter that defaults to the same value that is specified for REQUEST_MIN_RESOURCE_GRANT_PERCENT.

Like choosing a resource class, configuring REQUEST_MIN_RESOURCE_GRANT_PERCENT sets the value for the resources utilized by a request. The amount of resources indicated by the set value is guaranteed for allocation to the request before it begins execution. For customers migrating from resource classes to workload groups, consider following the How To article to map from resources classes to workload groups as a starting point.

Configuring REQUEST_MAX_RESOURCE_GRANT_PERCENT to a value greater than REQUEST_MIN_RESOURCE_GRANT_PERCENT allows the system to allocate more resources per request. While scheduling a request, system determines actual resource allocation to the request, which is between REQUEST_MIN_RESOURCE_GRANT_PERCENT and REQUEST_MAX_RESOURCE_GRANT_PERCENT, based on resource availability in shared pool and current load on the system. The resources must exist in the shared pool of resources when the query is scheduled.

> **NOTE**
> REQUEST_MIN_RESOURCE_GRANT_PERCENT and REQUEST_MAX_RESOURCE_GRANT_PERCENT have effective values that are dependent on the effective MIN_PERCENTAGE_RESOURCE and CAP_PERCENTAGE_RESOURCE values. See sys.dm_workload_management_workload_groups_stats for effective runtime values.

Execution Rules

On ad-hoc reporting systems, customers can accidentally execute runaway queries that severely impact the productivity of others. System admins are forced to spend time killing runaway queries to free up system resources. Workload groups offer the ability to configure a query execution timeout rule to cancel queries that have exceeded the specified value. The rule is configured by setting the QUERY_EXECUTION_TIMEOUT_SEC parameter in the CREATE WORKLOAD GROUP syntax.

Shared pool resources

Shared pool resources are the resources not configured for isolation. Workload groups with a MIN_PERCENTAGE_RESOURCE set to zero leverage resources in the shared pool to execute requests. Workload groups with a CAP_PERCENTAGE_RESOURCE greater than MIN_PERCENTAGE_RESOURCE also used shared resources. The amount of resources available in the shared pool is calculated as follows.

[Shared Pool] = 100 - [sum of `MIN_PERCENTAGE_RESOURCE` across all workload groups]

Access to resources in the shared pool is allocated on an importance basis. Requests with the same importance level will access shared pool resources on a first in/first out basis.

Next steps

- Quickstart: configure workload isolation
- CREATE WORKLOAD GROUP
- Convert resource classes to workload groups.

- Workload Management Portal Monitoring.

Azure Synapse Analytics – Workload Management Portal Monitoring

8/19/2022 • 6 minutes to read • Edit Online

This article explains how to monitor workload group resource utilization and query activity. For details on how to configure the Azure Metrics Explorer see the Getting started with Azure Metrics Explorer article. See the Resource utilization section in Azure Synapse Analytics Monitoring documentation for details on how to monitor system resource consumption. There are two different categories of workload group metrics provided for monitoring workload management: resource allocation and query activity. These metrics can be split and filtered by workload group. The metrics can be split and filtered based on if they are system defined (resource class workload groups) or user-defined (created by user with CREATE WORKLOAD GROUP syntax).

Workload management metric definitions

METRIC NAME	DESCRIPTION	AGGREGATION TYPE
Effective cap resource percent	*Effective cap resource percent* is a hard limit on the percentage of resources accessible by the workload group, taking into account *Effective min resource percentage* allocated for other workload groups. The *Effective cap resource percent* metric is configured using the `CAP_PERCENTAGE_RESOURCE` parameter in the CREATE WORKLOAD GROUP syntax. The effective value is described here. For example if a workload group `DataLoads` is created with `CAP_PERCENTAGE_RESOURCE` = 100 and another workload group is created with an Effective min resource percentage of 25%, the *Effective cap resource percent* for the `DataLoads` workload group is 75%. The *Effective cap resource percent* determines the upper bound of concurrency (and thus potential throughput) a workload group can achieve. If additional throughput is needed beyond what is currently reported by the *Effective cap resource percent* metric, either increase the `CAP_PERCENTAGE_RESOURCE`, decrease the `MIN_PERCENTAGE_RESOURCE` of other workload groups or scale up the instance to add more resources. Decreasing the `REQUEST_MIN_RESOURCE_GRANT_PERCENT` can increase concurrency, but may not increase overall throughput.	Min, Avg, Max

METRIC NAME	DESCRIPTION	AGGREGATION TYPE
Effective min resource percent	*Effective min resource percent* is the minimum percentage of resources reserved and isolated for the workload group taking into account the service level minimum. The Effective min resource percent metric is configured using the `MIN_PERCENTAGE_RESOURCE` parameter in the CREATE WORKLOAD GROUP syntax. The effective value is described here. Use the Sum aggregation type when this metric is unfiltered and unsplit to monitor the total workload isolation configured on the system. The *Effective min resource percent* determines the lower bound of guaranteed concurrency (and thus guaranteed throughput) a workload group can achieve. If additional guaranteed resources are needed beyond what is currently reported by the *Effective min resource percent* metric, increase the `MIN_PERCENTAGE_RESOURCE` parameter configured for the workload group. Decreasing the `REQUEST_MIN_RESOURCE_GRANT_PERCENT` can increase concurrency, but may not increase overall throughput.	Min, Avg, Max
Workload group active queries	This metric reports the active queries within the workload group. Using this metric unfiltered and unsplit displays all active queries running on the system.	Sum

METRIC NAME	DESCRIPTION	AGGREGATION TYPE
Workload group allocation by cap resource percent	This metric displays the percentage allocation of resources relative to the *Effective cap resource percent* per workload group. This metric provides the effective utilization of the workload group. Consider a workload group `DataLoads` with an *Effective cap resource percent* of 75% and a `REQUEST_MIN_RESOURCE_GRANT_PERCENT` configured at 25%. The *Workload group allocation by cap resource percent* value filtered to `DataLoads` would be 33% (25% / 75%) if a single query were running in this workload group. Use this metric to identify a workload group's utilization. A value close to 100% indicates all resources available to the workload group are being used. Additionally, the *Workload group queued queries metric* for the same workload group showing a value greater than zero would indicate the workload group would utilize additional resources if allocated. Conversely, if this metric is consistently low and the *Workload group active queries* is low the workload group is not being utilized. This situation is especially problematic if *Effective cap resource percent* is greater than zero as that would indicate underutilized workload isolation.	Min, Avg, Max
Workload group allocation by system percent	This metric displays the percentage allocation of resources relative to the entire system. Consider a workload group `DataLoads` with a `REQUEST_MIN_RESOURCE_GRANT_PERCENT` configured at 25%. *Workload group allocation by system percent* value filtered to `DataLoads` would be 25% (25% / 100%) if a single query were running in this workload group.	Min, Avg, Max

METRIC NAME	DESCRIPTION	AGGREGATION TYPE
Workload group query timeouts	Queries for the workload group that have timed out. Query timeouts reported by this metric are only once the query has started executing (it does not include wait time due to locking or resource waits). Query timeout is configured using the `QUERY_EXECUTION_TIMEOUT_SEC` parameter in the CREATE WORKLOAD GROUP syntax. Increasing the value could reduce the number of query timeouts. Consider increasing the `REQUEST_MIN_RESOURCE_GRANT_PERCENT` parameter for the workload group to reduce the amount of timeouts and allocate more resources per query. Note, increasing `REQUEST_MIN_RESOURCE_GRANT_PERCENT` reduces the amount of concurrency for the workload group.	Sum
Workload group queued queries	Queries for the workload group that are currently queued waiting to start execution. Queries can be queue because they are waiting for resources or locks. Queries could be waiting for numerous reasons. If the system is overloaded and the concurrency demand is greater than what is available, queries will queue. Consider adding more resources to the workload group by increasing the `CAP_PERCENTAGE_RESOURCE` parameter in the CREATE WORKLOAD GROUP statement. If `CAP_PERCENTAGE_RESOURCE` is greater than the *Effective cap resource percent* metric, the configured workload isolation for other workload group is impacting the resources allocated to this workload group. Consider lowering `MIN_PERCENTAGE_RESOURCE` of other workload groups or scale up the instance to add more resources.	Sum

Monitoring scenarios and actions

Below are a series of chart configurations to highlight workload management metric usage for troubleshooting along with associated actions to address the issue.

Underutilized workload isolation

Consider the following workload group and classifier configuration where a workload group named `wgPriority`

is created and *TheCEO* `membername` is mapped to it using the `wcCEOPriority` workload classifier. The `wgPriority` workload group has 25% workload isolation configured for it (`MIN_PERCENTAGE_RESOURCE` = 25). Each query submitted by *TheCEO* is given 5% of system resources (`REQUEST_MIN_RESOURCE_GRANT_PERCENT` = 5).

```
CREATE WORKLOAD GROUP wgPriority
WITH ( MIN_PERCENTAGE_RESOURCE = 25
      ,CAP_PERCENTAGE_RESOURCE = 50
      ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 5);

CREATE WORKLOAD CLASSIFIER wcCEOPriority
WITH ( WORKLOAD_GROUP = 'wgPriority'
      ,MEMBERNAME = 'TheCEO');
```

The below chart is configured as follows:
Metric 1: *Effective min resource percent* (Avg aggregation, `blue line`)
Metric 2: *Workload group allocation by system percent* (Avg aggregation, `purple line`)
Filter: [Workload Group] = `wgPriority`

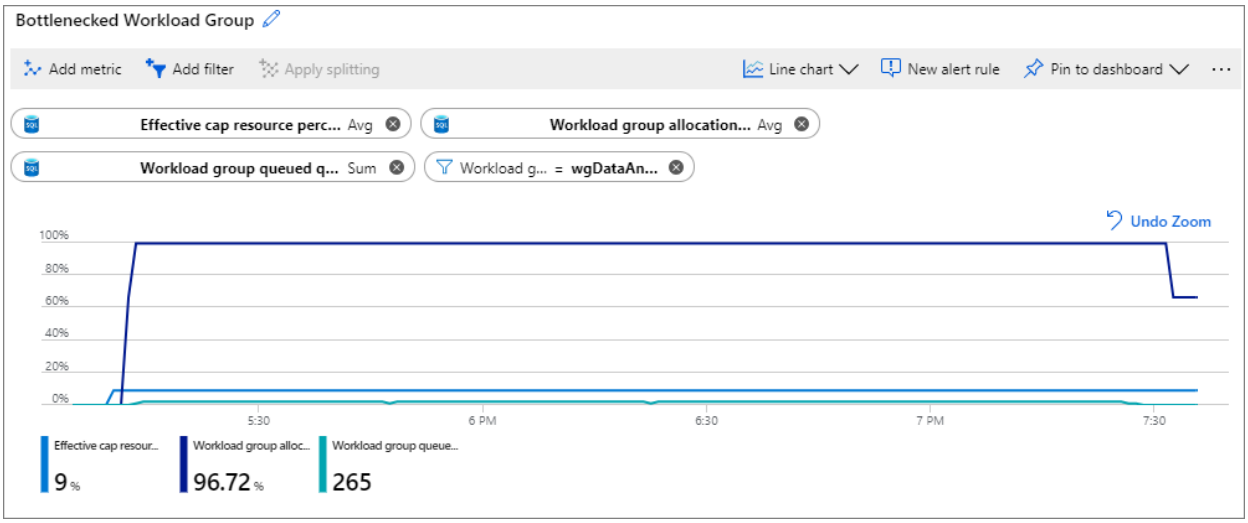
The chart shows that with 25% workload isolation, only 10% is being used on average. In this case, the `MIN_PERCENTAGE_RESOURCE` parameter value could be lowered to between 10 or 15 and allow for other workloads on the system to consume the resources.

Workload group bottleneck

Consider the following workload group and classifier configuration where a workload group named `wgDataAnalyst` is created and the *DataAnalyst* `membername` is mapped to it using the `wcDataAnalyst` workload classifier. The `wgDataAnalyst` workload group has 6% workload isolation configured for it (`MIN_PERCENTAGE_RESOURCE` = 6) and a resource limit of 9% (`CAP_PERCENTAGE_RESOURCE` = 9). Each query submitted by the *DataAnalyst* is given 3% of system resources (`REQUEST_MIN_RESOURCE_GRANT_PERCENT` = 3).

```
CREATE WORKLOAD GROUP wgDataAnalyst
WITH ( MIN_PERCENTAGE_RESOURCE = 6
      ,CAP_PERCENTAGE_RESOURCE = 9
      ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 3);

CREATE WORKLOAD CLASSIFIER wcDataAnalyst
WITH ( WORKLOAD_GROUP = 'wgDataAnalyst'
      ,MEMBERNAME = 'DataAnalyst');
```

The below chart is configured as follows:
Metric 1: *Effective cap resource percent* (Avg aggregation, `blue line`)
Metric 2: *Workload group allocation by cap resource percent* (Avg aggregation, `purple line`)
Metric 3: *Workload group queued queries* (Sum aggregation, `turquoise line`)
Filter: [Workload Group] = `wgDataAnalyst`



The chart shows that with a 9% cap on resources, the workload group is 90%+ utilized (from the *Workload group allocation by cap resource percent metric*). There is a steady queuing of queries as shown from the *Workload group queued queries metric*. In this case, increasing the `CAP_PERCENTAGE_RESOURCE` to a value higher than 9% will allow more queries to execute concurrently. Increasing the `CAP_PERCENTAGE_RESOURCE` assumes that there are enough resources available and not isolated by other workload groups. Verify the cap increased by checking the *Effective cap resource percent metric*. If more throughput is desired, also consider increasing the `REQUEST_MIN_RESOURCE_GRANT_PERCENT` to a value greater than 3. Increasing the `REQUEST_MIN_RESOURCE_GRANT_PERCENT` could allow queries to run faster.

Next steps

- Quickstart: Configure workload isolation using T-SQL
- CREATE WORKLOAD GROUP (Transact-SQL)
- CREATE WORKLOAD CLASSIFIER (Transact-SQL)
- Monitoring resource utilization

Memory and concurrency limits for dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

View the memory and concurrency limits allocated to the various performance levels and resource classes in Azure Synapse Analytics.

> **NOTE**
> Workload management workload groups provide more flexibility for configuring resources per request and concurrency than dynamic or static resource classes. See Workload Groups and the CREATE WORKLOAD GROUP syntax for further details.

Data warehouse capacity settings

The following tables show the maximum capacity for the data warehouse at different performance levels. To change the performance level, see Scale compute - portal.

Service Levels

The service levels range from DW100c to DW30000c.

PERFORMANCE LEVEL	COMPUTE NODES	DISTRIBUTIONS PER COMPUTE NODE	MEMORY PER DATA WAREHOUSE (GB)
DW100c	1	60	60
DW200c	1	60	120
DW300c	1	60	180
DW400c	1	60	240
DW500c	1	60	300
DW1000c	2	30	600
DW1500c	3	20	900
DW2000c	4	15	1200
DW2500c	5	12	1500
DW3000c	6	10	1800
DW5000c	10	6	3000
DW6000c	12	5	3600

PERFORMANCE LEVEL	COMPUTE NODES	DISTRIBUTIONS PER COMPUTE NODE	MEMORY PER DATA WAREHOUSE (GB)
DW7500c	15	4	4500
DW10000c	20	3	6000
DW15000c	30	2	9000
DW30000c	60	1	18000

The maximum service level is DW30000c, which has 60 Compute nodes and one distribution per Compute node. For example, a 600 TB data warehouse at DW30000c processes approximately 10 TB per Compute node.

Concurrency maximums for workload groups

With the introduction of workload groups, the concept of concurrency slots no longer applies. Resources per request are allocated on a percentage basis and specified in the workload group definition. However, even with the removal of concurrency slots, there are minimum amounts of resources needed per queries based on the service level. The below table defined the minimum amount of resources needed per query across service levels and the associated concurrency that can be achieved.

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	MIN % SUPPORTED FOR REQUEST_MIN_RESOURCE_GRANT_PER CENT
DW100c	4	25%
DW200c	8	12.5%
DW300c	12	8%
DW400c	16	6.25%
DW500c	20	5%
DW1000c	32	3%
DW1500c	32	3%
DW2000c	48	2%
DW2500c	48	2%
DW3000c	64	1.5%
DW5000c	64	1.5%
DW6000c	128	0.75%
DW7500c	128	0.75%
DW10000c	128	0.75%

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	MIN % SUPPORTED FOR REQUEST_MIN_RESOURCE_GRANT_PERCENT
DW15000c	128	0.75%
DW30000c	128	0.75%

Concurrency maximums for resource classes

To ensure each query has enough resources to execute efficiently, Synapse SQL tracks resource utilization by assigning concurrency slots to each query. The system puts queries into a queue based on importance and concurrency slots. Queries wait in the queue until enough concurrency slots are available. Importance and concurrency slots determine CPU prioritization. For more information, see Analyze your workload

Static resource classes

The following table shows the maximum concurrent queries and concurrency slots for each static resource class.

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	CONCURRENCY SLOTS AVAILABLE	SLOTS USED BY STATICRC10	SLOTS USED BY STATICRC20	SLOTS USED BY STATICRC30	SLOTS USED BY STATICRC40	SLOTS USED BY STATICRC50	SLOTS USED BY STATICRC60	SLOTS USED BY STATICRC70	SLOTS USED BY STATICRC80
DW100c	4	4	1	2	4	4	4	4	4	4
DW200c	8	8	1	2	4	8	8	8	8	8
DW300c	12	12	1	2	4	8	8	8	8	8
DW400c	16	16	1	2	4	8	16	16	16	16
DW500c	20	20	1	2	4	8	16	16	16	16
DW1000c	32	40	1	2	4	8	16	32	32	32
DW1500c	32	60	1	2	4	8	16	32	32	32
DW2000c	48	80	1	2	4	8	16	32	64	64
DW2500c	48	100	1	2	4	8	16	32	64	64
DW3000c	64	120	1	2	4	8	16	32	64	64

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	CONCURRENCY SLOTS AVAILABLE	SLOTS USED BY STATICRC10	SLOTS USED BY STATICRC20	SLOTS USED BY STATICRC30	SLOTS USED BY STATICRC40	SLOTS USED BY STATICRC50	SLOTS USED BY STATICRC60	SLOTS USED BY STATICRC70	SLOTS USED BY STATICRC80
DW5000c	64	200	1	2	4	8	16	32	64	128
DW6000c	128	240	1	2	4	8	16	32	64	128
DW7500c	128	300	1	2	4	8	16	32	64	128
DW10000c	128	400	1	2	4	8	16	32	64	128
DW15000c	128	600	1	2	4	8	16	32	64	128
DW30000c	128	1200	1	2	4	8	16	32	64	128

Dynamic resource classes

The following table shows the maximum concurrent queries and concurrency slots for each dynamic resource class. Dynamic resource classes use a 3-10-22-70 memory percentage allocation for small-medium-large-xlarge resource classes across all service levels.

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	CONCURRENCY SLOTS AVAILABLE	SLOTS USED BY SMALLRC	SLOTS USED BY MEDIUMRC	SLOTS USED BY LARGERC	SLOTS USED BY XLARGERC
DW100c	4	4	1	1	1	2
DW200c	8	8	1	1	1	5
DW300c	12	12	1	1	2	8
DW400c	16	16	1	1	3	11
DW500c	20	20	1	2	4	14
DW1000c	32	40	1	4	8	28
DW1500c	32	60	1	6	13	42
DW2000c	32	80	2	8	17	56
DW2500c	32	100	3	10	22	70
DW3000c	32	120	3	12	26	84

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	CONCURRENCY SLOTS AVAILABLE	SLOTS USED BY SMALLRC	SLOTS USED BY MEDIUMRC	SLOTS USED BY LARGERC	SLOTS USED BY XLARGERC
DW5000c	32	200	6	20	44	140
DW6000c	32	240	7	24	52	168
DW7500c	32	300	9	30	66	210
DW10000c	32	400	12	40	88	280
DW15000c	32	600	18	60	132	420
DW30000c	32	1200	36	120	264	840

When there are not enough concurrency slots free to start query execution, queries are queued and executed based on importance. If there is equivalent importance, queries are executed on a first-in, first-out basis. As a queries finishes and the number of queries and slots fall below the limits, Azure Synapse Analytics releases queued queries.

Next steps

To learn more about how to leverage resource classes to optimize your workload further please review the following articles:

- Workload management workload groups
- CREATE WORKLOAD GROUP
- Resource classes for workload management
- Analyzing your workload

Workload management with resource classes in Azure Synapse Analytics

8/19/2022 • 16 minutes to read • Edit Online

Guidance for using resource classes to manage memory and concurrency for Synapse SQL pool queries in Azure Synapse.

What are resource classes

The performance capacity of a query is determined by the user's resource class. Resource classes are pre-determined resource limits in Synapse SQL pool that govern compute resources and concurrency for query execution. Resource classes can help you configure resources for your queries by setting limits on the number of queries that run concurrently and on the compute-resources assigned to each query. There's a trade-off between memory and concurrency.

- Smaller resource classes reduce the maximum memory per query, but increase concurrency.
- Larger resource classes increase the maximum memory per query, but reduce concurrency.

There are two types of resource classes:

- Static resources classes, which are well suited for increased concurrency on a data set size that is fixed.
- Dynamic resource classes, which are well suited for data sets that are growing in size and need increased performance as the service level is scaled up.

Resource classes use concurrency slots to measure resource consumption. Concurrency slots are explained later in this article.

- To view the resource utilization for the resource classes, see Memory and concurrency limits.
- To adjust the resource class, you can run the query under a different user or change the current user's resource class membership.

Static resource classes

Static resource classes allocate the same amount of memory regardless of the current performance level, which is measured in data warehouse units. Since queries get the same memory allocation regardless of the performance level, scaling out the data warehouse allows more queries to run within a resource class. Static resource classes are ideal if the data volume is known and constant.

The static resource classes are implemented with these pre-defined database roles:

- staticrc10
- staticrc20
- staticrc30
- staticrc40
- staticrc50
- staticrc60
- staticrc70
- staticrc80

Dynamic resource classes

Dynamic Resource Classes allocate a variable amount of memory depending on the current service level. While

static resource classes are beneficial for higher concurrency and static data volumes, dynamic resource classes are better suited for a growing or variable amount of data. When you scale up to a larger service level, your queries automatically get more memory.

The dynamic resource classes are implemented with these pre-defined database roles:

- smallrc
- mediumrc
- largerc
- xlargerc

The memory allocation for each resource class is as follows.

SERVICE LEVEL	SMALLRC	MEDIUMRC	LARGERC	XLARGERC
DW100c	25%	25%	25%	70%
DW200c	12.5%	12.5%	22%	70%
DW300c	8%	10%	22%	70%
DW400c	6.25%	10%	22%	70%
DW500c	5%	10%	22%	70%
DW1000c to DW30000c	3%	10%	22%	70%

Default resource class

By default, each user is a member of the dynamic resource class **smallrc**.

The resource class of the service administrator is fixed at smallrc and cannot be changed. The service administrator is the user created during the provisioning process. The service administrator in this context is the login specified for the "Server admin login" when creating a new Synapse SQL pool with a new server.

> **NOTE**
> Users or groups defined as Active Directory admin are also service administrators.

Resource class operations

Resource classes are designed to improve performance for data management and manipulation activities. Complex queries can also benefit from running under a large resource class. For example, query performance for large joins and sorts can improve when the resource class is large enough to enable the query to execute in memory.

Operations governed by resource classes

These operations are governed by resource classes:

- INSERT-SELECT, UPDATE, DELETE
- SELECT (when querying user tables)
- ALTER INDEX - REBUILD or REORGANIZE
- ALTER TABLE REBUILD
- CREATE INDEX

- CREATE CLUSTERED COLUMNSTORE INDEX
- CREATE TABLE AS SELECT (CTAS)
- Data loading
- Data movement operations conducted by the Data Movement Service (DMS)

> **NOTE**
> SELECT statements on dynamic management views (DMVs) or other system views are not governed by any of the concurrency limits. You can monitor the system regardless of the number of queries executing on it.

Operations not governed by resource classes

Some queries always run in the smallrc resource class even though the user is a member of a larger resource class. These exempt queries do not count towards the concurrency limit. For example, if the concurrency limit is 16, many users can be selecting from system views without impacting the available concurrency slots.

The following statements are exempt from resource classes and always run in smallrc:

- CREATE or DROP TABLE
- ALTER TABLE ... SWITCH, SPLIT, or MERGE PARTITION
- ALTER INDEX DISABLE
- DROP INDEX
- CREATE, UPDATE, or DROP STATISTICS
- TRUNCATE TABLE
- ALTER AUTHORIZATION
- CREATE LOGIN
- CREATE, ALTER, or DROP USER
- CREATE, ALTER, or DROP PROCEDURE
- CREATE or DROP VIEW
- INSERT VALUES
- SELECT from system views and DMVs
- EXPLAIN
- DBCC

Concurrency slots

Concurrency slots are a convenient way to track the resources available for query execution. They are like tickets that you purchase to reserve seats at a concert because seating is limited. The total number of concurrency slots per data warehouse is determined by the service level. Before a query can start executing, it must be able to reserve enough concurrency slots. When a query completes, it releases its concurrency slots.

- A query running with 10 concurrency slots can access 5 times more compute resources than a query running with 2 concurrency slots.
- If each query requires 10 concurrency slots and there are 40 concurrency slots, then only 4 queries can run concurrently.

Only resource governed queries consume concurrency slots. System queries and some trivial queries don't consume any slots. The exact number of concurrency slots consumed is determined by the query's resource class.

View the resource classes

Resource classes are implemented as pre-defined database roles. There are two types of resource classes:

dynamic and static. To view a list of the resource classes, use the following query:

```
SELECT name
FROM   sys.database_principals
WHERE  name LIKE '%rc%' AND type_desc = 'DATABASE_ROLE';
```

Change a user's resource class

Resource classes are implemented by assigning users to database roles. When a user runs a query, the query runs with the user's resource class. For example, if a user is a member of the staticrc10 database role, their queries run with small amounts of memory. If a database user is a member of the xlargerc or staticrc80 database roles, their queries run with large amounts of memory.

To increase a user's resource class, use sp_addrolemember to add the user to a database role of a large resource class. The below code adds a user to the largerc database role. Each request gets 22% of the system memory.

```
EXEC sp_addrolemember 'largerc', 'loaduser';
```

To decrease the resource class, use sp_droprolemember. If 'loaduser' is not a member or any other resource classes, they go into the default smallrc resource class with a 3% memory grant.

```
EXEC sp_droprolemember 'largerc', 'loaduser';
```

Resource class precedence

Users can be members of multiple resource classes. When a user belongs to more than one resource class:

- Dynamic resource classes take precedence over static resource classes. For example, if a user is a member of both mediumrc(dynamic) and staticrc80 (static), queries run with mediumrc.
- Larger resource classes take precedence over smaller resource classes. For example, if a user is a member of mediumrc and largerc, queries run with largerc. Likewise, if a user is a member of both staticrc20 and statirc80, queries run with staticrc80 resource allocations.

Recommendations

> **NOTE**
> Consider leveraging workload management capabilities (workload isolation, classification and importance) for more control over your workload and predictable performance.

We recommend creating a user that is dedicated to running a specific type of query or load operation. Give that user a permanent resource class instead of changing the resource class on a frequent basis. Static resource classes afford greater overall control on the workload, so we suggest using static resource classes before considering dynamic resource classes.

Resource classes for load users

`CREATE TABLE` uses clustered columnstore indexes by default. Compressing data into a columnstore index is a memory-intensive operation, and memory pressure can reduce the index quality. Memory pressure can lead to needing a higher resource class when loading data. To ensure loads have enough memory, you can create a user that is designated for running loads and assign that user to a higher resource class.

The memory needed to process loads efficiently depends on the nature of the table loaded and the data size. For

more information on memory requirements, see Maximizing rowgroup quality.

Once you have determined the memory requirement, choose whether to assign the load user to a static or dynamic resource class.

- Use a static resource class when table memory requirements fall within a specific range. Loads run with appropriate memory. When you scale the data warehouse, the loads do not need more memory. By using a static resource class, the memory allocations stay constant. This consistency conserves memory and allows more queries to run concurrently. We recommend that new solutions use the static resource classes first as these provide greater control.
- Use a dynamic resource class when table memory requirements vary widely. Loads might require more memory than the current DWU or cDWU level provides. Scaling the data warehouse adds more memory to load operations, which allows loads to perform faster.

Resource classes for queries

Some queries are compute-intensive and some aren't.

- Choose a dynamic resource class when queries are complex, but don't need high concurrency. For example, generating daily or weekly reports is an occasional need for resources. If the reports are processing large amounts of data, scaling the data warehouse provides more memory to the user's existing resource class.
- Choose a static resource class when resource expectations vary throughout the day. For example, a static resource class works well when the data warehouse is queried by many people. When scaling the data warehouse, the amount of memory allocated to the user doesn't change. Consequently, more queries can be executed in parallel on the system.

Proper memory grants depend on many factors, such as the amount of data queried, the nature of the table schemas, and various joins, select, and group predicates. In general, allocating more memory allows queries to complete faster, but reduces the overall concurrency. If concurrency is not an issue, over-allocating memory does not harm throughput.

To tune performance, use different resource classes. The next section gives a stored procedure that helps you figure out the best resource class.

Example code for finding the best resource class

You can use the following specified stored procedure to figure out concurrency and memory grant per resource class at a given SLO and the best resource class for memory intensive CCI operations on non-partitioned CCI table at a given resource class:

Here's the purpose of this stored procedure:

1. To see the concurrency and memory grant per resource class at a given SLO. User needs to provide NULL for both schema and tablename as shown in this example.
2. To see the best resource class for the memory-intensive CCI operations (load, copy table, rebuild index, etc.) on non-partitioned CCI table at a given resource class. The stored proc uses table schema to find out the required memory grant.

Dependencies & Restrictions

- This stored procedure isn't designed to calculate the memory requirement for a partitioned cci table.
- This stored procedure doesn't take memory requirements into account for the SELECT part of CTAS/INSERT-SELECT and assumes it's a SELECT.
- This stored procedure uses a temp table, which is available in the session where this stored procedure was created.
- This stored procedure depends on the current offerings (for example, hardware configuration, DMS config), and if any of that changes then this stored proc won't work correctly.

- This stored procedure depends on existing concurrency limit offerings and if these change then this stored procedure won't work correctly.
- This stored procedure depends on existing resource class offerings and if these change then this stored procedure won't work correctly.

> **NOTE**
>
> If you are not getting output after executing stored procedure with parameters provided, then there could be two cases.
>
> 1. Either DW Parameter contains an invalid SLO value
> 2. Or, there is no matching resource class for the CCI operation on the table.
>
> For example, at DW100c, the highest memory grant available is 1 GB, and if table schema is wide enough to cross the requirement of 1 GB.

Usage example

Syntax:

```
EXEC dbo.prc_workload_management_by_DWU @DWU VARCHAR(7), @SCHEMA_NAME VARCHAR(128), @TABLE_NAME VARCHAR(128)
```

1. @DWU: Either provide a NULL parameter to extract the current DWU from the DW DB or provide any supported DWU in the form of 'DW100c'
2. @SCHEMA_NAME: Provide a schema name of the table
3. @TABLE_NAME: Provide a table name of the interest

Examples executing this stored proc:

```
EXEC dbo.prc_workload_management_by_DWU 'DW2000c', 'dbo', 'Table1';
EXEC dbo.prc_workload_management_by_DWU NULL, 'dbo', 'Table1';
EXEC dbo.prc_workload_management_by_DWU 'DW6000c', NULL, NULL;
EXEC dbo.prc_workload_management_by_DWU NULL, NULL, NULL;
```

The following statement creates Table1 that is used in the preceding examples.

```
CREATE TABLE Table1 (a int, b varchar(50), c decimal (18,10), d char(10), e varbinary(15), f float, g
datetime, h date);
```

Stored procedure definition

```
----------------------------------------------------------------------------
-- Dropping prc_workload_management_by_DWU procedure if it exists.
----------------------------------------------------------------------------
IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P' AND name = 'prc_workload_management_by_DWU')
DROP PROCEDURE dbo.prc_workload_management_by_DWU
GO

----------------------------------------------------------------------------
-- Creating prc_workload_management_by_DWU.
----------------------------------------------------------------------------
CREATE PROCEDURE dbo.prc_workload_management_by_DWU
(@DWU VARCHAR(8),
 @SCHEMA_NAME VARCHAR(128),
 @TABLE_NAME VARCHAR(128)
)
AS

IF @DWU IS NULL
BEGIN
-- Selecting proper DWU for the current DB if not specified.

SELECT @DWU = 'DW'+ CAST(CASE WHEN Mem> 4 THEN Nodes*500
  ELSE Mem*100
```

```
  END AS VARCHAR(10)) +'c'
    FROM (
      SELECT Nodes=count(distinct n.pdw_node_id), Mem=max(i.committed_target_kb/1000/1000/60)
        FROM sys.dm_pdw_nodes n
        CROSS APPLY sys.dm_pdw_nodes_os_sys_info i
        WHERE type = 'COMPUTE')A
END

-- Dropping temp table if exists.
IF OBJECT_ID('tempdb..#ref') IS NOT NULL
BEGIN
  DROP TABLE #ref;
END;

-- Creating ref. temp table (CTAS) to hold mapping info.
CREATE TABLE #ref
WITH (DISTRIBUTION = ROUND_ROBIN)
AS
WITH
-- Creating concurrency slots mapping for various DWUs.
alloc
AS
(
SELECT 'DW100c' AS DWU,4 AS max_queries,4 AS max_slots,1 AS slots_used_smallrc,1 AS slots_used_mediumrc,2 AS
slots_used_largerc,4 AS slots_used_xlargerc,1 AS slots_used_staticrc10,2 AS slots_used_staticrc20,4 AS
slots_used_staticrc30,4 AS slots_used_staticrc40,4 AS slots_used_staticrc50,4 AS slots_used_staticrc60,4 AS
slots_used_staticrc70,4 AS slots_used_staticrc80
  UNION ALL
   SELECT 'DW200c',8,8,1,2,4,8,1,2,4,8,8,8,8,8
  UNION ALL
   SELECT 'DW300c',12,12,1,2,4,8,1,2,4,8,8,8,8,8
  UNION ALL
   SELECT 'DW400c',16,16,1,4,8,16,1,2,4,8,16,16,16,16
  UNION ALL
   SELECT 'DW500c',20,20,1,4,8,16,1,2,4,8,16,16,16,16
  UNION ALL
   SELECT 'DW1000c',32,40,1,4,8,28,1,2,4,8,16,32,32,32
  UNION ALL
   SELECT 'DW1500c',32,60,1,6,13,42,1,2,4,8,16,32,32,32
  UNION ALL
   SELECT 'DW2000c',48,80,2,8,17,56,1,2,4,8,16,32,64,64
  UNION ALL
   SELECT 'DW2500c',48,100,3,10,22,70,1,2,4,8,16,32,64,64
  UNION ALL
   SELECT 'DW3000c',64,120,3,12,26,84,1,2,4,8,16,32,64,64
  UNION ALL
   SELECT 'DW5000c',64,200,6,20,44,140,1,2,4,8,16,32,64,128
  UNION ALL
   SELECT 'DW6000c',128,240,7,24,52,168,1,2,4,8,16,32,64,128
  UNION ALL
   SELECT 'DW7500c',128,300,9,30,66,210,1,2,4,8,16,32,64,128
  UNION ALL
   SELECT 'DW10000c',128,400,12,40,88,280,1,2,4,8,16,32,64,128
  UNION ALL
   SELECT 'DW15000c',128,600,18,60,132,420,1,2,4,8,16,32,64,128
  UNION ALL
   SELECT 'DW30000c',128,1200,36,120,264,840,1,2,4,8,16,32,64,128
)
-- Creating workload mapping to their corresponding slot consumption and default memory grant.
,map
AS
(
  SELECT CONVERT(varchar(20), 'SloDWGroupSmall') AS wg_name, slots_used_smallrc AS slots_used FROM alloc
WHERE DWU = @DWU
UNION ALL
  SELECT CONVERT(varchar(20), 'SloDWGroupMedium') AS wg_name, slots_used_mediumrc AS slots_used FROM alloc
WHERE DWU = @DWU
UNION ALL
  SELECT CONVERT(varchar(20), 'SloDWGroupLarge') AS wg_name, slots_used_largerc AS slots_used FROM alloc
```

```
WHERE DWU = @DWU
UNION ALL
  SELECT CONVERT(varchar(20), 'SloDWGroupXLarge') AS wg_name, slots_used_xlargerc AS slots_used FROM alloc
WHERE DWU = @DWU
  UNION ALL
  SELECT 'SloDWGroupC00',1
  UNION ALL
    SELECT 'SloDWGroupC01',2
  UNION ALL
    SELECT 'SloDWGroupC02',4
  UNION ALL
    SELECT 'SloDWGroupC03',8
  UNION ALL
    SELECT 'SloDWGroupC04',16
  UNION ALL
    SELECT 'SloDWGroupC05',32
  UNION ALL
    SELECT 'SloDWGroupC06',64
  UNION ALL
    SELECT 'SloDWGroupC07',128
)

-- Creating ref based on current / asked DWU.
, ref
AS
(
  SELECT  a1.*
  ,       m1.wg_name          AS wg_name_smallrc
  ,       m1.slots_used * 250 AS tgt_mem_grant_MB_smallrc
  ,       m2.wg_name          AS wg_name_mediumrc
  ,       m2.slots_used * 250 AS tgt_mem_grant_MB_mediumrc
  ,       m3.wg_name          AS wg_name_largerc
  ,       m3.slots_used * 250 AS tgt_mem_grant_MB_largerc
  ,       m4.wg_name          AS wg_name_xlargerc
  ,       m4.slots_used * 250 AS tgt_mem_grant_MB_xlargerc
  ,       m5.wg_name          AS wg_name_staticrc10
  ,       m5.slots_used * 250 AS tgt_mem_grant_MB_staticrc10
  ,       m6.wg_name          AS wg_name_staticrc20
  ,       m6.slots_used * 250 AS tgt_mem_grant_MB_staticrc20
  ,       m7.wg_name          AS wg_name_staticrc30
  ,       m7.slots_used * 250 AS tgt_mem_grant_MB_staticrc30
  ,       m8.wg_name          AS wg_name_staticrc40
  ,       m8.slots_used * 250 AS tgt_mem_grant_MB_staticrc40
  ,       m9.wg_name          AS wg_name_staticrc50
  ,       m9.slots_used * 250 AS tgt_mem_grant_MB_staticrc50
  ,       m10.wg_name          AS wg_name_staticrc60
  ,       m10.slots_used * 250 AS tgt_mem_grant_MB_staticrc60
  ,       m11.wg_name          AS wg_name_staticrc70
  ,       m11.slots_used * 250 AS tgt_mem_grant_MB_staticrc70
  ,       m12.wg_name          AS wg_name_staticrc80
  ,       m12.slots_used * 250 AS tgt_mem_grant_MB_staticrc80
  FROM alloc a1
  JOIN map   m1  ON a1.slots_used_smallrc     = m1.slots_used and m1.wg_name = 'SloDWGroupSmall'
  JOIN map   m2  ON a1.slots_used_mediumrc    = m2.slots_used and m2.wg_name = 'SloDWGroupMedium'
  JOIN map   m3  ON a1.slots_used_largerc     = m3.slots_used and m3.wg_name = 'SloDWGroupLarge'
  JOIN map   m4  ON a1.slots_used_xlargerc    = m4.slots_used and m4.wg_name = 'SloDWGroupXLarge'
  JOIN map   m5  ON a1.slots_used_staticrc10    = m5.slots_used and m5.wg_name NOT IN
('SloDWGroupSmall','SloDWGroupMedium','SloDWGroupLarge','SloDWGroupXLarge')
  JOIN map   m6  ON a1.slots_used_staticrc20    = m6.slots_used and m6.wg_name NOT IN
('SloDWGroupSmall','SloDWGroupMedium','SloDWGroupLarge','SloDWGroupXLarge')
  JOIN map   m7  ON a1.slots_used_staticrc30    = m7.slots_used and m7.wg_name NOT IN
('SloDWGroupSmall','SloDWGroupMedium','SloDWGroupLarge','SloDWGroupXLarge')
  JOIN map   m8  ON a1.slots_used_staticrc40    = m8.slots_used and m8.wg_name NOT IN
('SloDWGroupSmall','SloDWGroupMedium','SloDWGroupLarge','SloDWGroupXLarge')
  JOIN map   m9  ON a1.slots_used_staticrc50    = m9.slots_used and m9.wg_name NOT IN
('SloDWGroupSmall','SloDWGroupMedium','SloDWGroupLarge','SloDWGroupXLarge')
  JOIN map   m10  ON a1.slots_used_staticrc60    = m10.slots_used and m10.wg_name NOT IN
('SloDWGroupSmall','SloDWGroupMedium','SloDWGroupLarge','SloDWGroupXLarge')
  JOIN map   m11  ON a1.slots_used_staticrc70    = m11.slots_used and m11.wg_name NOT IN
```

```
('SloDWGroupSmall','SloDWGroupMedium','SloDWGroupLarge','SloDWGroupXLarge')
  JOIN map   m12  ON a1.slots_used_staticrc80    = m12.slots_used and m12.wg_name NOT IN
('SloDWGroupSmall','SloDWGroupMedium','SloDWGroupLarge','SloDWGroupXLarge')
  WHERE   a1.DWU = @DWU
)
SELECT  DWU
,       max_queries
,       max_slots
,       slots_used
,       wg_name
,       tgt_mem_grant_MB
,       up1 as rc
,       (ROW_NUMBER() OVER(PARTITION BY DWU ORDER BY DWU)) as rc_id
FROM
(
    SELECT  DWU
    ,       max_queries
    ,       max_slots
    ,       slots_used
    ,       wg_name
    ,       tgt_mem_grant_MB
    ,       REVERSE(SUBSTRING(REVERSE(wg_names),1,CHARINDEX('_',REVERSE(wg_names),1)-1)) as up1
    ,       REVERSE(SUBSTRING(REVERSE(tgt_mem_grant_MBs),1,CHARINDEX('_',REVERSE(tgt_mem_grant_MBs),1)-1))
as up2
    ,       REVERSE(SUBSTRING(REVERSE(slots_used_all),1,CHARINDEX('_',REVERSE(slots_used_all),1)-1)) as up3
    FROM    ref AS r1
    UNPIVOT
    (
        wg_name FOR wg_names IN (wg_name_smallrc,wg_name_mediumrc,wg_name_largerc,wg_name_xlargerc,
        wg_name_staticrc10, wg_name_staticrc20, wg_name_staticrc30, wg_name_staticrc40, wg_name_staticrc50,
        wg_name_staticrc60, wg_name_staticrc70, wg_name_staticrc80)
    ) AS r2
    UNPIVOT
    (
        tgt_mem_grant_MB FOR tgt_mem_grant_MBs IN (tgt_mem_grant_MB_smallrc,tgt_mem_grant_MB_mediumrc,
        tgt_mem_grant_MB_largerc,tgt_mem_grant_MB_xlargerc, tgt_mem_grant_MB_staticrc10,
tgt_mem_grant_MB_staticrc20,
        tgt_mem_grant_MB_staticrc30, tgt_mem_grant_MB_staticrc40, tgt_mem_grant_MB_staticrc50,
        tgt_mem_grant_MB_staticrc60, tgt_mem_grant_MB_staticrc70, tgt_mem_grant_MB_staticrc80)
    ) AS r3
    UNPIVOT
    (
        slots_used FOR slots_used_all IN (slots_used_smallrc,slots_used_mediumrc,slots_used_largerc,
        slots_used_xlargerc, slots_used_staticrc10, slots_used_staticrc20, slots_used_staticrc30,
        slots_used_staticrc40, slots_used_staticrc50, slots_used_staticrc60, slots_used_staticrc70,
        slots_used_staticrc80)
    ) AS r4
) a
WHERE   up1 = up2
AND     up1 = up3
;

-- Getting current info about workload groups.
WITH
dmv
AS
(
  SELECT
          rp.name                                       AS rp_name
  ,       rp.max_memory_kb*1.0/1048576                  AS rp_max_mem_GB
  ,       (rp.max_memory_kb*1.0/1024)
          *(request_max_memory_grant_percent/100)       AS max_memory_grant_MB
  ,       (rp.max_memory_kb*1.0/1048576)
          *(request_max_memory_grant_percent/100)       AS max_memory_grant_GB
  ,       wg.name                                       AS wg_name
  ,       wg.importance                                 AS importance
  ,       wg.request_max_memory_grant_percent           AS request_max_memory_grant_percent
  FROM    sys.dm_pdw_nodes_resource_governor_workload_groups wg
  JOIN    sys.dm_pdw_nodes_resource_governor_resource_pools rp    ON  wg.pdw_node_id  = rp.pdw_node_id
```

```
  JOIN    sys.dm_pdw_nodes_resource_governor_resource_pools rp    ON wg.pdw_node_id    = rp.pdw_node_id
                                                                  AND wg.pool_id      = rp.pool_id
  WHERE   rp.name = 'SloDWPool'
  GROUP BY
          rp.name
  ,       rp.max_memory_kb
  ,       wg.name
  ,       wg.importance
  ,       wg.request_max_memory_grant_percent
)
-- Creating resource class name mapping.
,names
AS
(
  SELECT 'smallrc' as resource_class, 1 as rc_id
  UNION ALL
    SELECT 'mediumrc', 2
  UNION ALL
    SELECT 'largerc', 3
  UNION ALL
    SELECT 'xlargerc', 4
  UNION ALL
    SELECT 'staticrc10', 5
  UNION ALL
    SELECT 'staticrc20', 6
  UNION ALL
    SELECT 'staticrc30', 7
  UNION ALL
    SELECT 'staticrc40', 8
  UNION ALL
    SELECT 'staticrc50', 9
  UNION ALL
    SELECT 'staticrc60', 10
  UNION ALL
    SELECT 'staticrc70', 11
  UNION ALL
    SELECT 'staticrc80', 12
)
,base AS
(   SELECT  schema_name
    ,       table_name
    ,       SUM(column_count)                   AS column_count
    ,       ISNULL(SUM(short_string_column_count),0)   AS short_string_column_count
    ,       ISNULL(SUM(long_string_column_count),0)    AS long_string_column_count
    FROM    (   SELECT  sm.name                                             AS schema_name
                ,       tb.name                                             AS table_name
                ,       COUNT(co.column_id)                                 AS column_count
                           ,       CASE    WHEN co.system_type_id IN (36,43,106,108,165,167,173,175,231,239)
                                   AND  co.max_length <= 32
                                   THEN COUNT(co.column_id)
                           END                                              AS short_string_column_count
                ,       CASE    WHEN co.system_type_id IN (165,167,173,175,231,239)
                                   AND  co.max_length > 32 and co.max_length <=8000
                                   THEN COUNT(co.column_id)
                           END                                              AS long_string_column_count
                FROM    sys.schemas AS sm
                JOIN    sys.tables  AS tb   on sm.[schema_id] = tb.[schema_id]
                JOIN    sys.columns AS co   ON tb.[object_id] = co.[object_id]
                           WHERE tb.name = @TABLE_NAME AND sm.name = @SCHEMA_NAME
                GROUP BY sm.name
                ,        tb.name
                ,        co.system_type_id
                ,        co.max_length            ) a
GROUP BY schema_name
,        table_name
)
, size AS
(
SELECT  schema_name
        table_name
```

```
,       table_name
,       75497472                                        AS table_overhead

,       column_count*1048576*8                          AS column_size
,       short_string_column_count*1048576*32                    AS short_string_size,
(long_string_column_count*16777216) AS long_string_size
FROM    base
UNION
SELECT CASE WHEN COUNT(*) = 0 THEN 'EMPTY' END as schema_name
         ,CASE WHEN COUNT(*) = 0 THEN 'EMPTY' END as table_name
         ,CASE WHEN COUNT(*) = 0 THEN 0 END as table_overhead
         ,CASE WHEN COUNT(*) = 0 THEN 0 END as column_size
         ,CASE WHEN COUNT(*) = 0 THEN 0 END as short_string_size

,CASE WHEN COUNT(*) = 0 THEN 0 END as long_string_size
FROM   base
)
, load_multiplier as
(
SELECT  CASE
          WHEN FLOOR(8 * (CAST (CAST(REPLACE(REPLACE(@DWU,'DW',''),'c','') AS INT) AS FLOAT)/6000)) > 0
            AND CHARINDEX(@DWU,'c')=0
          THEN FLOOR(8 * (CAST (CAST(REPLACE(REPLACE(@DWU,'DW',''),'c','') AS INT) AS FLOAT)/6000))
          ELSE 1
        END AS multiplication_factor
)
       SELECT  r1.DWU
       , schema_name
       , table_name
       , rc.resource_class as closest_rc_in_increasing_order
       , max_queries_at_this_rc = CASE
             WHEN (r1.max_slots / r1.slots_used > r1.max_queries)
                  THEN r1.max_queries
             ELSE r1.max_slots / r1.slots_used
                  END
       , r1.max_slots as max_concurrency_slots
       , r1.slots_used as required_slots_for_the_rc
       , r1.tgt_mem_grant_MB  as rc_mem_grant_MB
       ,
CAST((table_overhead*1.0+column_size+short_string_size+long_string_size)*multiplication_factor/1048576    AS
DECIMAL(18,2)) AS est_mem_grant_required_for_cci_operation_MB
       FROM    size
       , load_multiplier
       , #ref r1, names  rc
       WHERE r1.rc_id=rc.rc_id
                     AND
CAST((table_overhead*1.0+column_size+short_string_size+long_string_size)*multiplication_factor/1048576    AS
DECIMAL(18,2)) < r1.tgt_mem_grant_MB
       ORDER BY
ABS(CAST((table_overhead*1.0+column_size+short_string_size+long_string_size)*multiplication_factor/1048576
AS DECIMAL(18,2)) - r1.tgt_mem_grant_MB)
GO
```

Next steps

For more information about managing database users and security, see Secure a database in Synapse SQL. For more information about how larger resource classes can improve clustered columnstore index quality, see Memory optimizations for columnstore compression.

User-defined restore points

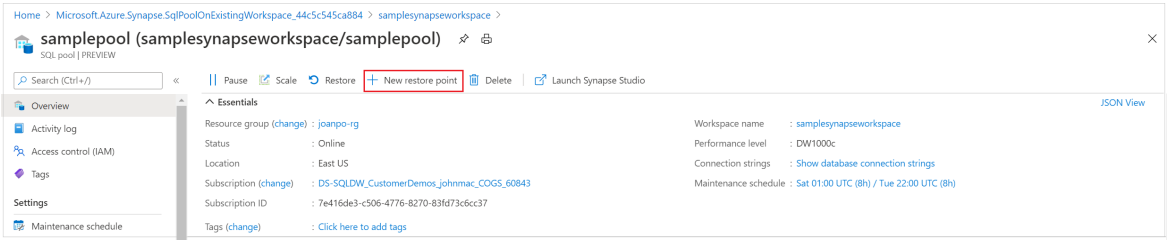
8/19/2022 • 2 minutes to read • Edit Online

In this article, you'll learn to create a new user-defined restore point for a dedicated SQL pool in Azure Synapse Analytics by using the Azure portal.

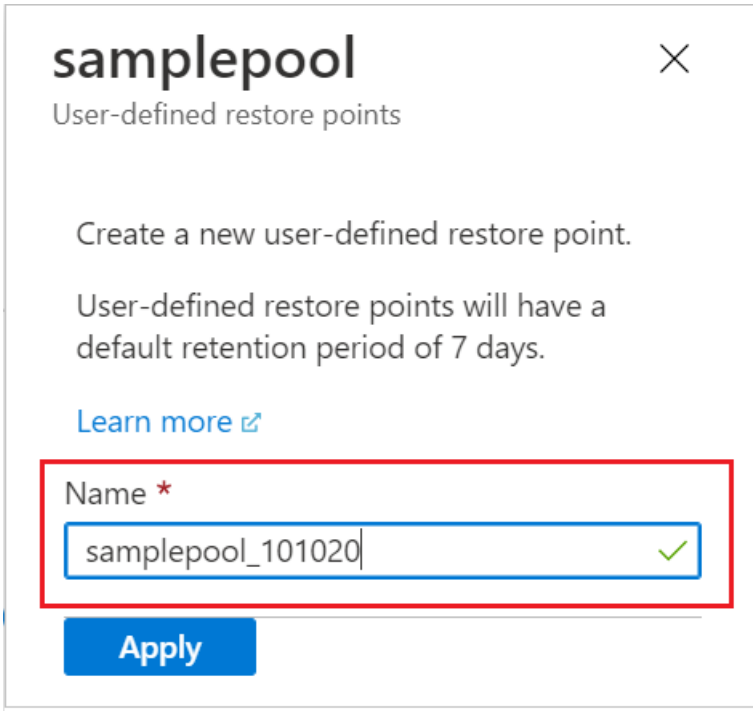
Create user-defined restore points through the Azure portal

User-defined restore points can also be created through Azure portal.

1. Sign in to your Azure portal account.
2. Navigate to the dedicated SQL pool that you want to create a restore point for.
3. Select **Overview** from the left pane, select **+ New restore point**. If the New Restore Point button isn't enabled, make sure that the dedicated SQL pool isn't paused.



4. Specify a name for your user-defined restore point and click **Apply**. User-defined restore points have a default retention period of seven days.



Next steps

- Restore an existing dedicated SQL pool

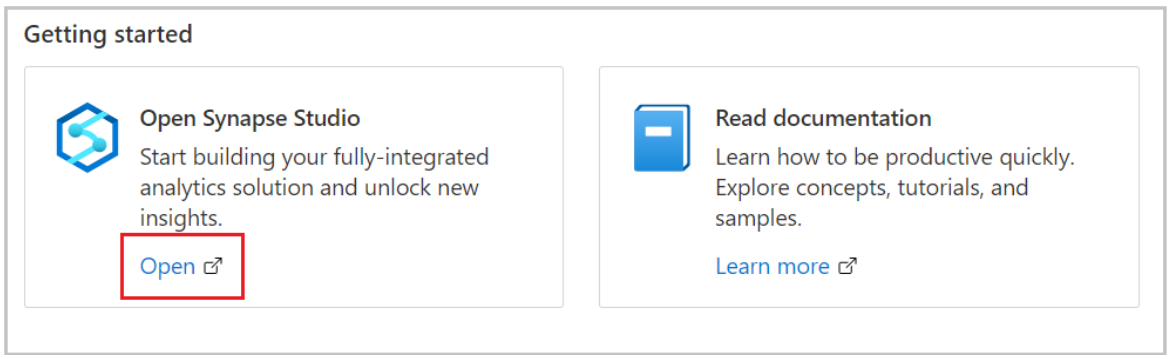
Restore an existing dedicated SQL pool

8/19/2022 • 5 minutes to read • Edit Online

In this article, you learn how to restore an existing dedicated SQL pool in Azure Synapse Analytics using Azure portal, Synapse Studio, and PowerShell. This article applies to both restores and geo-restores.

Restore an existing dedicated SQL pool through the Synapse Studio

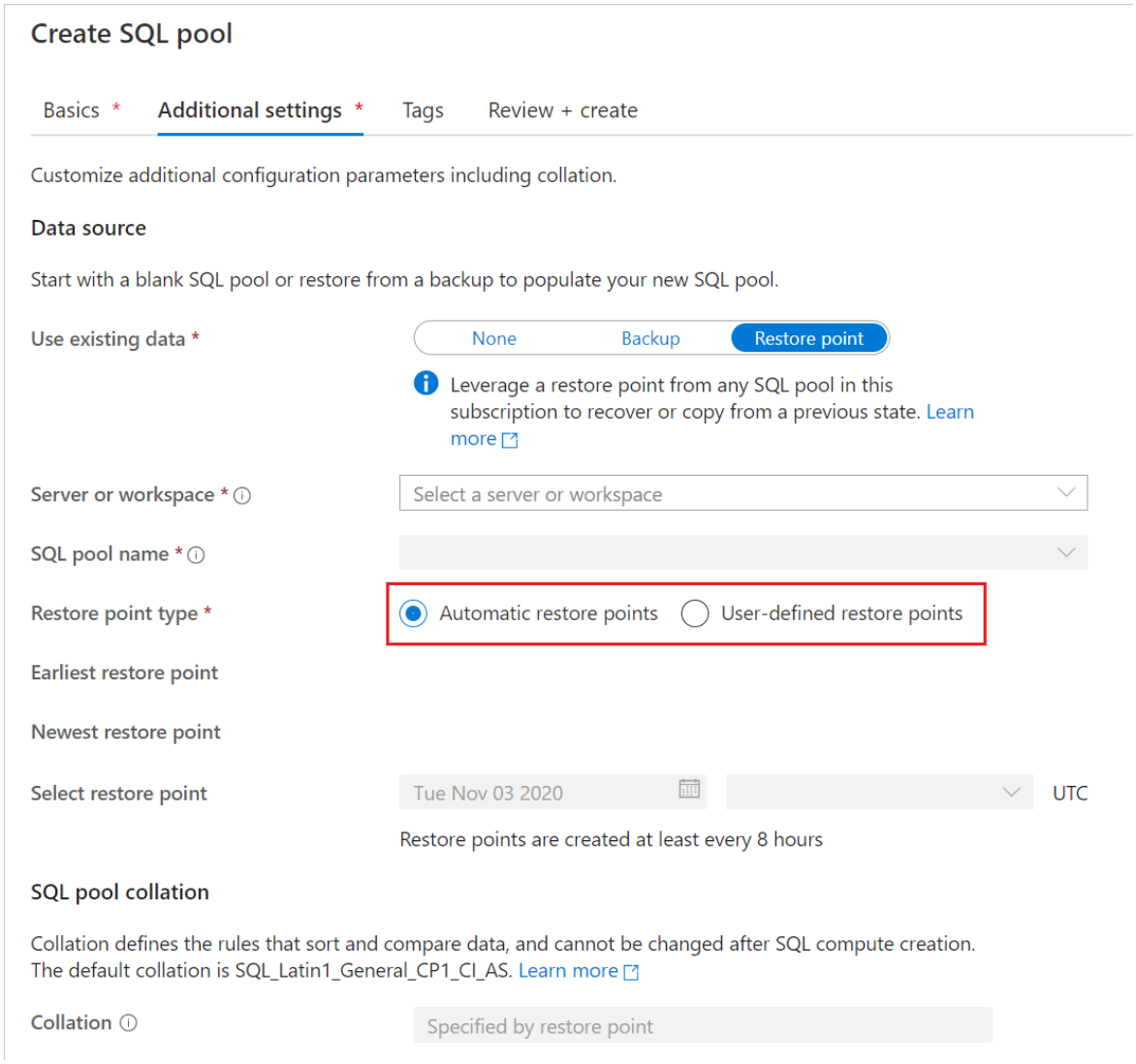
1. Sign in to the Azure portal.
2. Navigate to your Synapse workspace.
3. Under Getting Started -> Open Synapse Studio, select **Open**.



4. On the left hand navigation pane, select **Data**.
5. Select **Manage pools**.
6. Select **+ New** to create a new dedicated SQL pool.
7. In the Additional Settings tab, select a Restore Point to restore from.

   If you want to perform a geo-restore, select the workspace and dedicated SQL pool that you want to recover.

8. Select either **Automatic Restore Points** or **User-Defined Restore Points**.



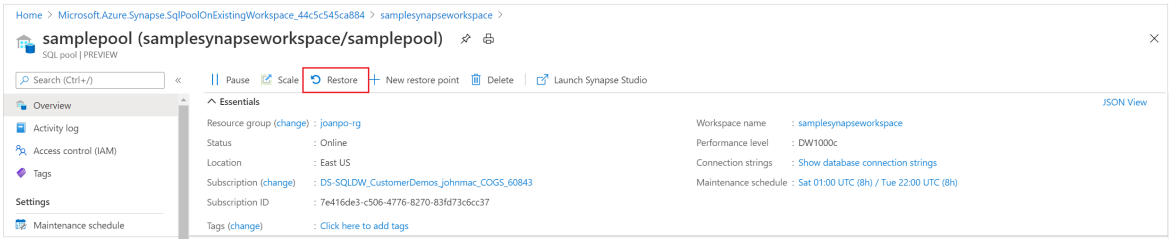
If the dedicated SQL pool doesn't have any automatic restore points, wait a few hours or create a user defined restore point before restoring. For User-Defined Restore Points, select an existing one or create a new one.

If you are restoring a geo-backup, simply select the workspace located in the source region and the dedicated SQL pool you want to restore.

9. Select **Review + Create**.

Restore an existing dedicated SQL pool through the Azure portal

1. Sign in to the Azure portal.
2. Navigate to the dedicated SQL pool that you want to restore from.
3. At the top of the Overview blade, select **Restore**.



4. Select either **Automatic Restore Points** or **User-Defined Restore Points**.

If the dedicated SQL pool doesn't have any automatic restore points, wait a few hours or create a user-

defined restore point before restoring.

If you want to perform a geo-restore, select the workspace and dedicated SQL pool that you want to recover.

5. Select **Review + Create**.

Restore an existing dedicated SQL pool through PowerShell

1. Open PowerShell.
2. Connect to your Azure account and list all the subscriptions associated with your account.
3. Select the subscription that contains the SQL pool to be restored.
4. List the restore points for the dedicated SQL pool.
5. Pick the desired restore point using the RestorePointCreationDate.
6. Restore the dedicated SQL pool to the desired restore point using Restore-AzSynapseSqlPool PowerShell cmdlet.
   a. To restore the dedicated SQL pool to a different workspace, make sure to specify the other workspace name. This workspace can also be in a different resource group and region.
   b. To restore to a different subscription, see the below section.
7. Verify that the restored dedicated SQL pool is online.

```
$SubscriptionName="<YourSubscriptionName>"
$ResourceGroupName="<YourResourceGroupName>"
$WorkspaceName="<YourWorkspaceNameWithoutURLSuffixSeeNote>"  # Without sql.azuresynapse.net
#$TargetResourceGroupName="<YourTargetResourceGroupName>" # uncomment to restore to a different workspace.
#$TargetWorkspaceName="<YourtargetWorkspaceNameWithoutURLSuffixSeeNote>"
$SQLPoolName="<YourDatabaseName>"
$NewSQLPoolName="<YourDatabaseName>"

Connect-AzAccount
Get-AzSubscription
Select-AzSubscription -SubscriptionName $SubscriptionName

# list all restore points
Get-AzSynapseSqlPoolRestorePoint -ResourceGroupName $ResourceGroupName -WorkspaceName $WorkspaceName -Name
$SQLPoolName
# Pick desired restore point using RestorePointCreationDate "xx/xx/xxxx xx:xx:xx xx"
$PointInTime="<RestorePointCreationDate>"

# Get the specific SQL pool to restore
$SQLPool = Get-AzSynapseSqlPool -ResourceGroupName $ResourceGroupName -WorkspaceName $WorkspaceName -Name
$SQLPoolName
# Transform Synapse SQL pool resource ID to SQL database ID because currently the restore command only
accepts the SQL database ID format.
$DatabaseID = $SQLPool.Id -replace "Microsoft.Synapse", "Microsoft.Sql" `
 -replace "workspaces", "servers" `
 -replace "sqlPools", "databases"

# Restore database from a restore point
$RestoredDatabase = Restore-AzSynapseSqlPool –FromRestorePoint -RestorePoint $PointInTime -ResourceGroupName
$SQLPool.ResourceGroupName `
    -WorkspaceName $SQLPool.WorkspaceName -TargetSqlPoolName $NewSQLPoolName –ResourceId $DatabaseID -
PerformanceLevel DW100c

# Use the following command to restore to a different workspace
#$TargetResourceGroupName = $SQLPool.ResourceGroupName # for restoring to different workspace in same
resourcegroup
#$RestoredDatabase = Restore-AzSynapseSqlPool –FromRestorePoint -RestorePoint $PointInTime -
ResourceGroupName $TargetResourceGroupName `
#    -WorkspaceName $TargetWorkspaceName -TargetSqlPoolName $NewSQLPoolName –ResourceId $DatabaseID -
PerformanceLevel DW100c

# Verify the status of restored database
$RestoredDatabase.status
```

Restore an existing dedicated SQL pool to a different subscription through PowerShell

When performing a cross-subscription restore, a synapse workspace dedicated SQL pool can only restore to a standalone dedicated SQL pool (formerly SQL DW). The PowerShell below is similar to the above however there are three main differences:

- After retrieving the SQL Pool object to be restored, the subscription context needs to be switched to the destination (or target) subscription name.
- When performing the restore, use the Az.Sql modules instead of the Az.Synapse modules.
- If it is required to restore the dedicated SQL pool to a Synapse workspace in the destination subscription, an additional restore step is required.

Steps:

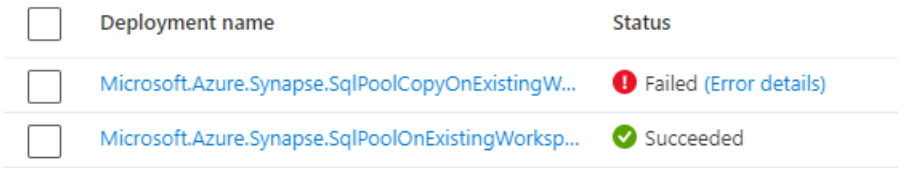
1. Open PowerShell.

2. Update Az.Sql Module to 3.8.0 (or greater) if needed
3. Connect to your Azure account and list all the subscriptions associated with your account.
4. Select the subscription that contains the SQL pool to be restored.
5. List the restore points for the dedicated SQL pool.
6. Pick the desired restore point using the RestorePointCreationDate.
7. Select the destination subscription in which the SQL pool should be restored.
8. Restore the dedicated SQL pool to the desired restore point using Restore-AzSqlDatabase PowerShell cmdlet.
9. Verify that the restored dedicated SQL pool (formerly SQL DW) is online.
10. **If the desired destination is a Synapse Workspace, uncomment the code to perform the additional restore step.**
    a. Create a restore point for the newly created data warehouse.
    b. Retrieve the last restore point created by using the "Select -Last 1" syntax.
    c. Perform the restore to the desired Synapse workspace.

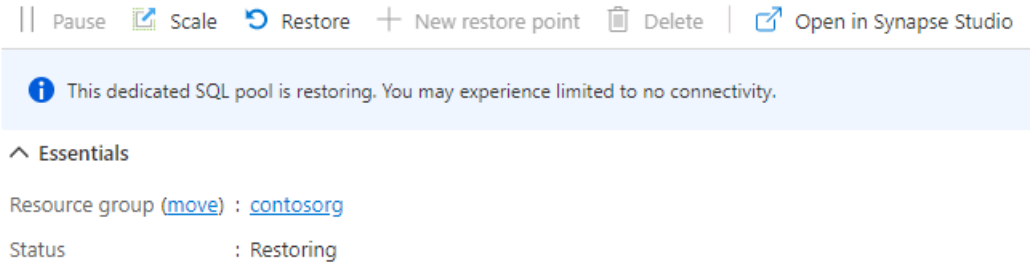
```
$SourceSubscriptionName="<YourSubscriptionName>"
$SourceResourceGroupName="<YourResourceGroupName>"
$SourceWorkspaceName="<YourServerNameWithoutURLSuffixSeeNote>"  # Without sql.azuresynapse.net
$SourceSQLPoolName="<YourDatabaseName>"
$TargetSubscriptionName="<YourTargetSubscriptionName>"
$TargetResourceGroupName="<YourTargetResourceGroupName>"
$TargetServerName="<YourTargetServerNameWithoutURLSuffixSeeNote>"  # Without sql.azuresynapse.net
$TargetDatabaseName="<YourDatabaseName>"
#$TargetWorkspaceName="<YourTargetWorkspaceName>" # uncomment if restore to a synapse workspace is required

# Update Az.Sql module to the latest version (3.8.0 or above)
# Update-Module -Name Az.Sql -RequiredVersion 3.8.0

Connect-AzAccount
Get-AzSubscription
Select-AzSubscription -SubscriptionName $SourceSubscriptionName

# list all restore points
Get-AzSynapseSqlPoolRestorePoint -ResourceGroupName $SourceResourceGroupName -WorkspaceName
$SourceWorkspaceName -Name $SourceSQLPoolName
# Pick desired restore point using RestorePointCreationDate "xx/xx/xxxx xx:xx:xx xx"
$PointInTime="<RestorePointCreationDate>"

# Get the specific SQL pool to restore
$SQLPool = Get-AzSynapseSqlPool -ResourceGroupName $SourceResourceGroupName -WorkspaceName
$SourceWorkspaceName -Name $SourceSQLPoolName
# Transform Synapse SQL pool resource ID to SQL database ID because currently the restore command only
accepts the SQL database ID format.
$DatabaseID = $SQLPool.Id -replace "Microsoft.Synapse", "Microsoft.Sql" `
 -replace "workspaces", "servers" `
 -replace "sqlPools", "databases"

# Switch context to the destination subscription
Select-AzSubscription -SubscriptionName $TargetSubscriptionName

# Restore database from a desired restore point of the source database to the target server in the desired
subscription
$RestoredDatabase = Restore-AzSqlDatabase –FromPointInTimeBackup –PointInTime $PointInTime -
ResourceGroupName $TargetResourceGroupName `
    -ServerName $TargetServerName -TargetDatabaseName $TargetDatabaseName –ResourceId $DatabaseID

# Verify the status of restored database
$RestoredDatabase.status

# uncomment below cmdlets to perform one more restore to push the SQL Pool to an existing workspace in the
destination subscription
# # Create restore point
# New-AzSqlDatabaseRestorePoint -ResourceGroupName $RestoredDatabase.ResourceGroupName -ServerName
$RestoredDatabase.ServerName `
#     -DatabaseName $RestoredDatabase.DatabaseName -RestorePointLabel "UD-001"
# # Gets the last restore point of the sql dw (will use the RestorePointCreationDate property)
# $RestorePoint = Get-AzSqlDatabaseRestorePoint -ResourceGroupName $RestoredDatabase.ResourceGroupName -
ServerName $RestoredDatabase.ServerName `
#     -DatabaseName $RestoredDatabase.DatabaseName | Select -Last 1
# # Restore to destination synapse workspace
# $FinalRestore = Restore-AzSynapseSqlPool –FromRestorePoint -RestorePoint
$RestorePoint.RestorePointCreationDate -ResourceGroupName $TargetResourceGroupName `
#     -WorkspaceName $TargetWorkspaceName -TargetSqlPoolName $TargetDatabaseName –ResourceId
$RestoredDatabase.ResourceID -PerformanceLevel DW100c
```

Troubleshooting

A restore operation can result in a deployment failure based on a "RequestTimeout" exception.



This timeout can be ignored. Review the dedicated SQL pool blade in the portal and it may still have status of "Restoring" and eventually will transition to "Online".



Next Steps

- Create a restore point

Restore a dedicated SQL pool from a deleted workspace

8/19/2022 • 2 minutes to read • Edit Online

In this article, you learn how to restore a dedicated SQL pool in Azure Synapse Analytics after an accidental drop of a workspace using PowerShell.

> **NOTE**
> This guidance is for Synapse Workspace dedicated sql pools only. For standalone dedicated sql pool (formerly SQL DW) please follow guidance Restore sql pool from deleted server.

Before you begin

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Restore the SQL pool from the dropped workspace

1. Open PowerShell
2. Connect to your Azure account.
3. Set the context to the subscription that contains the workspace that was dropped.
4. Specify the approximate datetime the workspace was dropped.
5. Construct the resource ID for the database you wish to recover from the dropped workspace.
6. Restore the database from the dropped workspace
7. Verify the status of the recovered database as 'online'.

```
$SubscriptionID="<YourSubscriptionID>"
$ResourceGroupName="<YourResourceGroupName>"
$WorkspaceName="<YourWorkspaceNameWithoutURLSuffixSeeNote>"  # Without sql.azuresynapse.net
$DatabaseName="<YourDatabaseName>"
$TargetResourceGroupName="<YourTargetResourceGroupName>"
$TargetWorkspaceName="<YourtargetServerNameWithoutURLSuffixSeeNote>"
$TargetDatabaseName="<YourDatabaseName>"

Connect-AzAccount
Set-AzContext -SubscriptionID $SubscriptionID

# Define the approximate point in time the workspace was dropped as DroppedDateTime "yyyy-MM-ddThh:mm:ssZ"
(ex. 2022-01-01T16:15:00Z)
$PointInTime=”<DroppedDateTime>”
$DroppedDateTime = Get-Date -Date $PointInTime


# construct the resource ID of the sql pool you wish to recover. The format required Microsoft.Sql. This
includes the approximate date time the server was dropped.
$SourceDatabaseID =
"/subscriptions/"+$SubscriptionID+"/resourceGroups/"+$ResourceGroupName+"/providers/Microsoft.Sql/servers/"+
$WorkspaceName+"/databases/"+$DatabaseName

# Restore to the target workspace with the source SQL pool.
$RestoredDatabase = Restore-AzSynapseSqlPool -FromDroppedSqlPool -DeletionDate $DroppedDateTime -
TargetSqlPoolName $TargetDatabaseName -ResourceGroupName $TargetResourceGroupName -WorkspaceName
$TargetWorkspaceName -ResourceId $SourceDatabaseID

# Verify the status of restored database
$RestoredDatabase.status
```

Troubleshooting

If "An unexpected error occurred while processing the request." message is received, the original database may not have any recovery points available due to the original workspace being short lived. Typically this is when the workspace existed for less than one hour.

Next Steps

- Create a restore point
- Restore a SQL pool

Pause and resume dedicated SQL pools with Synapse Pipelines

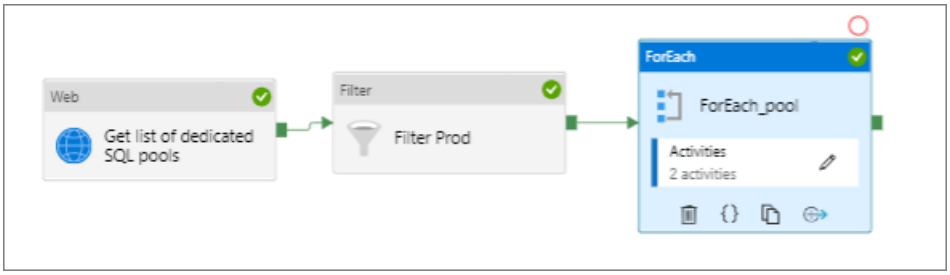
8/19/2022 • 9 minutes to read • Edit Online

Pause and resume for dedicated SQL pools can be automated using Synapse Pipelines in Azure Synapse Analytics. Pause and resume can be used to save costs for a dedicated SQL pool. This solution can easily be included in an existing data orchestration process.

The following steps will guide you through setting up automated pause and resume.

1. Create a pipeline.
2. Set up parameters in your pipeline.
3. Identify the list of dedicated SQL pools in your Azure Synapse workspace.
4. Filter any dedicated SQL pools that you don't want to pause or resume from the list.
5. Loop over each dedicated SQL pool and:
   a. Check the state of the dedicated SQL pool.
   b. Evaluate the state of the dedicated SQL pool.
   c. Pause or resume the dedicated SQL pool.

These steps are laid out in a simple pipeline in Azure Synapse:



Depending upon the nature of your environment, the whole process described here may not apply, and you may just want to choose the appropriate steps. The process described here can be used to pause or resume all instances in a development, test, or PoC environment. For a production environment, you're more likely to schedule pause or resume on an instance by instance basis so will only need Steps 5a through 5c.

The steps above use the REST APIs for Azure Synapse and Azure SQL:

- Dedicated SQL pool operations
- Azure SQL Database REST API

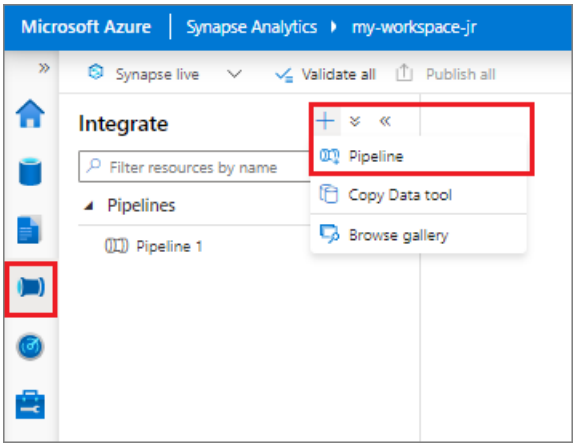
Synapse Pipelines allow for the automation of pause and resume, but you can execute these commands on-demand via the tool or application of your choice.

Prerequisites

- An existing Azure Synapse workspace
- At least one dedicated SQL pool
- Your workspace must be assigned the Azure contributor role to the affected Dedicated SQL Pool(s). See Grant Synapse administrators the Azure Contributor role on the workspace.

Step 1: Create a pipeline in Synapse Studio.

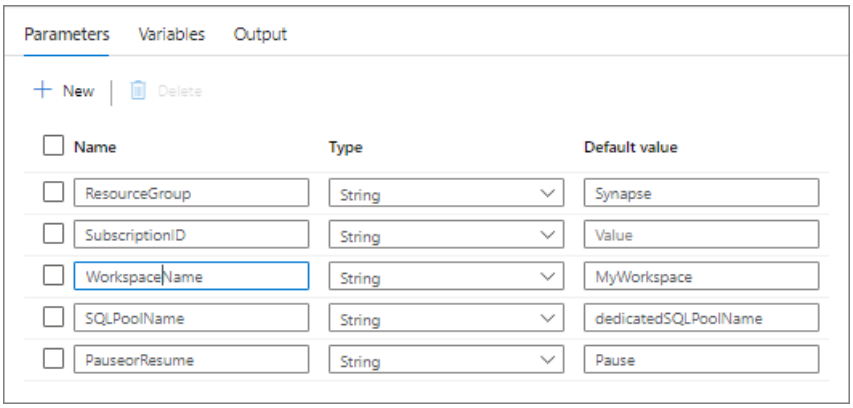
1. Navigate to your workspace and open Synapse Studio.
2. Select the **Integrate** icon, then select the + sign to create a new pipeline.
3. Name your pipeline PauseResume.



Step 2: Create pipeline parameters

The pipeline you'll create will be parameter driven. Parameters allow you to create a generic pipeline that you can use across multiple subscriptions, resource groups, or dedicated SQL pools. Select the **Parameters** tab near the bottom of the pipeline screen. Select **+New** to create each of the following parameters:

NAME	TYPE	DEFAULT VALUE	DESCRIPTION
ResourceGroup	string	Synapse	Name of the resource group for your dedicated SQL pools
SubscriptionID	string	`<SubscriptionID>`	Subscription ID for your resource group
WorkspaceName	string	Synapse	Name of your workspace
SQLPoolName	string	SQLPool1	Name of your dedicated SQL pool
PauseorResume	string	Pause	The state wanted at the end of the pipeline run



Step 3: Create list of dedicated SQL pools

Set up a **Web** activity, you'll use this activity to create the list of dedicated SQL pools by calling the dedicated SQL pools - List By Server REST API request. The output is a JSON string that contains a list of the dedicated SQL pools in your workspace. The JSON string is passed to the next activity.

1. Under **Activities** > **General** drag a **Web** activity to the pipeline canvas as the first stage of your pipeline.
2. In the **General** tab, name this stage GET List.
3. Select the **Settings** tab then click in the **URL** entry space, then select **Add dynamic content**. Copy and paste the GET request that has been parameterized using the @concat string function below into the dynamic content box. Select **Finish**. The following code is a simple Get request:

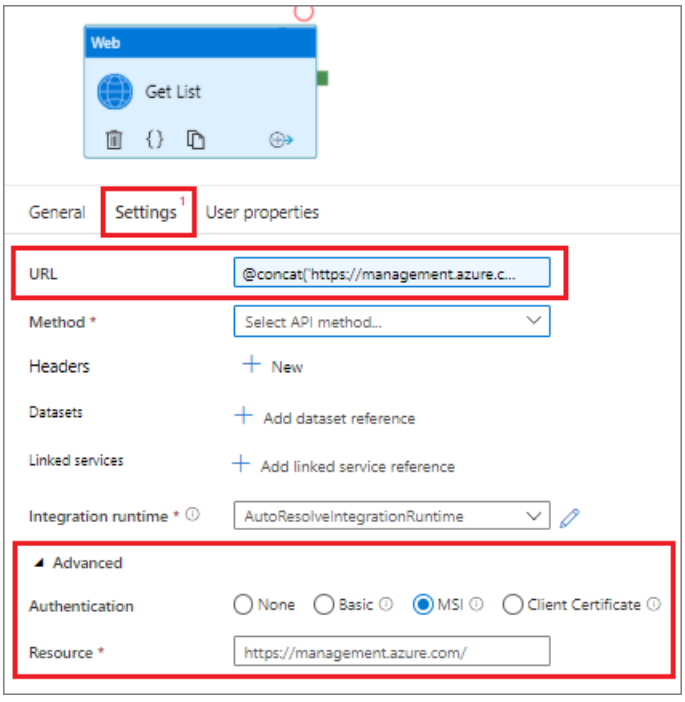
   ```
   GET https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group-name}/providers/Microsoft.Synapse/workspaces/{workspace-name}/sqlPools?api-version=2019-06-01-preview
   ```

   GET request that has been parameterized using the @concat string function:

   ```
   @concat('https://management.azure.com/subscriptions/',pipeline().parameters.SubscriptionID,'/resourceGroups/',pipeline().parameters.ResourceGroup,'/providers/Microsoft.Synapse/workspaces/',pipeline().parameters.WorkspaceName,'/sqlPools?api-version=2019-06-01-preview')
   ```

4. Select the drop-down for **Method** and select **Get**.
5. Select **Advanced** to expand the content. Select **MSI** as the Authentication type. For Resource enter `https://management.azure.com/`

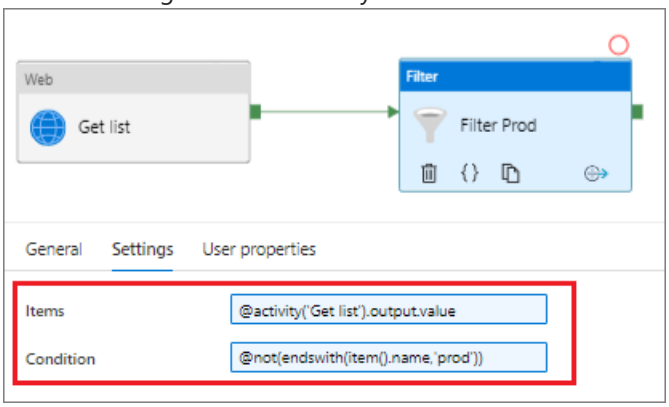
   > **IMPORTANT**
   > For all of the Web Activities / REST API Web calls, you need to ensure that Synapse Pipeline is authenticated against dedicated SQL pool. Managed Identity is required to run these REST API calls.



Step 4: Filter dedicated SQL pools

Remove dedicated SQL pools that you don't want to pause or resume. Use a filter activity that filters the values passed from the Get list activity. In this example, we're extracting the records from the array that don't have "prod" in the name. Apply other conditions as needed. For example, filter on the sku/name of the Synapse workspace to ensure only valid dedicated SQL pools are identified.

1. Select and drag the **Filter** activity under **Iteration & conditionals** to the pipeline canvas.

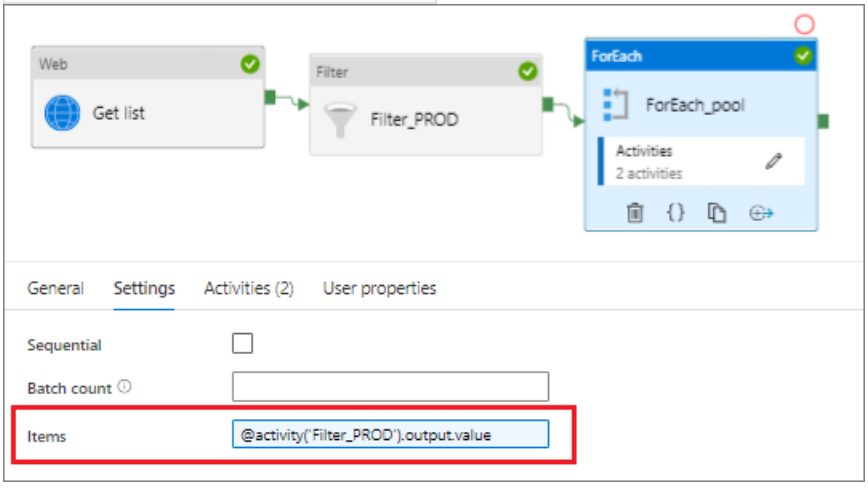


2. Connect the Get List Web activity to the Filter activity. Select the green tab on the Web activity and drag it to the Filter box.
3. Enter `@activity('Get list').output.value` for **Items** where GET List is the name of the preceding Web activity
4. Enter `@not(endswith(item().name,'prod'))` for **Condition**. The remaining records in the array are then passed to the next activity.

Step 5: Create a ForEach loop

Create a ForEach activity to loop over each dedicated SQL pool.

1. Select and drag the **ForEach** activity under **Iteration & conditionals** to the pipeline canvas.
2. On the **General** tab name the activity, we have used 'ForEach_pool'.
3. On the Settings tab, select the **Items** input and select **Add dynamic content**. Scroll to the **Activity outputs** and select the output from your filter activity. Add `.value` to the activity. the value should be similar to `@activity('Filter_PROD').output.value`. Select **finish**.



4. Select the **Activities** tab and select the edit pencil to open the ForEach loop canvas.

Step 5a: Check the state of the dedicated SQL pools

Checking the state of the dedicated SQL pool requires a Web Activity, similar to step 1. This activity calls the Check dedicated SQL pool state REST API for Azure Synapse.

1. Select and drag a **Web** activity under **General** to the pipeline canvas.

2. In the **General** tab, name this stage CheckState.

3. Select the **Settings** tab.

4. Click in the **URL** entry space, then select **Add dynamic content**. Copy and paste the GET request that has been parameterized using the @concat string function from below into the dynamic content box. Select **Finish**. Checking the state again uses a Get request using the following call:

   ```
   GET https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group-name}/providers/Microsoft.Synapse/workspaces/{workspace-name}/sqlPools/{database-name}?api-version=2019-06-01-preview HTTP/1.1
   ```

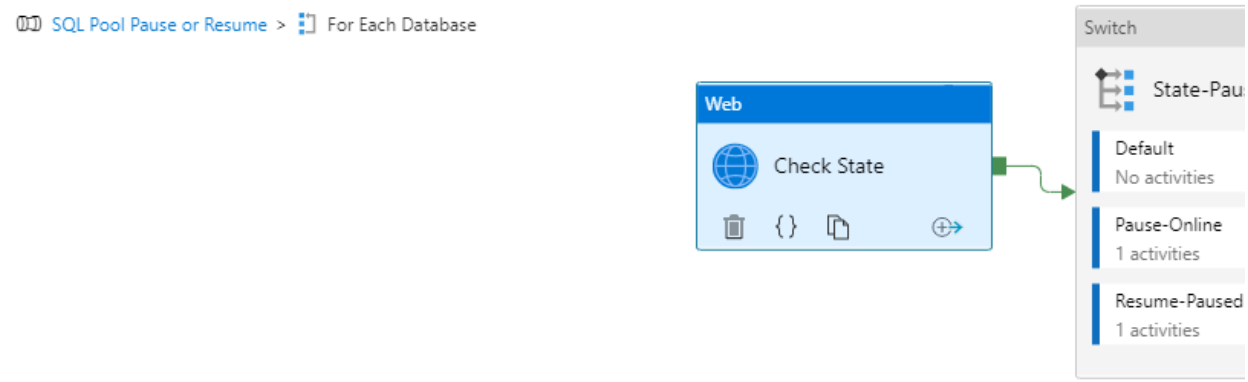
   The parameterize GET request using the @concat string function:

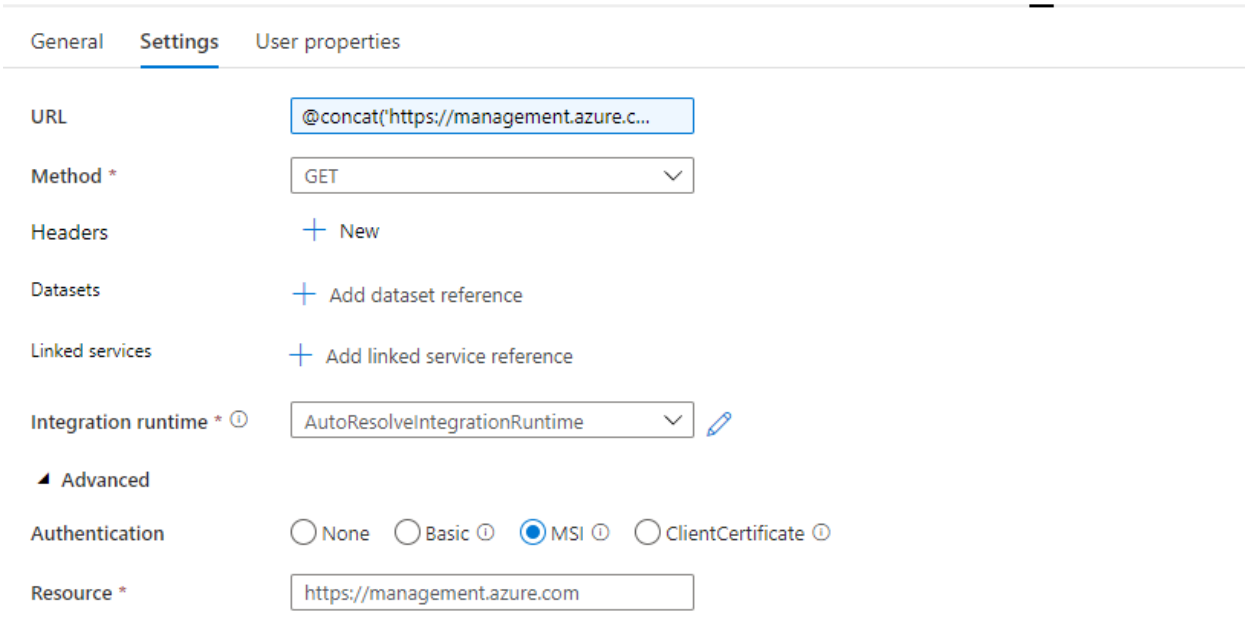
   ```
   @concat('https://management.azure.com/subscriptions/',pipeline().parameters.SubscriptionID,'/resourceGroups/',pipeline().parameters.ResourceGroup,'/providers/Microsoft.Synapse/workspaces/',pipeline().parameters.WorkspaceName,'/sqlPools/',item().name,'?api-version=2019-06-01-preview')
   ```

   In this case, we are using item().name, which is the name of the dedicated SQL pool from Step 1 that was passed to this activity from the ForEach loop. If you are using a pipeline to control a single dedicated SQL pool, you can embed the name of your dedicated SQL pool here, or use a parameter from the pipeline. For example, you could use pipeline().parameters.SQLPoolName.

   The output is a JSON string that contains details of the dedicated SQL pool, including its status (in properties.status). The JSON string is passed to the next activity.

5. Select the drop-down for **Method** and select **Get** Select **Advanced** to expand the content. Select **MSI** as the Authentication type. For Resource enter `https://management.azure.com/`





Step 5b: Evaluate the state of the dedicated SQL pools

Evaluate the desired state, Pause or Resume, and the current status, Online, or Paused, and then initiate Pause or Resume as needed.

1. Select and drag a **Switch** activity, under **Iteration & conditionals**, to the pipeline canvas.
2. Connect the **Switch** activity to the **CheckState** activity. Select the green tab on the Web activity and drag it to the Switch box.
3. In the **General** tab, name this stage State-PauseOrResume.

   Based on the desired state and the current status, only the following two combinations will require a change in state: Paused->Resume or Online->Pause.
4. On the **Activities** tab, copy the code below into the **Expression**.

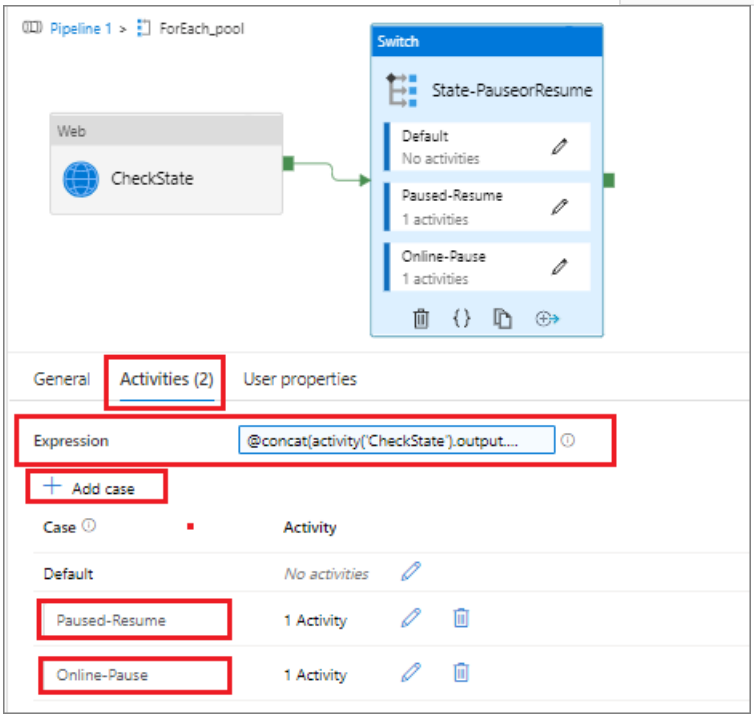
   ```
   @concat(activity('CheckState').output.properties.status,'-',pipeline().parameters.PauseOrResume)
   ```

   Where Check State is the name of the preceding Web activity with output.properties.status defining the current status and pipeline().parameters.PauseOrResume indicates the desired state.

   The check condition does a check of the desired state and the current status. If the desired state is Resume and the current status is Paused, a Resume Activity is invoked within the Paused-Resume Case. If the

desired state is Pause and the current status is Online, a Pause Activity is invoked with the Online-Pause Case. Any other cases, such as a desired state of Pause and a current status of Paused, or a desired state of Resume and a current status of Online, would require no action and be handled by the Default case, which has no activities.

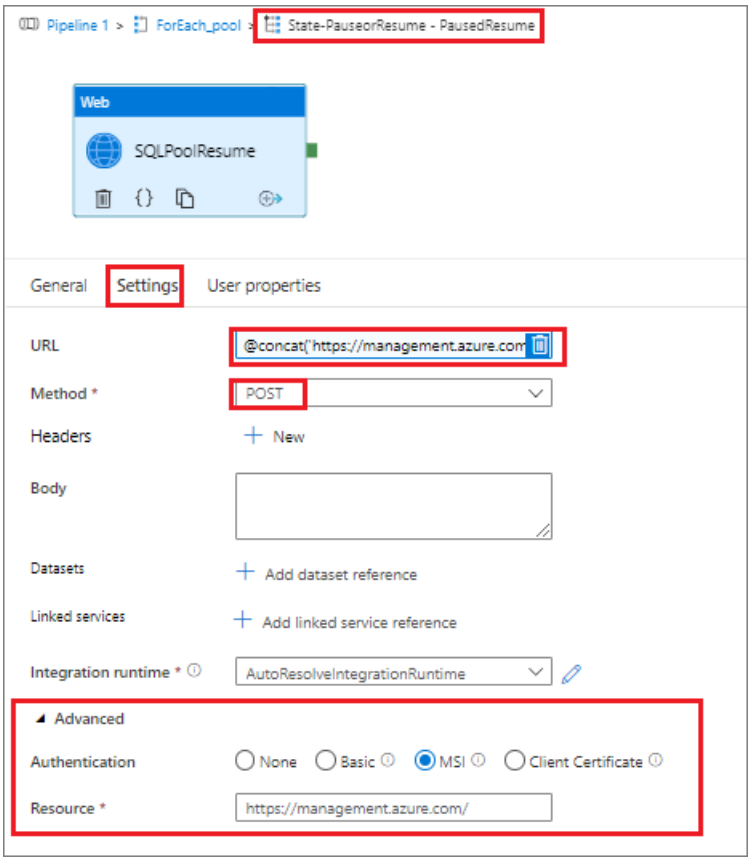
5. On the Activities tab, select **+ Add Case**. Add the cases `Paused-Resume` and `Online-Pause`.



Step 5c: Pause or Resume dedicated SQL pools

The final and only relevant step for some requirements, is to initiate the pause or resume of your dedicated SQL pool. This step again uses a Web activity, calling the Pause or Resume compute REST API for Azure Synapse.

1. Select the activity edit pencil and add a **Web** activity to the State-PauseorResume canvas.
2. Select the **Settings** tab then click in the **URL** entry space, then select **Add dynamic content**. Copy and paste the POST request that has been parameterized using the @concat string function below into the dynamic content box. Select **Finish**.
3. Select the drop-down for **Method** and select **POST**.
4. In the Body section type "Pause and Resume"
5. Select **Advanced** to expand the content. Select **MSI** as the Authentication type. For Resource enter `https://management.azure.com/`
6. Add a second activity for the resume functionality using the parameterized code below.



The example here is to resume a dedicated SQL pool, invoking a POST request using the following call:

```
POST https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group-name}/providers/Microsoft.Synapse/workspaces/{workspace-name}/sqlPools/{database-name}/resume?api-version=2019-06-01-preview HTTP/1.1
```

You can parameterize the POST statement from above using the @concat string function:

```
@concat('https://management.azure.com/subscriptions/',pipeline().parameters.SubscriptionID,'/resourceGroups/',pipeline().parameters.ResourceGroup,'/providers/Microsoft.Synapse/workspaces/',pipeline().parameters.WorkspaceName,'/sqlPools/',activity('CheckState').output.name,'/resume?api-version=2019-06-01-preview')
```

In this case, we are using the activity 'Check State'.output.name with the names of the dedicated SQL pools from Step 3a that were passed to this activity through the Switch Condition. If you are using a single activity against a single database, you could embed the name of your dedicated SQL pool here, or use a parameter from the pipeline. For example, you could use the pipeline().parameters.DatabaseName.

The POST request to pause a dedicated SQL pool is:

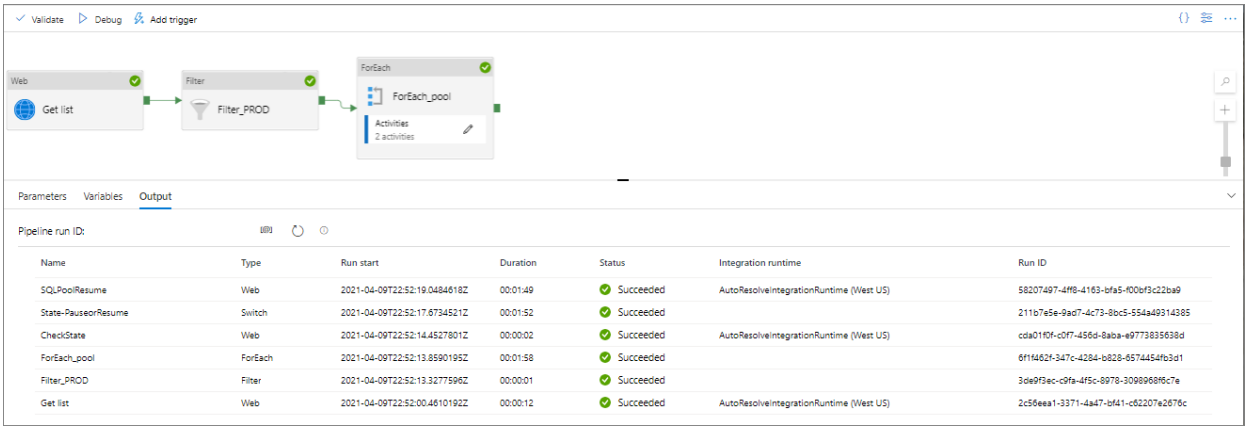
```
POST https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group-name}/providers/Microsoft.Synapse/workspaces/{workspace-name}/sqlPools/{database-name}/pause?api-version=2019-06-01-preview HTTP/1.1
```

The POST request can be parameterized using the @concat string function as shown:

```
@concat('https://management.azure.com/subscriptions/',pipeline().parameters.SubscriptionID,'/resourceGroups/',pipeline().parameters.ResourceGroup,'/providers/Microsoft.Synapse/workspaces/',pipeline().parameters.WorkspaceName,'/sqlPools/',activity('CheckState').output.name,'/pause?api-version=2019-06-01-preview')
```

Pipeline run output

When the full pipeline is run, you will see the output listed below. You can run your pipeline by selecting **Debug** mode or by selecting **add trigger**. For the pipeline results below, the pipeline parameter named "ResourceGroup" was set to a single resource group that had two Synapse Workspaces. One was named testprod and was filtered out, the second was named test1. The test1 dedicated SQL pool was paused, so the job initiated a resume.

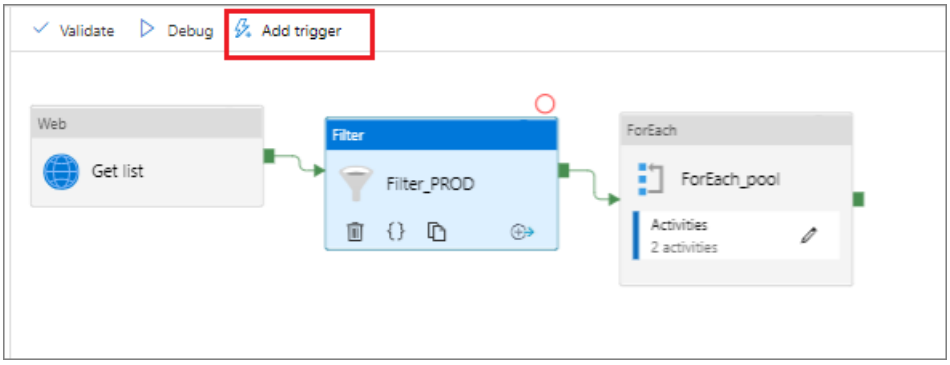


Save your pipeline

To save your pipeline, select **Publish all** above your pipeline.

Schedule your pause or resume pipeline to run

To schedule your pipeline, select **Add trigger** at the top of your pipeline. Follow the screens to schedule your pipeline to run at a specified time.



Next steps

Further details on Managed Identity for Azure Synapse, and how Managed Identity is added to your dedicated SQL pool can be found here:

Azure Synapse workspace managed identity

Grant permissions to workspace managed identity

Use Synapse Studio to monitor your SQL requests

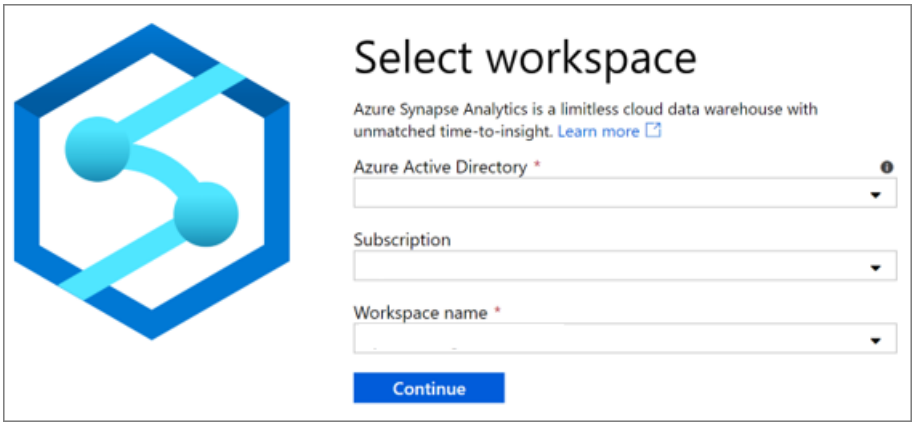
8/19/2022 • 2 minutes to read • Edit Online

With Synapse Studio, you can run SQL scripts on the SQL pools in your workspace.

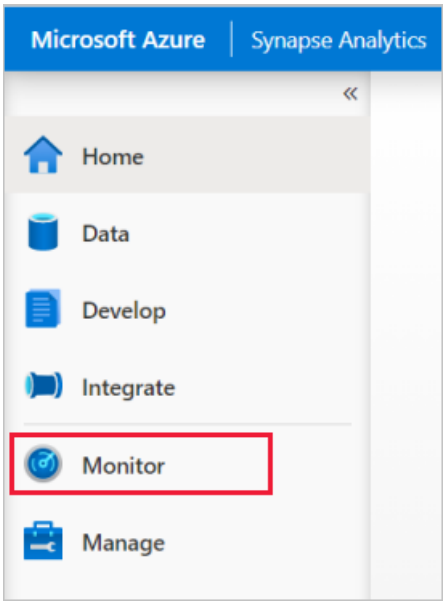
This article explains how to monitor your SQL requests, allowing you to keep an eye on the status of running requests and discover details of historical requests.

Access SQL requests list

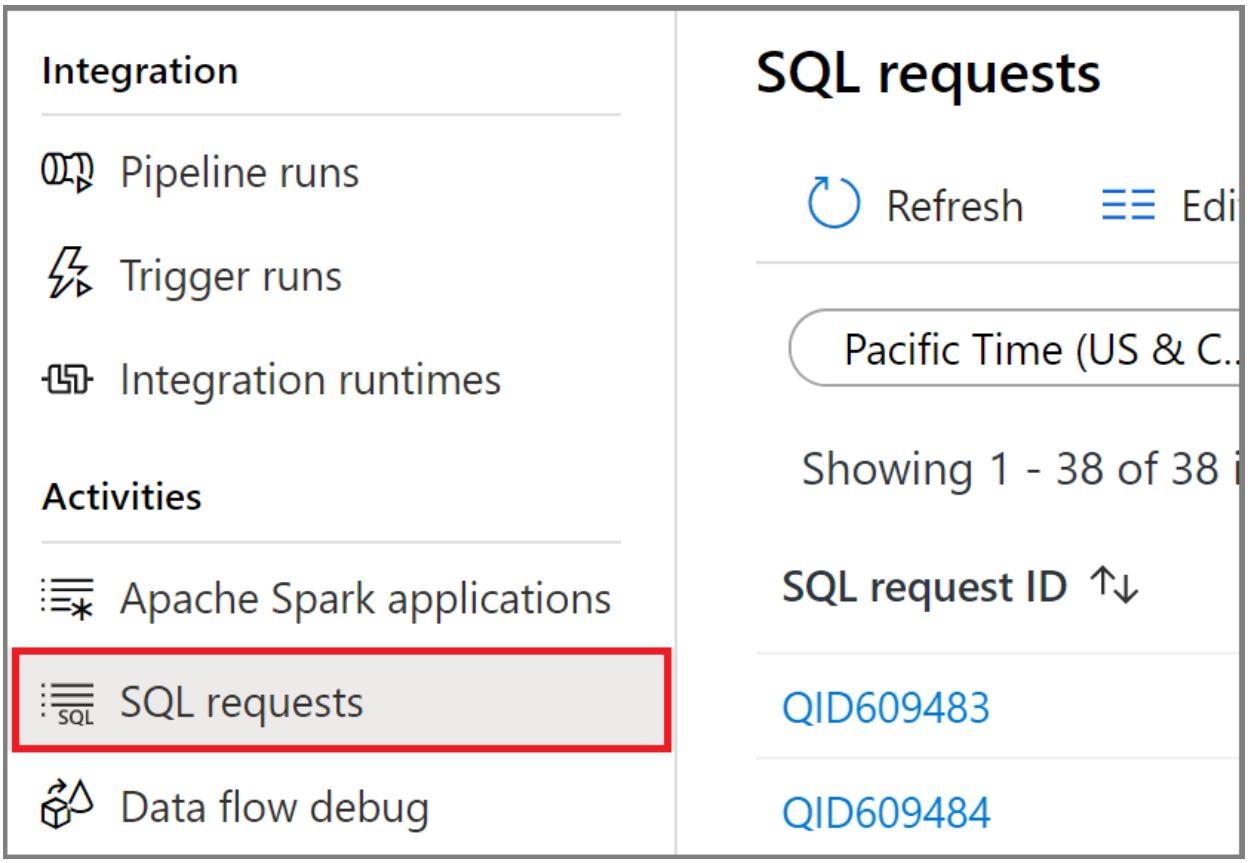
To see the list of SQL requests in your workspace, first open the Synapse Studio and select your workspace.



Once you've opened your workspace, select the **Monitor** section on the left.

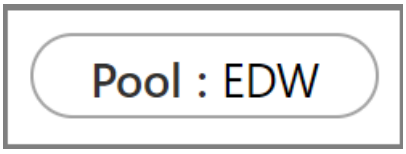


Select **SQL requests** to view the list of SQL requests.



Select a SQL pool

By default, the SQL request history for the built-in serverless SQL pool are shown in this view. You can select one of your dedicated SQL pools to see the SQL request history of that pool.



Filter your SQL requests

You can filter the list of SQL requests to the ones that interest you. The filters at the top of the screen allow you to specify a field on which you'd like to filter.

For example, you can filter the view to see only the SQL requests submitted by `maria@contoso.com`:

View details about a specific SQL request

To view the details about one of your SQL requests, open the SQL request to navigate to the details view. For the complex requests running on dedicated SQL pools, you can monitor the progress.

Next steps

For more information on monitoring pipeline runs, see the Monitor pipeline runs in Synapse Studio article.

For more information on monitoring Apache Spark applications, see the Monitor Apache Spark applications in

Synapse Studio article.

To analyze historical queries, see Historical query storage and analysis in Azure Synapse Analytics.

Use Synapse Studio to monitor your SQL pools

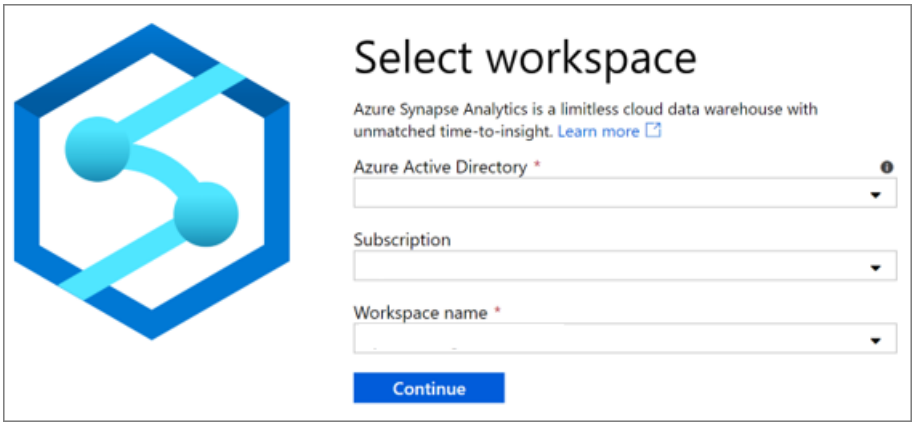
8/19/2022 • 2 minutes to read • Edit Online

With Synapse Studio, you can run SQL scripts on the SQL pools in your workspace.

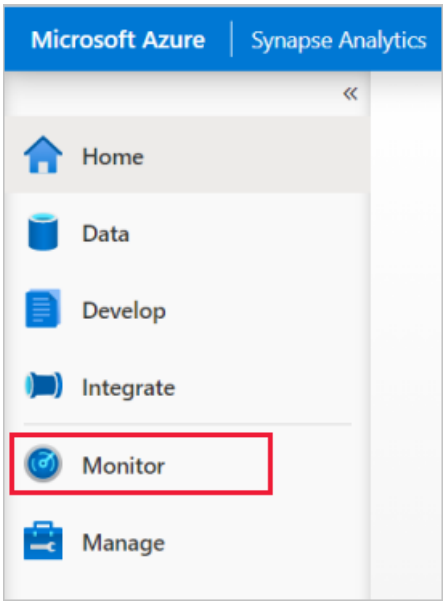
This article explains how to monitor your SQL pools, allowing you to keep an eye on the status and activity of your pools.

Access SQL pools list

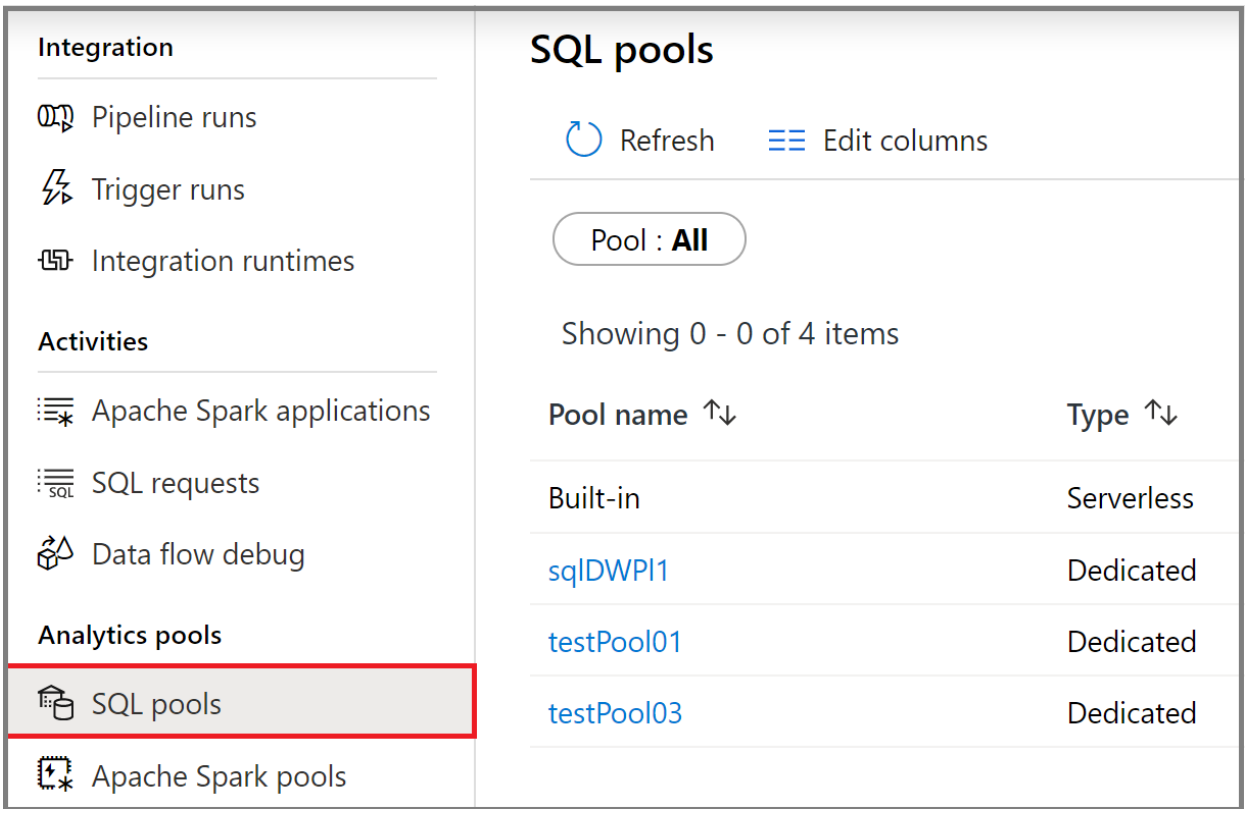
To see the list of SQL pools in your workspace, first open the Synapse Studio and select your workspace.



Once you've opened your workspace, select the **Monitor** section on the left.



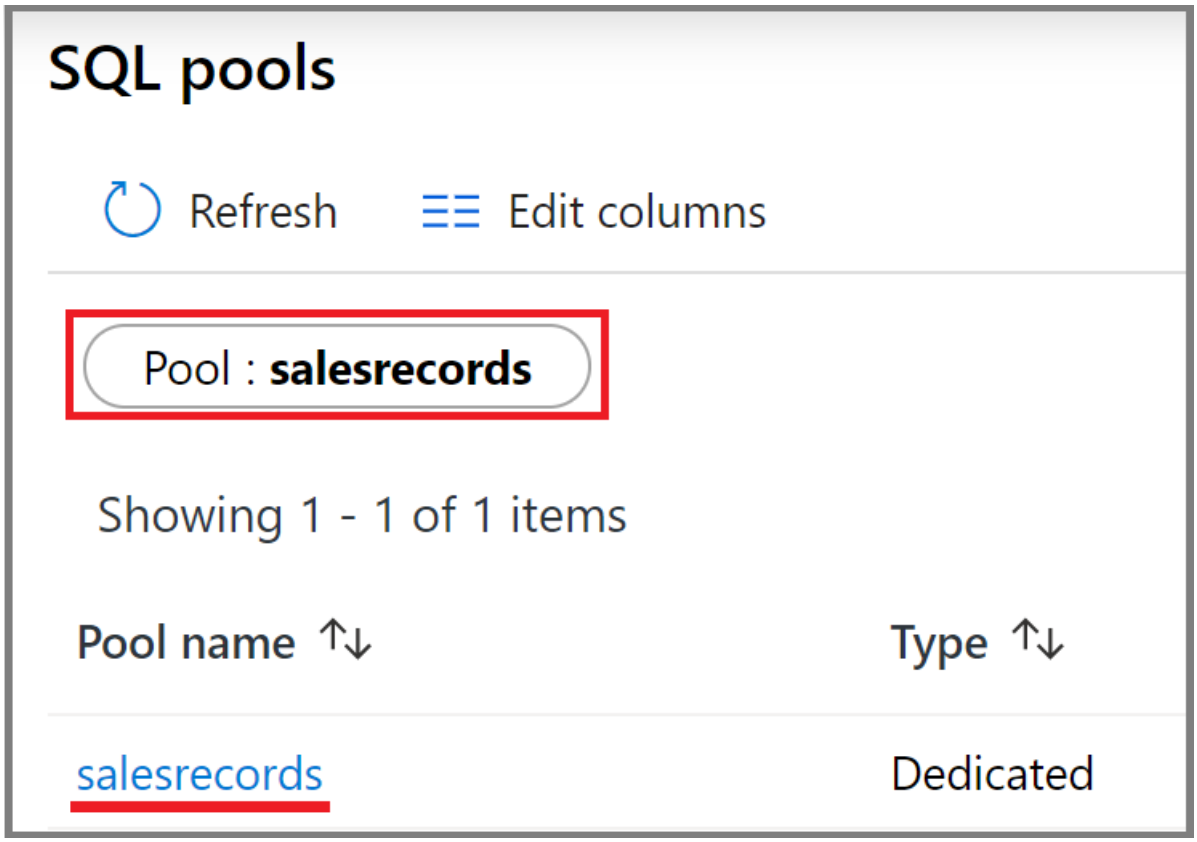
Select **SQL pools** to view the list of SQL pools.



Filter your SQL pools

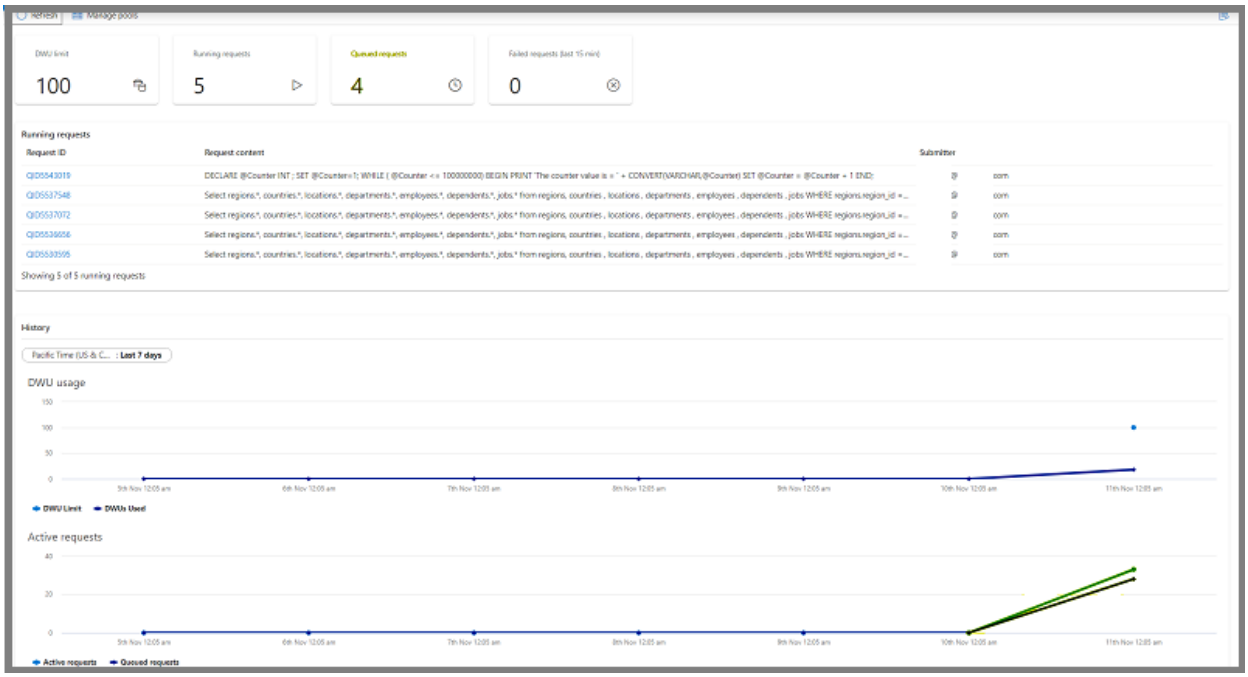
You can filter the list of SQL pools to the ones that interest you. The filters at the top of the screen allow you to specify a field on which you'd like to filter.

For example, you can filter the view to see only the SQL pools containing the name "salesrecords":



View details about a specific SQL pool

To view the details about one of your SQL pools, select the SQL pool to view the details.



Next steps

For more information on monitoring pipeline runs, see the Monitor pipeline runs in Synapse Studio article.

For more information on monitoring SQL requests, see the Monitor SQL requests in Synapse Studio article.

How to monitor the adaptive cache

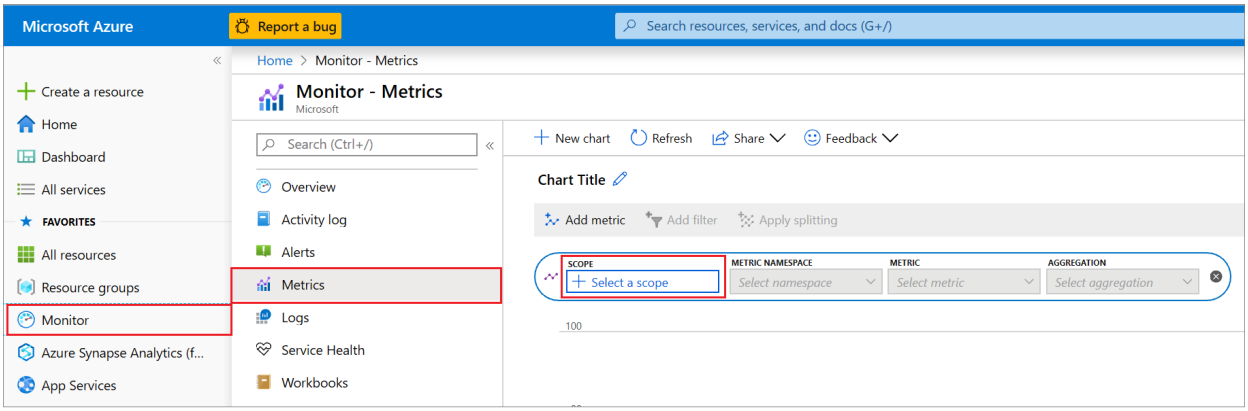
8/19/2022 • 2 minutes to read • Edit Online

This article describes how to monitor and troubleshoot slow query performance by determining whether your workload is optimally leveraging the adaptive cache for dedicated SQL pools.

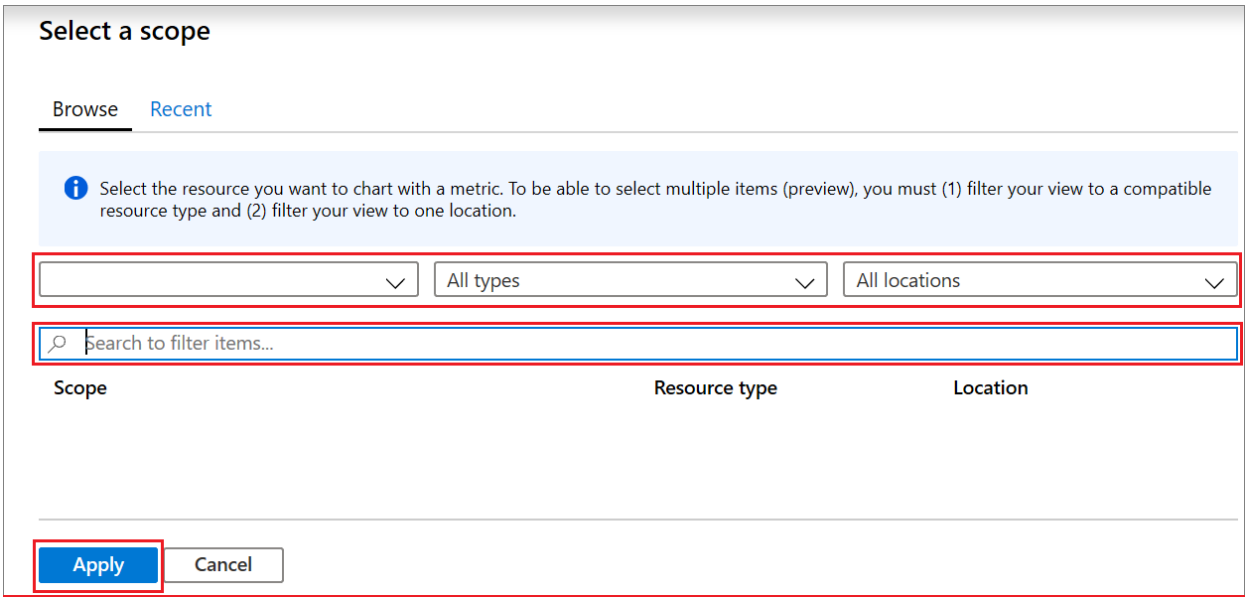
The dedicated SQL pool storage architecture automatically tiers your most frequently queried columnstore segments in a cache residing on NVMe based SSDs. You will have greater performance when your queries retrieve segments that are residing in the cache.

Troubleshoot using the Azure portal

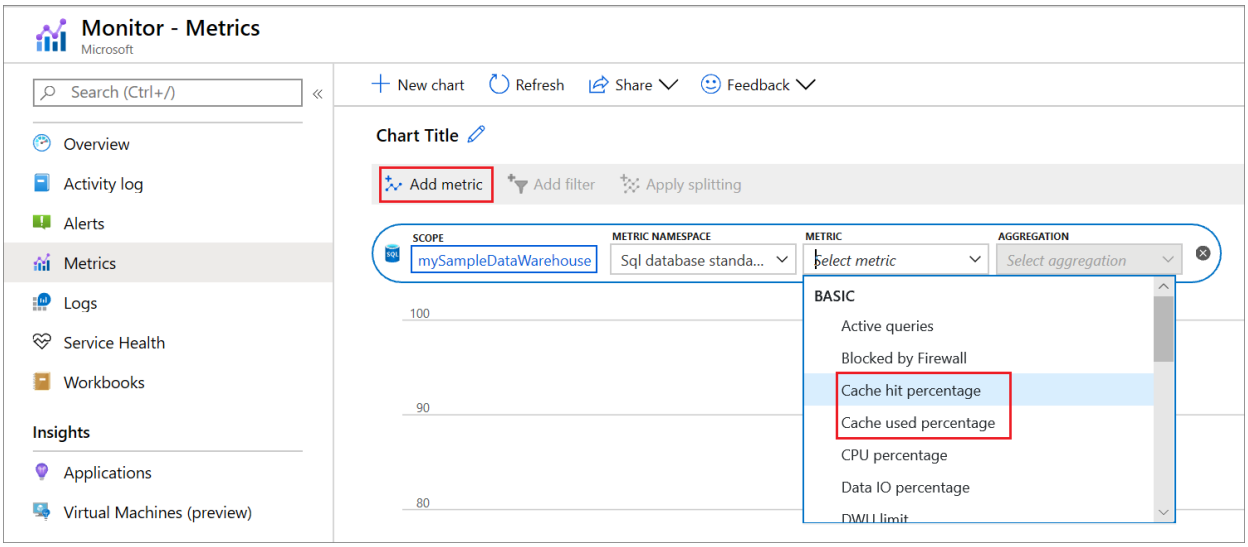
You can use Azure Monitor to view cache metrics to troubleshoot query performance. First go to the Azure portal and click on **Monitor**, **Metrics** and **+ Select a scope**:



Use the search and drop down bars to locate your dedicated SQL pool. Then select apply.



The key metrics for troubleshooting the cache are **Cache hit percentage** and **Cache used percentage**. Select **Cache hit percentage** then use the **add metric** button to add **Cache used percentage**.



Cache hit and used percentage

The matrix below describes scenarios based on the values of the cache metrics:

	HIGH CACHE HIT PERCENTAGE	LOW CACHE HIT PERCENTAGE
High Cache used percentage	Scenario 1	Scenario 2
Low Cache used percentage	Scenario 3	Scenario 4

Scenario 1: You are optimally using your cache. Troubleshoot other areas that may be slowing down your queries.

Scenario 2: Your current working data set cannot fit into the cache which causes a low cache hit percentage due to physical reads. Consider scaling up your performance level and rerun your workload to populate the cache.

Scenario 3: It is likely that your query is running slow due to reasons unrelated to the cache. Troubleshoot other areas that may be slowing down your queries. You can also consider scaling down your instance to reduce your cache size to save costs.

Scenario 4: You had a cold cache which could be the reason why your query was slow. Consider rerunning your query as your working dataset should now be in cached.

> **IMPORTANT**
> If the cache hit percentage or cache used percentage isn't updating after rerunning your workload, your working set can already be residing in memory. Only clustered columnstore tables are cached.

Next steps

For more information on general query performance tuning, see Monitor query execution.

Monitor your Azure Synapse Analytics dedicated SQL pool workload using DMVs

8/19/2022 • 9 minutes to read • Edit Online

This article describes how to use Dynamic Management Views (DMVs) to monitor your workload including investigating query execution in SQL pool.

Permissions

To query the DMVs in this article, you need either **VIEW DATABASE STATE** or **CONTROL** permission. Usually, granting **VIEW DATABASE STATE** is the preferred permission as it is much more restrictive.

```
GRANT VIEW DATABASE STATE TO myuser;
```

Monitor connections

All logins to your data warehouse are logged to sys.dm_pdw_exec_sessions. This DMV contains the last 10,000 logins. The session_id is the primary key and is assigned sequentially for each new logon.

```
-- Other Active Connections
SELECT * FROM sys.dm_pdw_exec_sessions where status <> 'Closed' and session_id <> session_id();
```

Monitor query execution

All queries executed on SQL pool are logged to sys.dm_pdw_exec_requests. This DMV contains the last 10,000 queries executed. The request_id uniquely identifies each query and is the primary key for this DMV. The request_id is assigned sequentially for each new query and is prefixed with QID, which stands for query ID. Querying this DMV for a given session_id shows all queries for a given logon.

> **NOTE**
> Stored procedures use multiple Request IDs. Request IDs are assigned in sequential order.

Here are steps to follow to investigate query execution plans and times for a particular query.

STEP 1: Identify the query you wish to investigate

```
-- Monitor active queries
SELECT *
FROM sys.dm_pdw_exec_requests
WHERE status not in ('Completed','Failed','Cancelled')
  AND session_id <> session_id()
ORDER BY submit_time DESC;

-- Find top 10 queries longest running queries
SELECT TOP 10 *
FROM sys.dm_pdw_exec_requests
ORDER BY total_elapsed_time DESC;
```

From the preceding query results, **note the Request ID** of the query that you would like to investigate.

Queries in the **Suspended** state can be queued due to a large number of active running queries. These queries also appear in the sys.dm_pdw_waits. In that case, look for waits such as UserConcurrencyResourceType. For information on concurrency limits, see Memory and concurrency limits or Resource classes for workload management. Queries can also wait for other reasons such as for object locks. If your query is waiting for a resource, see Investigating queries waiting for resources further down in this article.

To simplify the lookup of a query in the sys.dm_pdw_exec_requests table, use LABEL to assign a comment to your query, which can be looked up in the sys.dm_pdw_exec_requests view.

```
-- Query with Label
SELECT *
FROM sys.tables
OPTION (LABEL = 'My Query')
;

-- Find a query with the Label 'My Query'
-- Use brackets when querying the label column, as it it a key word
SELECT  *
FROM    sys.dm_pdw_exec_requests
WHERE   [label] = 'My Query';
```

STEP 2: Investigate the query plan

Use the Request ID to retrieve the query's distributed SQL (DSQL) plan from sys.dm_pdw_request_steps

```
-- Find the distributed query plan steps for a specific query.
-- Replace request_id with value from Step 1.

SELECT * FROM sys.dm_pdw_request_steps
WHERE request_id = 'QID####'
ORDER BY step_index;
```

When a DSQL plan is taking longer than expected, the cause can be a complex plan with many DSQL steps or just one step taking a long time. If the plan is many steps with several move operations, consider optimizing your table distributions to reduce data movement. The Table distribution article explains why data must be moved to solve a query. The article also explains some distribution strategies to minimize data movement.

To investigate further details about a single step, the *operation_type* column of the long-running query step and note the **Step Index**:

- Proceed with Step 3 for **SQL operations**: OnOperation, RemoteOperation, ReturnOperation.
- Proceed with Step 4 for **Data Movement operations**: ShuffleMoveOperation, BroadcastMoveOperation, TrimMoveOperation, PartitionMoveOperation, MoveOperation, CopyOperation.

STEP 3: Investigate SQL on the distributed databases

Use the Request ID and the Step Index to retrieve details from sys.dm_pdw_sql_requests, which contains execution information of the query step on all of the distributed databases.

```
-- Find the distribution run times for a SQL step.
-- Replace request_id and step_index with values from Step 1 and 3.

SELECT * FROM sys.dm_pdw_sql_requests
WHERE request_id = 'QID####' AND step_index = 2;
```

When the query step is running, DBCC PDW_SHOWEXECUTIONPLAN can be used to retrieve the SQL Server estimated plan from the SQL Server plan cache for the step running on a particular distribution.

```
-- Find the SQL Server execution plan for a query running on a specific SQL pool or control node.
-- Replace distribution_id and spid with values from previous query.

DBCC PDW_SHOWEXECUTIONPLAN(1, 78);
```

STEP 4: Investigate data movement on the distributed databases

Use the Request ID and the Step Index to retrieve information about a data movement step running on each distribution from sys.dm_pdw_dms_workers.

```
-- Find information about all the workers completing a Data Movement Step.
-- Replace request_id and step_index with values from Step 1 and 3.

SELECT * FROM sys.dm_pdw_dms_workers
WHERE request_id = 'QID####' AND step_index = 2;
```

- Check the *total_elapsed_time* column to see if a particular distribution is taking significantly longer than others for data movement.
- For the long-running distribution, check the *rows_processed* column to see if the number of rows being moved from that distribution is significantly larger than others. If so, this finding might indicate skew of your underlying data. One cause for data skew is distributing on a column with many NULL values (whose rows will all land in the same distribution). Prevent slow queries by avoiding distribution on these types of columns or filtering your query to eliminate NULLs when possible.

If the query is running, you can use DBCC PDW_SHOWEXECUTIONPLAN to retrieve the SQL Server estimated plan from the SQL Server plan cache for the currently running SQL Step within a particular distribution.

```
-- Find the SQL Server estimated plan for a query running on a specific SQL pool Compute or control node.
-- Replace distribution_id and spid with values from previous query.

DBCC PDW_SHOWEXECUTIONPLAN(55, 238);
```

Monitor waiting queries

If you discover that your query is not making progress because it is waiting for a resource, here is a query that shows all the resources a query is waiting for.

```
-- Find queries
-- Replace request_id with value from Step 1.

SELECT waits.session_id,
      waits.request_id,
      requests.command,
      requests.status,
      requests.start_time,
      waits.type,
      waits.state,
      waits.object_type,
      waits.object_name
FROM   sys.dm_pdw_waits waits
   JOIN  sys.dm_pdw_exec_requests requests
   ON waits.request_id=requests.request_id
WHERE waits.request_id = 'QID####'
ORDER BY waits.object_name, waits.object_type, waits.state;
```

If the query is actively waiting on resources from another query, then the state will be **AcquireResources**. If the

query has all the required resources, then the state will be **Granted**.

Monitor tempdb

Tempdb is used to hold intermediate results during query execution. High utilization of the tempdb database can lead to slow query performance. For every DW100c configured, 399 GB of tempdb space is allocated (DW1000c would have 3.99 TB of total tempdb space). Below are tips for monitoring tempdb usage and for decreasing tempdb usage in your queries.

Monitoring tempdb with views

To monitor tempdb usage, first install the microsoft.vw_sql_requests view from the Microsoft Toolkit for SQL pool. You can then execute the following query to see the tempdb usage per node for all executed queries:

```
-- Monitor tempdb
SELECT
    sr.request_id,
    ssu.session_id,
    ssu.pdw_node_id,
    sr.command,
    sr.total_elapsed_time,
    exs.login_name AS 'LoginName',
    DB_NAME(ssu.database_id) AS 'DatabaseName',
    (es.memory_usage * 8) AS 'MemoryUsage (in KB)',
    (ssu.user_objects_alloc_page_count * 8) AS 'Space Allocated For User Objects (in KB)',
    (ssu.user_objects_dealloc_page_count * 8) AS 'Space Deallocated For User Objects (in KB)',
    (ssu.internal_objects_alloc_page_count * 8) AS 'Space Allocated For Internal Objects (in KB)',
    (ssu.internal_objects_dealloc_page_count * 8) AS 'Space Deallocated For Internal Objects (in KB)',
    CASE es.is_user_process
    WHEN 1 THEN 'User Session'
    WHEN 0 THEN 'System Session'
    END AS 'SessionType',
    es.row_count AS 'RowCount'
FROM sys.dm_pdw_nodes_db_session_space_usage AS ssu
    INNER JOIN sys.dm_pdw_nodes_exec_sessions AS es ON ssu.session_id = es.session_id AND ssu.pdw_node_id =
es.pdw_node_id
    INNER JOIN sys.dm_pdw_nodes_exec_connections AS er ON ssu.session_id = er.session_id AND ssu.pdw_node_id
= er.pdw_node_id
    INNER JOIN microsoft.vw_sql_requests AS sr ON ssu.session_id = sr.spid AND ssu.pdw_node_id =
sr.pdw_node_id
    LEFT JOIN sys.dm_pdw_exec_requests exr on exr.request_id = sr.request_id
    LEFT JOIN sys.dm_pdw_exec_sessions exs on exr.session_id = exs.session_id
WHERE DB_NAME(ssu.database_id) = 'tempdb'
    AND es.session_id <> @@SPID
    AND es.login_name <> 'sa'
ORDER BY sr.request_id;
```

If you have a query that is consuming a large amount of memory or have received an error message related to the allocation of tempdb, it could be due to a very large CREATE TABLE AS SELECT (CTAS) or INSERT SELECT statement running that is failing in the final data movement operation. This can usually be identified as a ShuffleMove operation in the distributed query plan right before the final INSERT SELECT. Use sys.dm_pdw_request_steps to monitor ShuffleMove operations.

The most common mitigation is to break your CTAS or INSERT SELECT statement into multiple load statements so the data volume will not exceed the 2TB per node tempdb limit (when at or above DW500c). You can also scale your cluster to a larger size which will spread the tempdb size across more nodes reducing the tempdb on each individual node.

In addition to CTAS and INSERT SELECT statements, large, complex queries running with insufficient memory can spill into tempdb causing queries to fail. Consider running with a larger resource class to avoid spilling into tempdb.

Monitor memory

Memory can be the root cause for slow performance and out of memory issues. Consider scaling your data warehouse if you find SQL Server memory usage reaching its limits during query execution.

The following query returns SQL Server memory usage and memory pressure per node:

```
-- Memory consumption
SELECT
  pc1.cntr_value as Curr_Mem_KB,
  pc1.cntr_value/1024.0 as Curr_Mem_MB,
  (pc1.cntr_value/1048576.0) as Curr_Mem_GB,
  pc2.cntr_value as Max_Mem_KB,
  pc2.cntr_value/1024.0 as Max_Mem_MB,
  (pc2.cntr_value/1048576.0) as Max_Mem_GB,
  pc1.cntr_value * 100.0/pc2.cntr_value AS Memory_Utilization_Percentage,
  pc1.pdw_node_id
FROM
-- pc1: current memory
sys.dm_pdw_nodes_os_performance_counters AS pc1
-- pc2: total memory allowed for this SQL instance
JOIN sys.dm_pdw_nodes_os_performance_counters AS pc2
ON pc1.object_name = pc2.object_name AND pc1.pdw_node_id = pc2.pdw_node_id
WHERE
pc1.counter_name = 'Total Server Memory (KB)'
AND pc2.counter_name = 'Target Server Memory (KB)'
```

Monitor transaction log size

The following query returns the transaction log size on each distribution. If one of the log files is reaching 160 GB, you should consider scaling up your instance or limiting your transaction size.

```
-- Transaction log size
SELECT
  instance_name as distribution_db,
  cntr_value*1.0/1048576 as log_file_size_used_GB,
  pdw_node_id
FROM sys.dm_pdw_nodes_os_performance_counters
WHERE
instance_name like 'Distribution_%'
AND counter_name = 'Log File(s) Used Size (KB)'
```

Monitor transaction log rollback

If your queries are failing or taking a long time to proceed, you can check and monitor if you have any transactions rolling back.

```
-- Monitor rollback
SELECT
    SUM(CASE WHEN t.database_transaction_next_undo_lsn IS NOT NULL THEN 1 ELSE 0 END),
    t.pdw_node_id,
    nod.[type]
FROM sys.dm_pdw_nodes_tran_database_transactions t
JOIN sys.dm_pdw_nodes nod ON t.pdw_node_id = nod.pdw_node_id
GROUP BY t.pdw_node_id, nod.[type]
```

Monitor PolyBase load

The following query provides an approximate estimate of the progress of your load. The query only shows files currently being processed.

```
-- To track bytes and files
SELECT
    r.command,
    s.request_id,
    r.status,
    count(distinct input_name) as nbr_files,
    sum(s.bytes_processed)/1024/1024/1024 as gb_processed
FROM
    sys.dm_pdw_exec_requests r
    inner join sys.dm_pdw_dms_external_work s
        on r.request_id = s.request_id
GROUP BY
    r.command,
    s.request_id,
    r.status
ORDER BY
    nbr_files desc,
    gb_processed desc;
```

Monitor query blockings

The following query provides the top 500 blocked queries in the environment.

```
--Collect the top blocking
SELECT
    TOP 500 waiting.request_id AS WaitingRequestId,
    waiting.object_type AS LockRequestType,
    waiting.object_name AS ObjectLockRequestName,
    waiting.request_time AS ObjectLockRequestTime,
    blocking.session_id AS BlockingSessionId,
    blocking.request_id AS BlockingRequestId
FROM
    sys.dm_pdw_waits waiting
    INNER JOIN sys.dm_pdw_waits blocking
    ON waiting.object_type = blocking.object_type
    AND waiting.object_name = blocking.object_name
WHERE
    waiting.state = 'Queued'
    AND blocking.state = 'Granted'
ORDER BY
    ObjectLockRequestTime ASC;
```

Retrieve query text from waiting and blocking queries

The following query provides the query text and identifier for the waiting and blocking queries to easily troubleshoot.

```
-- To retrieve query text from waiting and blocking queries

SELECT waiting.session_id AS WaitingSessionId,
       waiting.request_id AS WaitingRequestId,
       COALESCE(waiting_exec_request.command,waiting_exec_request.command2) AS WaitingExecRequestText,
       blocking.session_id AS BlockingSessionId,
       blocking.request_id AS BlockingRequestId,
       COALESCE(blocking_exec_request.command,blocking_exec_request.command2) AS BlockingExecRequestText,
       waiting.object_name AS Blocking_Object_Name,
       waiting.object_type AS Blocking_Object_Type,
       waiting.type AS Lock_Type,
       waiting.request_time AS Lock_Request_Time,
       datediff(ms, waiting.request_time, getdate())/1000.0 AS Blocking_Time_sec
FROM sys.dm_pdw_waits waiting
       INNER JOIN sys.dm_pdw_waits blocking
       ON waiting.object_type = blocking.object_type
       AND waiting.object_name = blocking.object_name
       INNER JOIN sys.dm_pdw_exec_requests blocking_exec_request
       ON blocking.request_id = blocking_exec_request.request_id
       INNER JOIN sys.dm_pdw_exec_requests waiting_exec_request
       ON waiting.request_id = waiting_exec_request.request_id
WHERE waiting.state = 'Queued'
       AND blocking.state = 'Granted'
ORDER BY Lock_Request_Time DESC;
```

Next steps

For more information about DMVs, see System views.

Monitor workload - Azure portal

8/19/2022 • 2 minutes to read • Edit Online

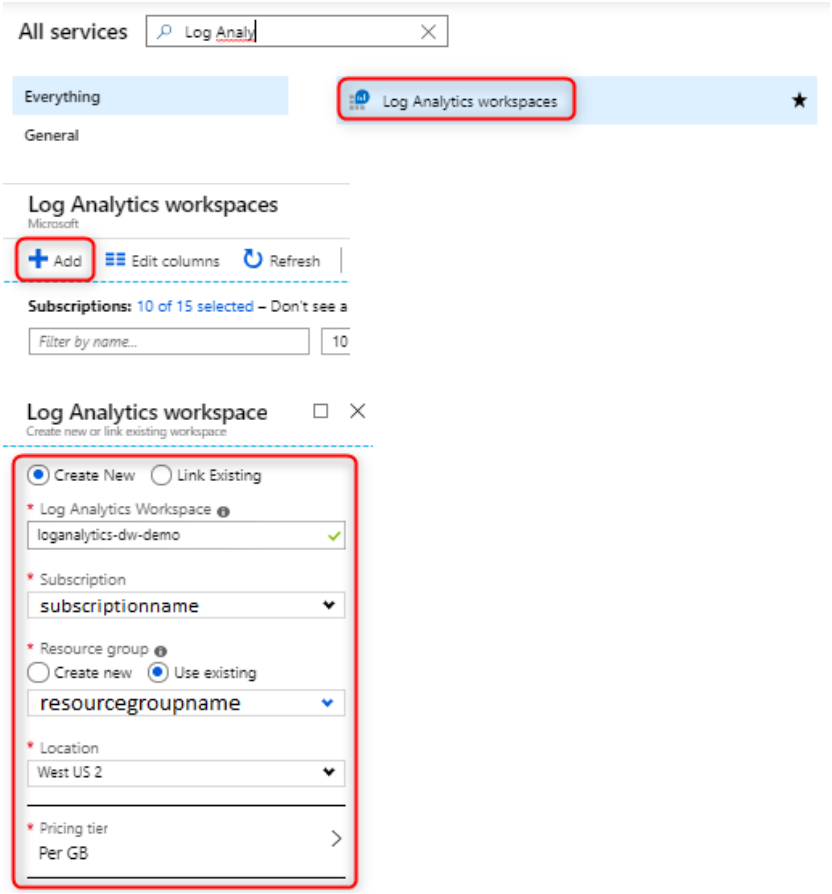
This article describes how to use the Azure portal to monitor your workload. This includes setting up Azure Monitor Logs to investigate query execution and workload trends using log analytics for Synapse SQL.

Prerequisites

- Azure subscription: If you don't have an Azure subscription, create a free account before you begin.
- SQL pool: We will be collecting logs for a SQL pool. If you don't have a SQL pool provisioned, see the instructions in Create a SQL pool.

Create a Log Analytics workspace

Navigate to the browse blade for Log Analytics workspaces and create a workspace



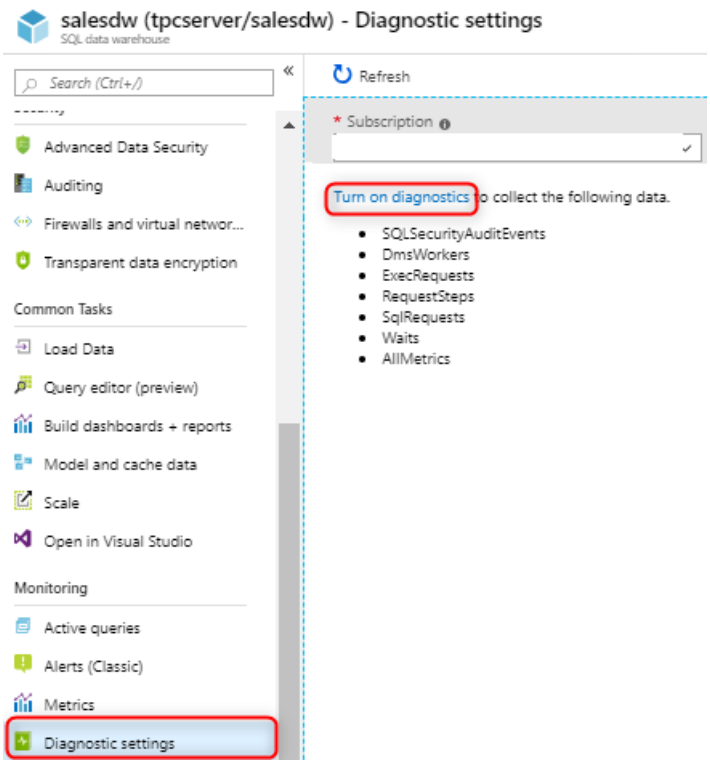
For more details on workspaces, visit the following documentation.

Turn on Resource logs

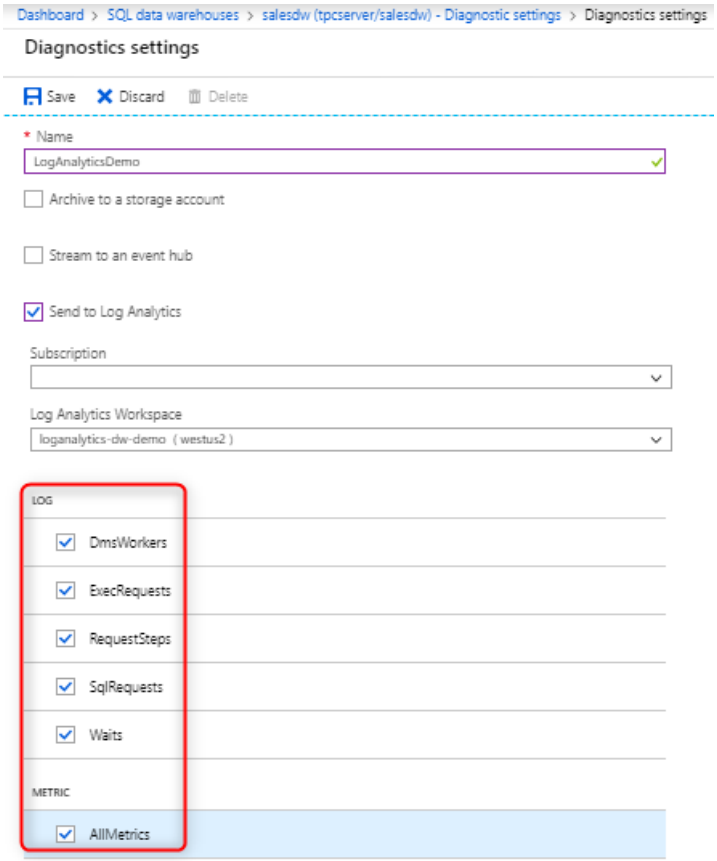
Configure diagnostic settings to emit logs from your SQL pool. Logs consist of telemetry views equivalent to the most commonly used performance troubleshooting DMVs. Currently the following views are supported:

- sys.dm_pdw_exec_requests
- sys.dm_pdw_request_steps
- sys.dm_pdw_dms_workers

- sys.dm_pdw_waits
- sys.dm_pdw_sql_requests



Logs can be emitted to Azure Storage, Stream Analytics, or Log Analytics. For this tutorial, select Log Analytics.



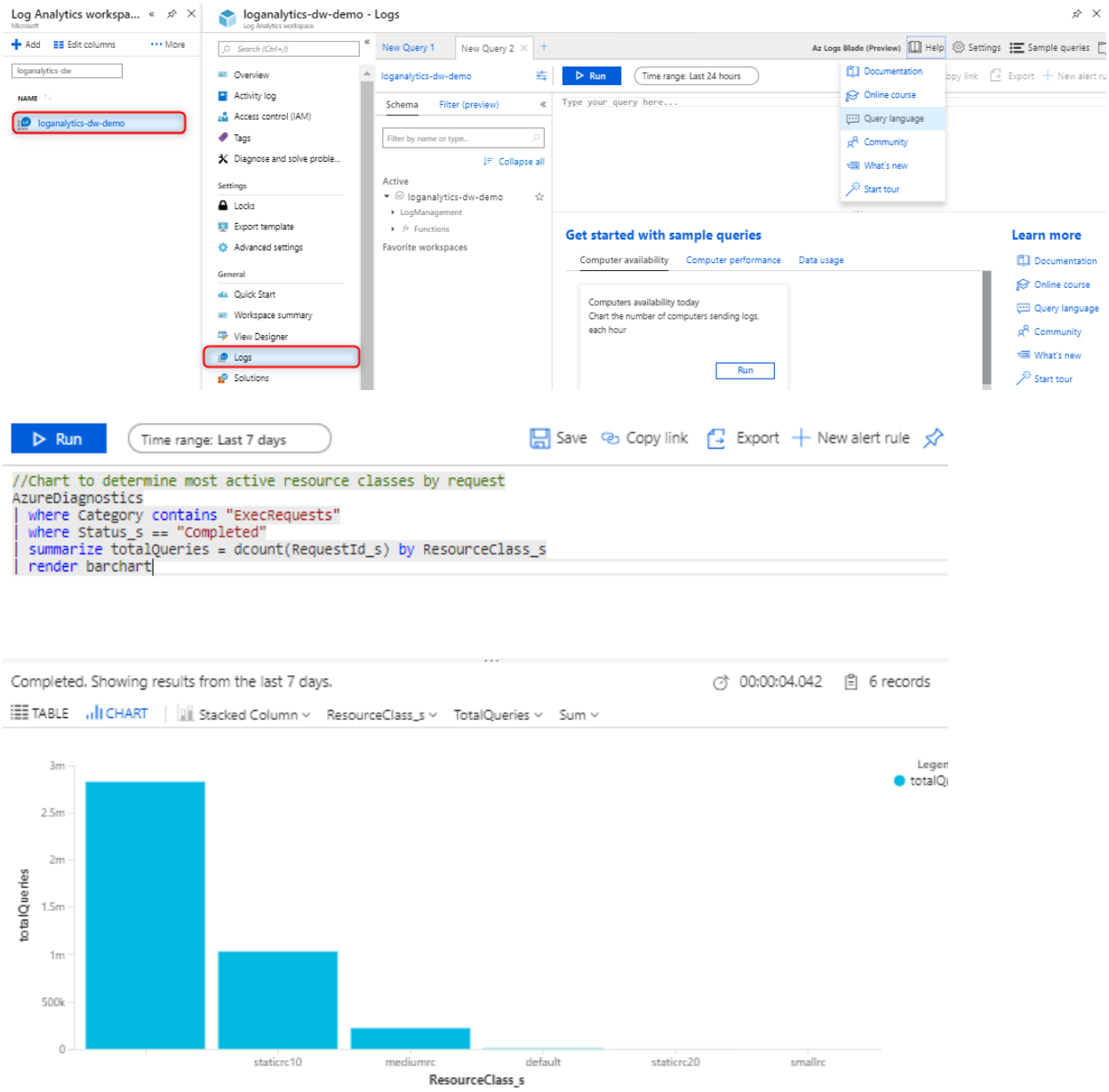
Run queries against Log Analytics

Navigate to your Log Analytics workspace where you can do the following:

- Analyze logs using log queries and save queries for reuse
- Save queries for reuse

- Create log alerts
- Pin query results to a dashboard

For details on the capabilities of log queries, visit the following documentation.



Sample log queries

```
//List all queries
AzureDiagnostics
| where Category contains "ExecRequests"
| project TimeGenerated, StartTime_t, EndTime_t, Status_s, Command_s, ResourceClass_s,
duration=datetime_diff('millisecond',EndTime_t, StartTime_t)
```

```
//Chart the most active resource classes
AzureDiagnostics
| where Category contains "ExecRequests"
| where Status_s == "Completed"
| summarize totalQueries = dcount(RequestId_s) by ResourceClass_s
| render barchart
```

```
//Count of all queued queries
AzureDiagnostics
| where Category contains "waits"
| where Type_s == "UserConcurrencyResourceType"
| summarize totalQueuedQueries = dcount(RequestId_s)
```

Next steps

Now that you have set up and configured Azure monitor logs, customize Azure dashboards to share across your team.

Historical query storage and analysis in Azure Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

Historic query analysis is one of the crucial needs of data engineers. Azure Synapse Analytics supports four main ways to analyze query history and performance. These include Query Store, DMVs, Azure Log Analytics, and Azure Data Explorer.

This article will show you how to use each of these options for your needs. Review use cases when it comes to analyzing query history, and the best method for each.

CUSTOMER NEED	QUERY STORE	DMVS	AZURE LOG ANALYTICS	AZURE DATA EXPLORER
Out of the box solution	Needs enabling	✔	Addition service required	Addition service required
Longer analysis periods	30 days	Up to 10000 rows of history	Customizable	Customizable
Crucial metrics availability	Limited	✔	Limited	Customizable
Use SQL for analysis	✔	✔	KQL needed	SQL support is limited

Query Store

The Query Store feature provides insight on query plan choice and performance. It simplifies performance troubleshooting by helping you quickly find performance differences caused by query plan changes.

Query Store is not enabled by default for new Azure Synapse Analytics databases. To enable Query Store to run the following T-SQL command:

```
ALTER DATABASE <database_name>
SET QUERY_STORE = ON;
```

For example:

```
ALTER DATABASE [SQLPOOL1]
SET QUERY_STORE = ON;
```

You can run performance auditing and troubleshooting related tasks by finding last executed queries, execution counts, longest running queries, queries with maximum physical I/O leads. Please refer to Monitoring Performance By Using the Query Store for sample queries.

Advantages:

- Up to 30 days of storage for query data. Default 7 days.

- Data can be consumed in the same tool that you'd run the query in.

Known Limitation:

- Default storage of historic query data is less.
- Scenarios for analysis are limited in Query Store for Azure Synapse when compared to using DMVs.

DMVs

Dynamic Management Views (DMVs) are extremely useful when it comes to gathering information on query wait times, execution plans, memory, etc. It is highly recommended to label your query of interest to track it down later. For example:

```
-- Query with Label
SELECT *
FROM sys.tables
OPTION (LABEL = 'My Query');
```

For more information on labeling your queries in Azure Synapse SQL, see Use query labels in Synapse SQL.

For more information on using DMVs to monitor your Azure Synapse Analytics workload, see Monitor your dedicated SQL pool workload using DMVs. For documentation on catalog views specific to Azure Synapse Analytics, see Azure Synapse Analytics Catalog Views.

Advantages:

- Data can be consumed in the same querying tool.
- DMVs provide extensive options for analysis.

Known Limitations:

- DMVs are limited to 10,000 rows of historic entries.
- Views are reset when pool is paused/resumed.

Log Analytics

Log Analytics workspaces can be created easily in the Azure portal. For further instructions on how to connect Synapse with Log Analytics, see Monitor workload - Azure portal.

Like Azure Data Explorer, Log Analytics uses the Kusto Query Language (KQL). For more information about Kusto syntax, see Kusto query overview.

Along with configurable retention period, you choose the workspace you are specifically targeting to query in Log Analytics. Log Analytics gives you the flexibility to store data, run, and save queries.

Advantages:

- Azure Log Analytics has a customizable log retention policy

Known Limitations:

- Using KQL adds to the learning curve.
- Limited views can be logged out of the box.

Azure Data Explorer (ADX)

Azure Data Explorer (ADX) is a leading data exploration service. This service can be used to analyze historic queries from Azure Synapse Analytics. To setup an Azure Data Factory (ADF) pipeline to copy and store logs to

ADX, see Copy data to or from Azure Data Explorer. In ADX, you can run performant Kusto query to analyze your logs. You can combine other strategies here, for example to query and load DMV output to ADX via ADF.

Advantages:

- ADX provides a customizable log retention policy.
- Performant query execution against large amount of data, especially queries involving string search.

Known Limitation:

- Using KQL adds to the learning curve.

Next steps

- Azure Data Explorer
- Azure Data Factory
- Log Analytics in Azure Monitor
- Azure Synapse SQL architecture
- Get Started with Azure Synapse Analytics

Analyze your workload for dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

Techniques for analyzing your dedicated SQL pool workload in Azure Synapse Analytics.

Resource Classes

Dedicated SQL pool provides resource classes to assign system resources to queries. For more information on resource classes, see Resource classes & workload management. Queries will wait if the resource class assigned to a query needs more resources than are currently available.

Queued query detection and other DMVs

You can use the `sys.dm_pdw_exec_requests` DMV to identify queries that are waiting in a concurrency queue. Queries waiting for a concurrency slot have a status of **suspended**.

```
SELECT  r.[request_id]                          AS Request_ID
,       r.[status]                              AS Request_Status
,       r.[submit_time]                         AS Request_SubmitTime
,       r.[start_time]                          AS Request_StartTime
,       DATEDIFF(ms,[submit_time],[start_time]) AS Request_InitiateDuration_ms
,       r.resource_class                        AS Request_resource_class
FROM    sys.dm_pdw_exec_requests r
;
```

Workload management roles can be viewed with `sys.database_principals`.

```
SELECT  ro.[name]           AS [db_role_name]
FROM    sys.database_principals ro
WHERE   ro.[type_desc]      = 'DATABASE_ROLE'
AND     ro.[is_fixed_role]  = 0
;
```

The following query shows which role each user is assigned to.

```
SELECT  r.name AS role_principal_name
,       m.name AS member_principal_name
FROM    sys.database_role_members rm
JOIN    sys.database_principals AS r            ON rm.role_principal_id      = r.principal_id
JOIN    sys.database_principals AS m            ON rm.member_principal_id    = m.principal_id
WHERE   r.name IN ('mediumrc','largerc','xlargerc')
;
```

Dedicated SQL pool has the following wait types:

- **LocalQueriesConcurrencyResourceType**: Queries that sit outside of the concurrency slot framework. DMV queries and system functions such as `SELECT @@VERSION` are examples of local queries.
- **UserConcurrencyResourceType**: Queries that sit inside the concurrency slot framework. Queries against end-user tables represent examples that would use this resource type.
- **DmsConcurrencyResourceType**: Waits resulting from data movement operations.

- **BackupConcurrencyResourceType**: This wait indicates that a database is being backed up. The maximum value for this resource type is 1. If multiple backups have been requested at the same time, the others queue. In general, we recommend a minimum time between consecutive snapshots of 10 minutes.

The `sys.dm_pdw_waits` DMV can be used to see which resources a request is waiting for.

```
SELECT  w.[wait_id]
,       w.[session_id]
,       w.[type]                                        AS Wait_type
,       w.[object_type]
,       w.[object_name]
,       w.[request_id]
,       w.[request_time]
,       w.[acquire_time]
,       w.[state]
,       w.[priority]
,       SESSION_ID()                                    AS Current_session
,       s.[status]                                      AS Session_status
,       s.[login_name]
,       s.[query_count]
,       s.[client_id]
,       s.[sql_spid]
,       r.[command]                                     AS Request_command
,       r.[label]
,       r.[status]                                      AS Request_status
,       r.[submit_time]
,       r.[start_time]
,       r.[end_compile_time]
,       r.[end_time]
,       DATEDIFF(ms,r.[submit_time],r.[start_time])     AS Request_queue_time_ms
,       DATEDIFF(ms,r.[start_time],r.[end_compile_time])  AS Request_compile_time_ms
,       DATEDIFF(ms,r.[end_compile_time],r.[end_time])   AS Request_execution_time_ms
,       r.[total_elapsed_time]
FROM    sys.dm_pdw_waits w
JOIN    sys.dm_pdw_exec_sessions s  ON w.[session_id] = s.[session_id]
JOIN    sys.dm_pdw_exec_requests r  ON w.[request_id] = r.[request_id]
WHERE    w.[session_id] <> SESSION_ID();
```

The `sys.dm_pdw_resource_waits` DMV shows the wait information for a given query. Resource wait time measures the time waiting for resources to be provided. Signal wait time is the time it takes for the underlying SQL servers to schedule the query onto the CPU.

```
SELECT  [session_id]
,       [type]
,       [object_type]
,       [object_name]
,       [request_id]
,       [request_time]
,       [acquire_time]
,       DATEDIFF(ms,[request_time],[acquire_time])  AS acquire_duration_ms
,       [concurrency_slots_used]                    AS concurrency_slots_reserved
,       [resource_class]
,       [wait_id]                                   AS queue_position
FROM    sys.dm_pdw_resource_waits
WHERE    [session_id] <> SESSION_ID();
```

You can also use the `sys.dm_pdw_resource_waits` DMV calculate how many concurrency slots have been granted.

```
SELECT  SUM([concurrency_slots_used]) as total_granted_slots
FROM    sys.[dm_pdw_resource_waits]
WHERE   [state]           = 'Granted'
AND     [resource_class] is not null
AND     [session_id]     <> session_id();
```

The `sys.dm_pdw_wait_stats` DMV can be used for historic trend analysis of waits.

```
SELECT   w.[pdw_node_id]
,        w.[wait_name]
,        w.[max_wait_time]
,        w.[request_count]
,        w.[signal_time]
,        w.[completed_count]
,        w.[wait_time]
FROM    sys.dm_pdw_wait_stats w;
```

Next steps

For more information about managing database users and security, see Secure a dedicated SQL pool (formerly SQL DW). For more information about how larger resource classes can improve clustered columnstore index quality, see Rebuilding indexes to improve segment quality.

Configure workload importance in dedicated SQL pool for Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

Setting importance in dedicated SQL pool for Azure Synapse allows you to influence the scheduling of queries. Queries with higher importance will be scheduled to run before queries with lower importance. To assign importance to queries, you need to create a workload classifier.

Create a Workload Classifier with Importance

Often in a data warehouse scenario you have users, on a busy system, who need to run their queries quickly. The user could be executives of the company who need to run reports or the user could be an analyst running an adhoc query. To assign importance, you create a workload classifier and importance is assigned to a query. The examples below use the create workload classifier syntax to create two classifiers. `Membername` can be a single user or a group. To find existing dedicated SQL pool users, run:

```
Select name from sys.sysusers
```

To create a workload classifier, for a user with higher importance run:

```
CREATE WORKLOAD CLASSIFIER ExecReportsClassifier
    WITH (WORKLOAD_GROUP = 'xlargerc'
         ,MEMBERNAME     = 'name'
         ,IMPORTANCE     = above_normal);
```

To create a workload classifier for a user running adhoc queries with lower importance run:

```
CREATE WORKLOAD CLASSIFIER AdhocClassifier
    WITH (WORKLOAD_GROUP = 'xlargerc'
         ,MEMBERNAME     = 'name'
         ,IMPORTANCE     = below_normal);
```

Next Steps

- For more information about workload management, see Workload Classification
- For more information on Importance, see Workload Importance

Go to Manage and monitor Workload Importance

Manage and monitor workload importance in dedicated SQL pool for Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

Manage and monitor dedicated SQL pool request level importance in Azure Synapse using DMVs and catalog views.

Monitor importance

Monitor importance using the new importance column in the sys.dm_pdw_exec_requests dynamic management view. The below monitoring query shows submit time and start time for queries. Review the submit time and start time along with importance to see how importance influenced scheduling.

```
SELECT s.login_name, r.status, r.importance, r.submit_time, r.start_time
  FROM sys.dm_pdw_exec_sessions s
  JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
  WHERE r.resource_class is not null
ORDER BY r.start_time
```

To look further into how queries are being schedule, use the catalog views.

Manage importance with catalog views

The sys.workload_management_workload_classifiers catalog view contains information on classifiers. To exclude the system-defined classifiers that map to resource classes execute the following code:

```
SELECT *
  FROM sys.workload_management_workload_classifiers
  WHERE classifier_id > 12
```

The catalog view, sys.workload_management_workload_classifier_details, contains information on the parameters used in creation of the classifier. The below query shows that ExecReportsClassifier was created on the `membername` parameter for values with ExecutiveReports:

```
SELECT c.name,cd.classifier_type, classifier_value
  FROM sys.workload_management_workload_classifiers c
  JOIN sys.workload_management_workload_classifier_details cd
    ON cd.classifier_id = c.classifier_id
  WHERE c.name = 'ExecReportsClassifier'
```

Results | Messages

	name	classifier_type	classifier_value
1	ExecReportsClassifier	membername	ExecutiveReports

To simplify troubleshooting misclassification, we recommended you remove resource class role mappings as you create workload classifiers. The code below returns existing resource class role memberships. Run sp_droprolemember for each `membername` returned from the corresponding resource class. Below is an example of checking for existence before dropping a workload classifier:

```
IF EXISTS (SELECT 1 FROM sys.workload_management_workload_classifiers WHERE name = 'ExecReportsClassifier')
  DROP WORKLOAD CLASSIFIER ExecReportsClassifier;
GO
```

Next steps

- For more information on Classification, see Workload Classification.
- For more information on Importance, see Workload Importance

Go to Configure Workload Importance

Convert Resource Classes to Workload Groups

8/19/2022 • 2 minutes to read • Edit Online

Workload groups provide a mechanism to isolate and contain system resources. Additionally, workload groups allow you to set execution rules for the requests running in them. A query timeout execution rule allows runaway queries to be canceled without user intervention. This article explains how to take an existing resource class and create a workload group with a similar configuration. In addition, an optional query timeout rule is added.

> **NOTE**
>
> See the Mixing resource class assignments with classifiers section in the Workload Classification concept document for guidance on using workload groups and resource classes at the same time.

Understanding the existing resource class configuration

Workload groups require a parameter called `REQUEST_MIN_RESOURCE_GRANT_PERCENT` that specifies the percentage of overall system resources allocated per request. Resource allocation is done for resource classes by allocating concurrency slots. To determine the value to specify for `REQUEST_MIN_RESOURCE_GRANT_PERCENT`, use the sys.dm_workload_management_workload_groups_stats DMV. For example, the below query query returns a value that can be used for the `REQUEST_MIN_RESOURCE_GRANT_PERCENT` parameter to create a workload group similar to staticrc40.

```
SELECT Request_min_resource_grant_percent = Effective_request_min_resource_grant_percent
  FROM sys.dm_workload_management_workload_groups_stats
  WHERE name = 'staticrc40'
```

> **NOTE**
>
> Workload groups operate based on percentage of overall system resources.

Because workload groups operate based on percentage of overall system resources, as you scale up and down, the percentage of resources allocated to static resource classes relative to the overall system resources changes. For example, staticrc40 at DW1000c allocates 19.2% of the overall system resources. At DW2000c, 9.6% are allocated. This model is similar if you wish to scale up for concurrency versus allocating more resources per request.

Create Workload Group

With the known `REQUEST_MIN_RESOURCE_GRANT_PERCENT`, you can use the CREATE WORKLOAD GROUP syntax to create the workload group. You can optionally specify a `MIN_PERCENTAGE_RESOURCE` that is greater than zero to isolate resources for the workload group. Also, you can optionally specify `CAP_PERCENTAGE_RESOURCE` less than 100 to limit the amount of resources the workload group can consume.

Using mediumrc as a basis for an example, the below code sets the `MIN_PERCENTAGE_RESOURCE` to dedicate 10% of the system resources to `wgDataLoads` and guarantees one query will be able to run all the times. Additionally, `CAP_PERCENTAGE_RESOURCE` is set to 40% and limits this workload group to four concurrent requests. By setting the `QUERY_EXECUTION_TIMEOUT_SEC` parameter to 3600, any query that runs for more than 1 hour will be automatically

canceled.

```
CREATE WORKLOAD GROUP wgDataLoads WITH
( REQUEST_MIN_RESOURCE_GRANT_PERCENT = 10
 ,MIN_PERCENTAGE_RESOURCE = 10
 ,CAP_PERCENTAGE_RESOURCE = 40
 ,QUERY_EXECUTION_TIMEOUT_SEC = 3600)
```

Create the Classifier

Previously, the mapping of queries to resource classes was done with sp_addrolemember. To achieve the same functionality and map requests to workload groups, use the CREATE WORKLOAD CLASSIFIER syntax. Using sp_addrolemember only allowed you to map resources to a request based on a login. A classifier provides additional options besides login, such as: - label - session - time The below example assigns queries from the `AdfLogin` login that also have the OPTION LABEL set to `factloads` to the workload group `wgDataLoads` created above.

```
CREATE WORKLOAD CLASSIFIER wcDataLoads WITH
( WORKLOAD_GROUP = 'wgDataLoads'
 ,MEMBERNAME = 'AdfLogin'
 ,WLM_LABEL = 'factloads')
```

Test with a sample query

Below is a sample query and a DMV query to ensure the workload group and classifier are configured correctly.

```
SELECT SUSER_SNAME() --should be 'AdfLogin'

--change to a valid table AdfLogin has access to
SELECT TOP 10 *
  FROM nation
  OPTION (label='factloads')

SELECT request_id, [label], classifier_name, group_name, command
  FROM sys.dm_pdw_exec_requests
  WHERE [label] = 'factloads'
  ORDER BY submit_time DESC
```

Next steps

- Workload Isolation
- How-To Create a Workload Group
- CREATE WORKLOAD CLASSIFIER (Transact-SQL)
- CREATE WORKLOAD GROUP (Transact-SQL)

Troubleshooting a misclassified workload in Azure Synapse Analytics

8/19/2022 • 9 minutes to read • Edit Online

This article covers guidance on how to troubleshoot a misclassified workload, and how to identify the reason behind the classification, for workloads in a dedicated SQL pool.

Azure Synapse Analytics provides workload management capabilities like classifying workloads to appropriate workload groups, assigning importance, and isolating resources to meet SLAs.

However, in some scenarios, a combination of these capabilities can lead to workload classification that doesn't reflect user intent. This article lists such common scenarios and how to troubleshoot them. First, you should query basic information for troubleshooting misclassified workload scenarios.

> **NOTE**
> Classifying managed identities (MI) behavior differs between the dedicated SQL pool in Azure Synapse workspaces and the standalone dedicated SQL pool (formerly SQL DW). While the standalone dedicated SQL pool MI maintains the assigned identity, for Azure Synapse workspaces the MI runs as **dbo**. This cannot be changed. The dbo role, by default, is classified to smallrc. Creating a classifier for the dbo role allows for assigning requests to a workload group other than smallrc. If dbo alone is too generic for classification and has broader impacts, consider using label, session or time-based classification in conjunction with the dbo role classification.

Basic troubleshooting information

To troubleshoot a misclassified workload scenario the following information is needed:

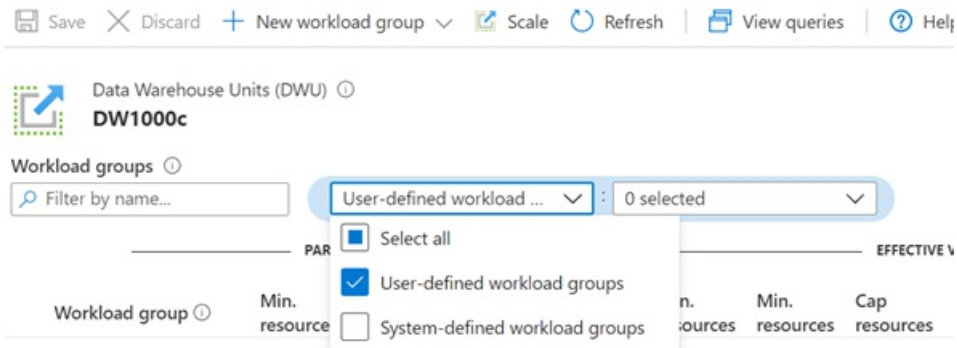
- List of all workload groups
- List of all workload classifiers and associated workload groups
- List of users and mapped workload groups (system and user defined)
- List of workload group and classifier details of a request

Workload groups

Azure portal

To get a list of all workload groups (including system workload groups) and associated details in the Azure portal:

1. Go to the Azure Synapse workspace under which the dedicated SQL Pool of interest has been created.
2. On the left side pane, all pool types created under the workspace are listed. Select **SQL Pools** under **Analytical Pools** section.
3. Select the dedicated SQL pool of interest.
4. In the left side pane, select **Workload management** under **Settings**.
5. Under **Workload groups** section, all workloads are listed. By default only **User-Defined Workload groups** are listed. To view a list of both user-defined and system-defined workload groups, edit the filter and **Select All**.



T-SQL

To view workload groups using T-SQL, connect to the dedicated SQL Pool using SQL Server Management Studio (SSMS) and issue following query:

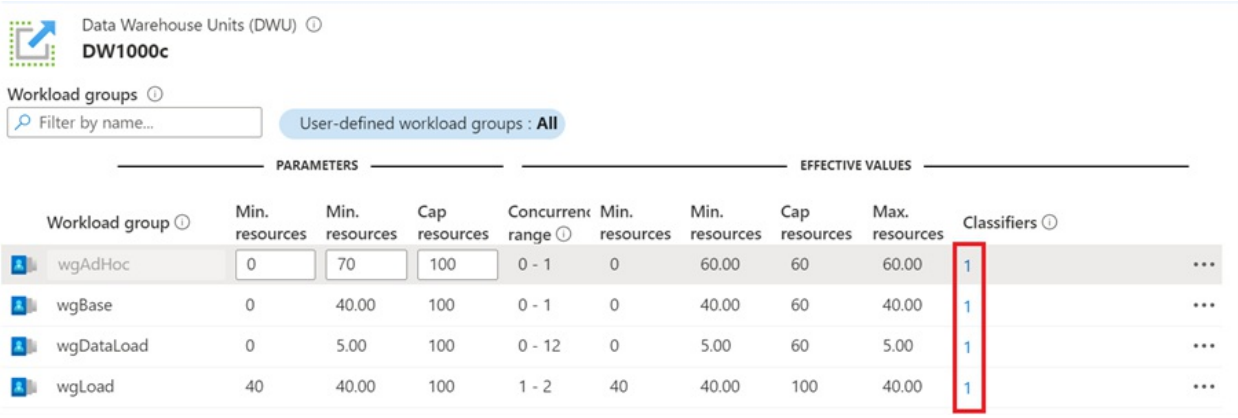
```
SELECT * FROM sys.workload_management_workload_groups;
```

For more information, see sys.workload_management_workload_groups.

Workload classifiers

Azure portal

To list all workload classifiers (including system-defined classifiers) by workload group in the Azure portal, see the numbers listed in **Classifiers** column in Workload group table (see previous section).



T-SQL

To get the list of all workload classifiers (including system-defined classifiers) using T-SQL, use following query:

```
SELECT * FROM sys.workload_management_workload_classifiers;
```

For more information, see sys.workload_management_workload_classifiers.

Users and mapped resource classes

To get a list of system defined resource classes and mapped users in T-SQL, use following query:

```
SELECT  r.name AS [Resource Class]
,       m.name AS [Member Name]
FROM    sys.database_role_members AS rm
JOIN    sys.database_principals AS r ON rm.role_principal_id = r.principal_id
JOIN    sys.database_principals AS m ON rm.member_principal_id = m.principal_id
WHERE   r.name IN
('mediumrc','largerc','xlargerc','staticrc10','staticrc20','staticrc30','staticrc40','staticrc50','staticrc6
0','staticrc70','staticrc80');
```

Workload group and classifier details of a request

The first step in troubleshooting a misclassified workload problem is to identify the workload group and workload classifier for a query. Use the dynamic management view sys.dm_pdw_exec_requests to view submitted queries and their classification:

```
SELECT * FROM sys.dm_pdw_exec_requests;
```

For more information, see sys.dm_pdw_exec_requests.

Common scenarios of misclassified workloads

Here are some common scenarios where unintended misclassified workloads can occur:

Mixed usage of resource classes and user-defined workload management

In scenarios where resource classes and workload groups are used together, user role to resource class mappings can conflict with workload classifier rules, and lead to unintended query classification.

Consider following scenario:

1. A database user, DBAUser, is assigned to largerc resource class role using `sp_addrolemember` procedure.
2. DBAUser has created a new workload group and classifier using workload management.
3. A newly-created workload classifier maps database role DBARole to mediumrc resource class with high importance.
4. DBAUser is a made a member of the DBARole database role.
5. When DBAUser runs a query, the query is expected to run on mediumrc based on workload classifier. Instead it will be assigned to largerc, as **user** mapping takes precedence over **role membership** mapping to a classifier.

It's best to avoid mixing usage of resource classes and workload management groups to do workload management. For more information on steps to convert resource classes to workloads, see Convert Resource Classes to Workload Groups.

However, there can be situations where both resource classes and workload management need to be used together. In such scenarios, to simplify troubleshooting misclassification, we recommend you remove resource class role mappings as you create workload classifiers. The code below returns existing resource class role memberships. Run sp_droprolemember for each member name returned from the corresponding resource class.

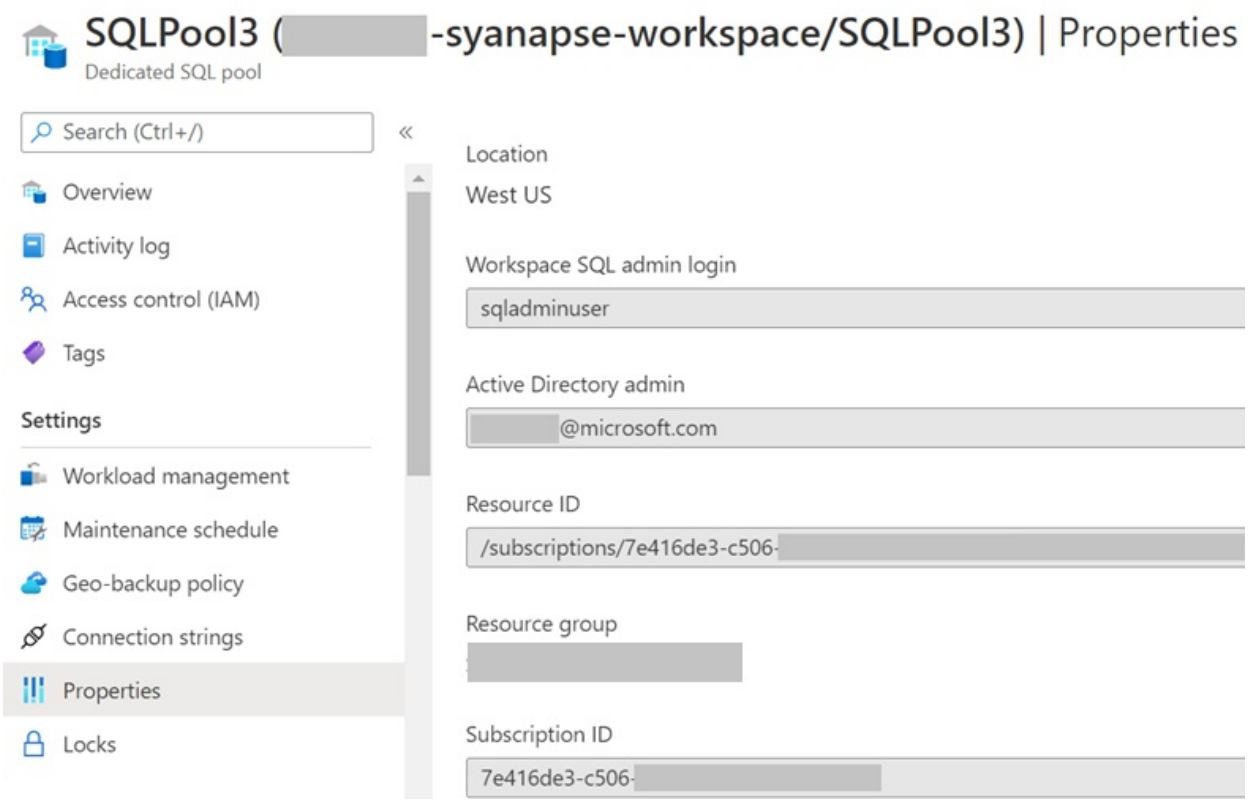
```
SELECT  r.name AS [Resource Class]
, m.name AS membername
FROM    sys.database_role_members AS rm
JOIN    sys.database_principals AS r ON rm.role_principal_id = r.principal_id
JOIN    sys.database_principals AS m ON rm.member_principal_id = m.principal_id
WHERE   r.name IN
('mediumrc','largerc','xlargerc','staticrc10','staticrc20','staticrc30','staticrc40','staticrc50','staticrc6
0','staticrc70','staticrc80');

--for each row returned run
EXEC sp_droprolemember '[Resource Class]', membername;
```

Some administrative users are always mapped to smallrc workload group

Consider a scenario for the Azure Synapse Workspace SQL Admin login, the Azure Synapse Azure Active Directory admin (user or group member), or a database owner. These users may still have a workload classifier or have been added to a resource class role other than smallrc. All queries executed by these user will still run on smallrc resource class, even though the user is mapped to a different resource class or workload group.

Recommendation: These administrative users can't change their default workload group. For more information, see workload management with resource classes. It is recommended that critical or performance-sensitive workloads not run as one of these administrative users in the dedicated SQL pool.

The Azure Synapse Workspace SQL Admin login and the Azure Synapse Azure Active Directory admin (user or group member) are specified in the Azure portal:



Similarly, the database owner (dbo) and **db_owner** database roles are not allowed to change their default resource class. If a user is either the database owner or added under **db_owner** database role, all queries executed by the user go to smallrc by default. These roles can't be added to a resource class other than smallrc. However, if a user who is part of this role would like to classify their queries to a different workload group, they can use MEMBERNAME option in workload classifier definition.

Use workload group precedence for better classification

In scenarios where workload classifiers are mapped to multiple workload groups, or a user is mapped to multiple resource classes, precedence order determines chosen workload group and Resource class. List of precedence rules:

> **NOTE**
> As mentioned in Mixing Resource Classes and user-defined Workload Management section, it isn't recommended to combine usage of Resource classes and user-defined workload groups/classes.

If resource classes are being used:

In scenarios where resource classes are being used it's best to create a dedicated user for each type of workload being run. However, if a user is a member of multiple resource classes, then precedence rules documented in Resource Class Precedence take effect:

1. Dynamic resource class takes precedence over static resource class. For example, if a user is member of mediumrc (dynamic) and staticrc80(static), user queries run with mediumrc.
2. Bigger resource classes are preferred over smaller resource classes. For example, if a user is member of staticrc20 and staticrc80, then user queries will run with staticrc80.

If Workload Management Capabilities are used

WLM provides capability to create multiple workload classifiers for same user or workload group. Classifier definition statement has multiple parameters based on which incoming requests are assigned to workloads. These parameters have a **weight** score as shown below and this score determines order of precedence:

CLASSIFIER PARAMETER	WEIGHT
MEMBERNAME:USER	64
MEMBERNAME:ROLE	32
WLM_LABEL	16
WLM_CONTEXT	8
START_TIME/END_TIME	4

Let us see these precedence rules in action using an example. If a user creates two workload classifiers as follows:

```
CREATE WORKLOAD CLASSIFIER CLASSIFIER-1 WITH
( WORKLOAD_GROUP = 'wgDataLoad'
 ,MEMBERNAME     = 'USER-1'
 ,WLM_LABEL      = 'dimension_loads'
 ,IMPORTANCE     = 'High');

CREATE WORKLOAD CLASSIFIER CLASSIFIER-2 WITH
( WORKLOAD_GROUP = 'wgUserqueries'
 ,MEMBERNAME     = 'USER-1'
 ,START_TIME     = '18:00'
 ,END_TIME       = '07:30'
 ,IMPORTANCE     = 'Low')
```

Queries submitted by User-1 can be submitted via both classifiers. Query run by User-1 with the 'dimension_loads' label between 6PM and 7AM UTC will be assigned to `wgDashboards` as weight score of WLM_LABEL is higher than START_TIME/END_TIME. The weighting of `CLASSIFIER-1` is 80 (64 for user, plus 16 for WLM_LABEL). The weighting of `CLASSIFIER-2` is 68 (64 for user, 4 for START_TIME/END_TIME).

For more information on workload classification, see classification weighting and query labels.

What happens if precedence in workload classification leads to a tie?

Even after workload classifier precedence is applied, it is possible for a query to be classified to multiple workload groups. For example, consider following classifiers:

```
CREATE WORKLOAD CLASSIFIER CLASSIFIER-1 WITH
( WORKLOAD_GROUP = 'wgDataLoad'
 ,MEMBERNAME     = 'USER-1'
 ,WLM_LABEL      = 'dimension_loads'
 ,IMPORTANCE     = 'High');

CREATE WORKLOAD CLASSIFIER CLASSIFIER-2 WITH
( WORKLOAD_GROUP = 'wgUserqueries'
 ,MEMBERNAME     = 'USER-1'
 ,WLM_LABEL      = 'dimension_loads'
 ,IMPORTANCE     = 'Low');
```

If a user runs a query with `OPTION (LABEL = 'dimension_loads')`, it can be classified to either `wgDataLoad` or

`wgUserqueries` based on precedence rules. But which workload group is chosen?

Workload group and classifier tie-breakers

If there was a tie in workload groups or classifiers, following precedence takes effect:

1. The workload group with the highest resource allocation is chosen. This behavior optimizes for performance in scenarios where logins are members of multiple resource classes. This behavior also ensures backward compatibility.

Consider the following two workload groups and workload classifiers:

```
-- Workload Groups
CREATE WORKLOAD GROUP wgDataLoad
WITH
( MIN_PERCENTAGE_RESOURCE = 26
,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 2
,CAP_PERCENTAGE_RESOURCE = 100 )

CREATE WORKLOAD GROUP wgUserqueries
WITH
( MIN_PERCENTAGE_RESOURCE = 15
,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 5
,CAP_PERCENTAGE_RESOURCE = 50 );

--Workload Classifiers
CREATE WORKLOAD CLASSIFIER CLASSIFIER-1 WITH
( WORKLOAD_GROUP = 'wgDataLoad'
,MEMBERNAME     = 'USER-1'
,WLM_LABEL      = 'dimension_loads'
,IMPORTANCE     = 'High');

CREATE WORKLOAD CLASSIFIER CLASSIFIER-2 WITH
( WORKLOAD_GROUP = 'wgUserqueries'
,MEMBERNAME     = 'USER-1'
,WLM_LABEL      = 'dimension_loads'
,IMPORTANCE     = 'High');
```

If a user runs a query with `OPTION (LABEL = 'dimension_loads')`, the query meets criteria for both the classifiers. However, the user request will be routed to `wgUserQueries` workload group as it has higher minimum resource allocation per request.

2. Next, concurrency setting and available concurrency for respective workload groups is considered for tie-breakers. The workload group with the highest available concurrency at a time when request arrived will give the query best chance of executing.

Consider following two workload groups and workload classifiers:

```
--Workload Groups
CREATE WORKLOAD GROUP wgDataLoad
WITH
( MIN_PERCENTAGE_RESOURCE = 30
 ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 2 (concurrency of 15)
 ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 2
 ,CAP_PERCENTAGE_RESOURCE = 100 );

CREATE WORKLOAD GROUP wgUserqueries
WITH
( MIN_PERCENTAGE_RESOURCE = 30
 ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 2 (concurrency of 15)
 ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 2
 ,CAP_PERCENTAGE_RESOURCE = 100 );

-- Workload Classifiers
CREATE WORKLOAD CLASSIFIER CLASSIFIER-1 WITH
( WORKLOAD_GROUP = 'wgDataLoad'
 ,MEMBERNAME     = 'USER-1'
 ,WLM_LABEL      = 'dimension_loads'
 ,IMPORTANCE     = 'High');

CREATE WORKLOAD CLASSIFIER CLASSIFIER-2 WITH
( WORKLOAD_GROUP = 'wgUserqueries'
 ,MEMBERNAME     = 'USER-1'
 ,WLM_LABEL      = 'dimension_loads'
 ,IMPORTANCE     = 'High');
```

If a user runs a query with `OPTION (LABEL = 'dimension_loads')`, both the classifiers have a tie as the query meets criteria for both. When user submits the query, consider the scenario where five concurrent queries are getting executed in the `wgUserqueries` group and ten queries are getting executed in the `wgDataLoad` group. The user request will be routed to `wgUserqueries` group using `CLASSIFIER-2`, as the `wgUserqueries` workload group has higher available concurrency at the time the user submitted a query.

3. Next, the importance setting of the request is considered for tie-breakers. If there was a tie in workload classification using precedence rules, the request will be routed to workload group that has highest importance. For more information, see workload importance.

4. Finally, the creation time of the workload group is considered for tie-breakers. The user request will be routed to the workload group that was created most recently.

Next steps

- For more information on workload classification, see Workload Classification.
- For more information on workload importance, see Workload Importance

Go to Configure Workload Importance

Load data from Azure Data Lake Storage into dedicated SQL pools in Azure Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

This guide outlines how to use the COPY statement to load data from Azure Data Lake Storage. For quick examples on using the COPY statement across all authentication methods, visit the following documentation: Securely load data using dedicated SQL pools.

> **NOTE**
>
> To provide feedback or report issues on the COPY statement, send an email to the following distribution list: sqldwcopypreview@service.microsoft.com.

- Create the target table to load data from Azure Data Lake Storage.
- Create the COPY statement to load data into the data warehouse.

If you don't have an Azure subscription, create a free account before you begin.

Before you begin

Before you begin this tutorial, download and install the newest version of SQL Server Management Studio (SSMS).

To run this tutorial, you need:

- A dedicated SQL pool. See Create a dedicated SQL pool and query data.
- A Data Lake Storage account. See Get started with Azure Data Lake Storage. For this storage account, you will need to configure or specify one of the following credentials to load: A storage account key, shared access signature (SAS) key, an Azure Directory Application user, or an AAD user which has the appropriate Azure role to the storage account.

Create the target table

Connect to your dedicated SQL pool and create the target table you will to load to. In this example, we are creating a product dimension table.

```
-- A: Create the target table
-- DimProduct
CREATE TABLE [dbo].[DimProduct]
(
    [ProductKey] [int] NOT NULL,
    [ProductLabel] [nvarchar](255) NULL,
    [ProductName] [nvarchar](500) NULL
)
WITH
(
    DISTRIBUTION = HASH([ProductKey]),
    CLUSTERED COLUMNSTORE INDEX
    --HEAP
);
```

Create the COPY statement

Connect to your SQL dedicated pool and run the COPY statement. For a complete list of examples, visit the following documentation: Securely load data using dedicated SQL pools.

```
-- B: Create and execute the COPY statement

COPY INTO [dbo].[DimProduct]
--The column list allows you map, omit, or reorder input file columns to target table columns.
--You can also specify the default value when there is a NULL value in the file.
--When the column list is not specified, columns will be mapped based on source and target ordinality
(
 ProductKey default -1 1,
 ProductLabel default 'myStringDefaultWhenNull' 2,
 ProductName default 'myStringDefaultWhenNull' 3
)
--The storage account location where you data is staged
FROM 'https://storageaccount.blob.core.windows.net/container/directory/'
WITH
(
   --CREDENTIAL: Specifies the authentication method and credential access your storage account
   CREDENTIAL = (IDENTITY = '', SECRET = ''),
   --FILE_TYPE: Specifies the file type in your storage account location
   FILE_TYPE = 'CSV',
   --FIELD_TERMINATOR: Marks the end of each field (column) in a delimited text (CSV) file
   FIELDTERMINATOR = '|',
   --ROWTERMINATOR: Marks the end of a record in the file
   ROWTERMINATOR = '0x0A',
   --FIELDQUOTE: Specifies the delimiter for data of type string in a delimited text (CSV) file
   FIELDQUOTE = '',
   ENCODING = 'UTF8',
   DATEFORMAT = 'ymd',
   --MAXERRORS: Maximum number of reject rows allowed in the load before the COPY operation is canceled
   MAXERRORS = 10,
   --ERRORFILE: Specifies the directory where the rejected rows and the corresponding error reason should be
written
   ERRORFILE = '/errorsfolder',
) OPTION (LABEL = 'COPY: ADLS tutorial');
```

Optimize columnstore compression

By default, tables are defined as a clustered columnstore index. After a load completes, some of the data rows might not be compressed into the columnstore. There's a variety of reasons why this can happen. To learn more, see manage columnstore indexes.

To optimize query performance and columnstore compression after a load, rebuild the table to force the columnstore index to compress all the rows.

```
ALTER INDEX ALL ON [dbo].[DimProduct] REBUILD;
```

Optimize statistics

It is best to create single-column statistics immediately after a load. There are some choices for statistics. For example, if you create single-column statistics on every column it might take a long time to rebuild all the statistics. If you know certain columns are not going to be in query predicates, you can skip creating statistics on those columns.

If you decide to create single-column statistics on every column of every table, you can use the stored

procedure code sample `prc_sqldw_create_stats` in the statistics article.

The following example is a good starting point for creating statistics. It creates single-column statistics on each column in the dimension table, and on each joining column in the fact tables. You can always add single or multi-column statistics to other fact table columns later on.

Achievement unlocked!

You have successfully loaded data into your data warehouse. Great job!

Next steps

Loading data is the first step to developing a data warehouse solution using Azure Synapse Analytics. Check out our development resources.

Learn how to develop tables for data warehousing

For more loading examples and references, view the following documentation:

- COPY statement reference documentation
- COPY examples for each authentication method
- COPY quickstart for a single table

Load data into Azure Synapse Analytics using Azure Data Factory or a Synapse pipeline

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Azure Synapse Analytics is a cloud-based, scale-out database that's capable of processing massive volumes of data, both relational and non-relational. Azure Synapse Analytics is built on the massively parallel processing (MPP) architecture that's optimized for enterprise data warehouse workloads. It offers cloud elasticity with the flexibility to scale storage and compute independently.

Getting started with Azure Synapse Analytics is now easier than ever. Azure Data Factory and its equivalent pipelines feature within Azure Synapse itself provide a fully managed cloud-based data integration service. You can use the service to populate an Azure Synapse Analytics with data from your existing system and save time when building your analytics solutions.

Azure Data Factory and Synapse pipelines offer the following benefits for loading data into Azure Synapse Analytics:

- **Easy to set up**: An intuitive 5-step wizard with no scripting required.
- **Rich data store support**: Built-in support for a rich set of on-premises and cloud-based data stores. For a detailed list, see the table of Supported data stores.
- **Secure and compliant**: Data is transferred over HTTPS or ExpressRoute. The global service presence ensures that your data never leaves the geographical boundary.
- **Unparalleled performance by using PolyBase**: Polybase is the most efficient way to move data into Azure Synapse Analytics. Use the staging blob feature to achieve high load speeds from all types of data stores, including Azure Blob storage and Data Lake Store. (Polybase supports Azure Blob storage and Azure Data Lake Store by default.) For details, see Copy activity performance.

This article shows you how to use the Copy Data tool to *load data from Azure SQL Database into Azure Synapse Analytics*. You can follow similar steps to copy data from other types of data stores.

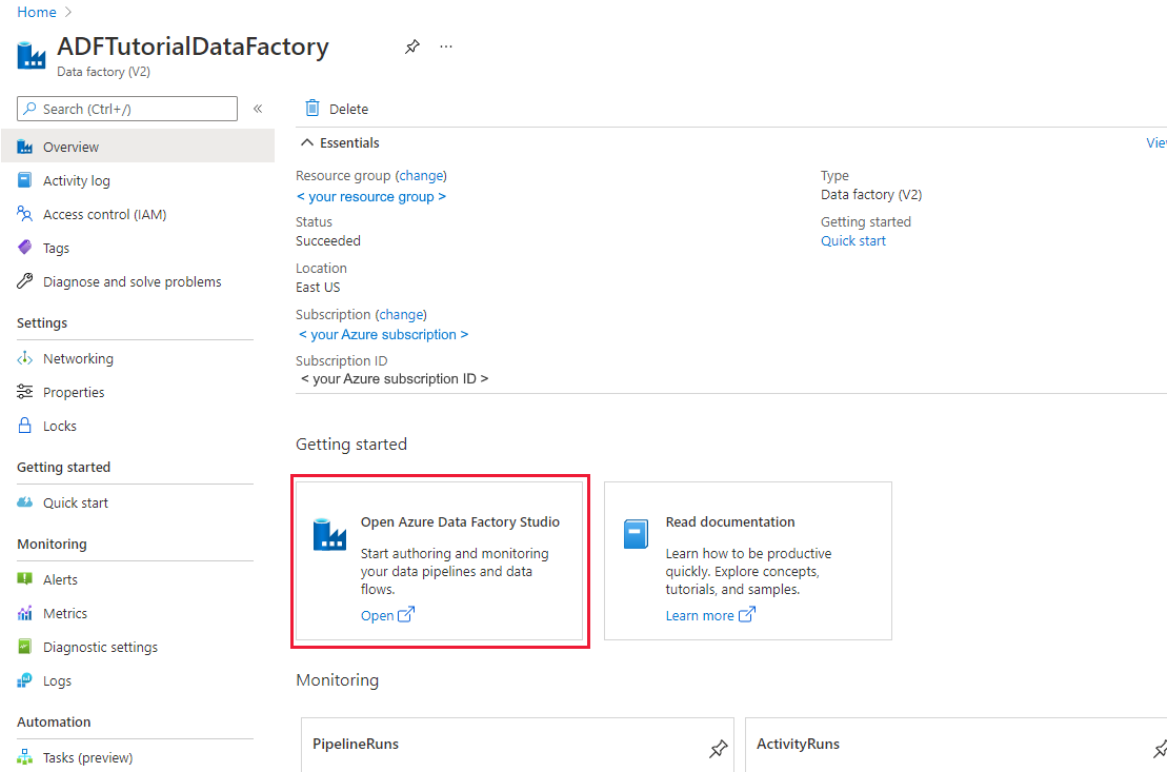
> **NOTE**
> For more information, see Copy data to or from Azure Synapse Analytics.

Prerequisites

- Azure subscription: If you don't have an Azure subscription, create a free account before you begin.
- Azure Synapse Analytics: The data warehouse holds the data that's copied over from the SQL database. If you don't have an Azure Synapse Analytics, see the instructions in Create an Azure Synapse Analytics.
- Azure SQL Database: This tutorial copies data from the Adventure Works LT sample dataset in Azure SQL Database. You can create this sample database in SQL Database by following the instructions in Create a sample database in Azure SQL Database.
- Azure storage account: Azure Storage is used as the *staging* blob in the bulk copy operation. If you don't have an Azure storage account, see the instructions in Create a storage account.

Create a data factory

1. If you have not created your data factory yet, follow the steps in Quickstart: Create a data factory by using the Azure portal and Azure Data Factory Studio to create one. After creating it, browse to the data factory in the Azure portal.

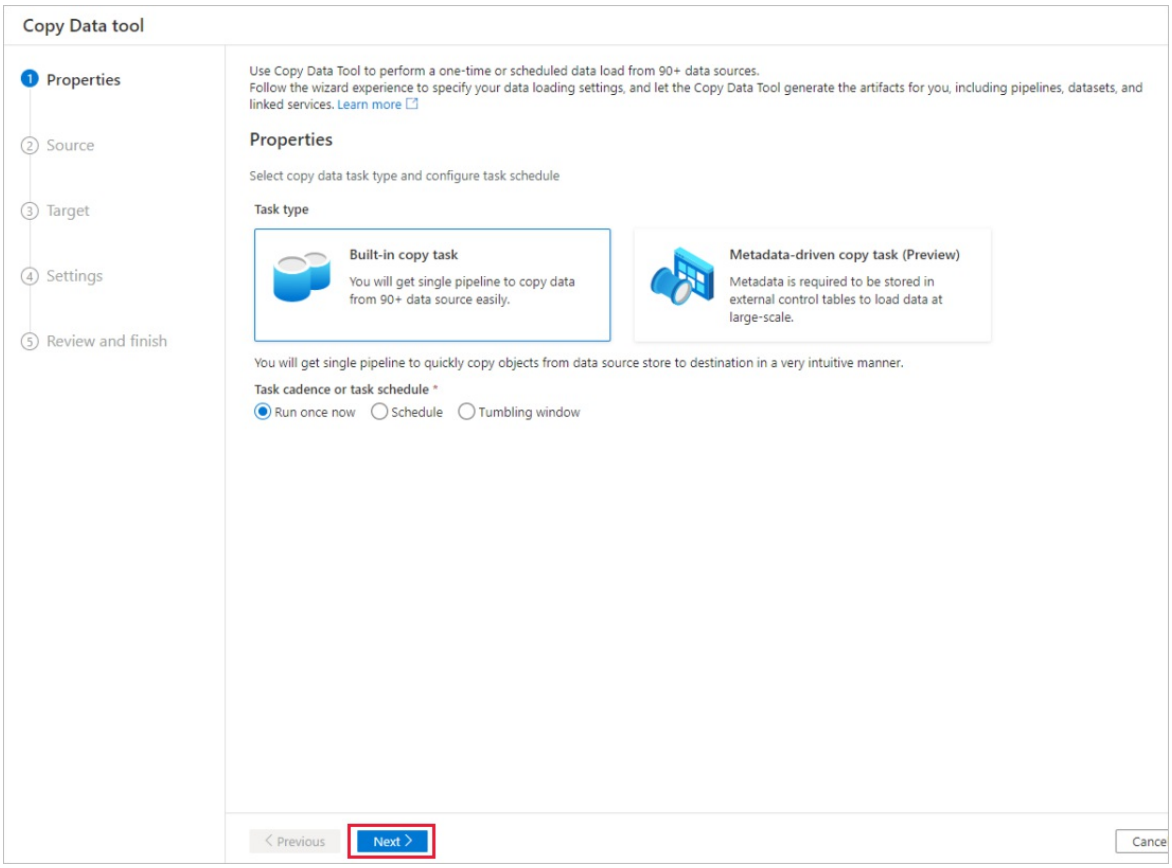


2. Select **Open** on the **Open Azure Data Factory Studio** tile to launch the Data Integration application in a separate tab.

Load data into Azure Synapse Analytics

1. In the home page of Azure Data Factory or Azure Synapse workspace, select the **Ingest** tile to launch the Copy Data tool. Then choose the **Built-in copy task**.

2. In the **Properties** page, choose **Built-in copy task** under **Task type**, then select **Next**.

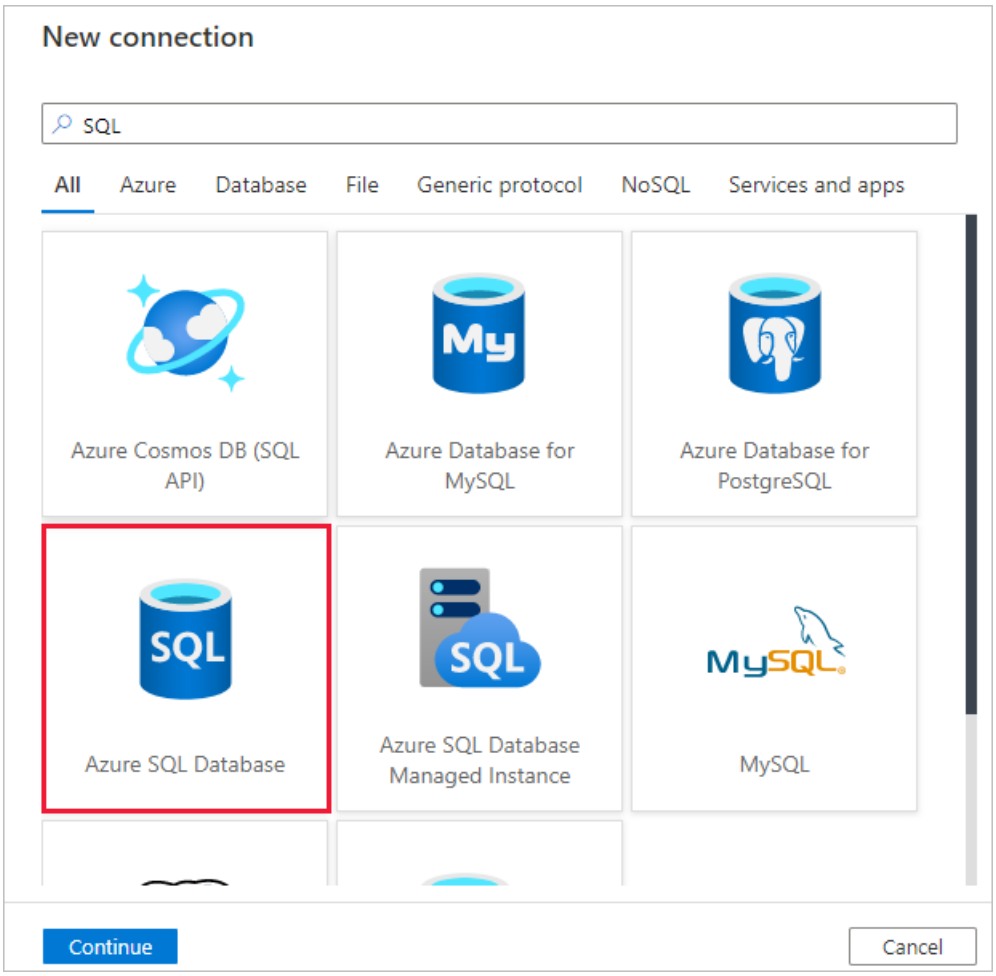


3. In the **Source data store** page, complete the following steps:

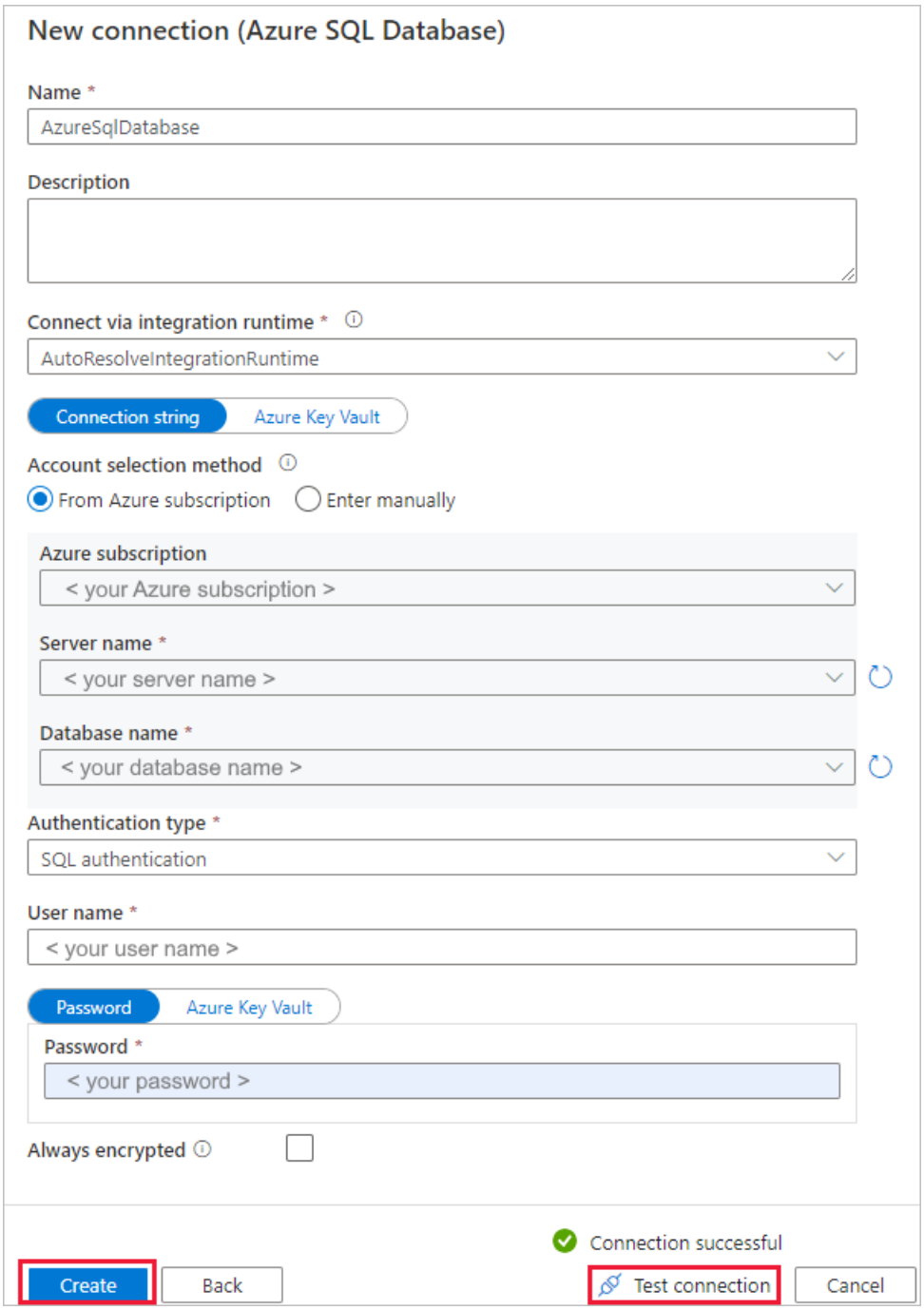
> **TIP**
>
> In this tutorial, you use *SQL authentication* as the authentication type for your source data store, but you can choose other supported authentication methods:*Service Principal* and *Managed Identity* if needed. Refer to corresponding sections in this article for details. To store secrets for data stores securely, it's also recommended to use an Azure Key Vault. Refer to this article for detailed illustrations.

a. Select **+ New connection**.

b. Select **Azure SQL Database** from the gallery, and select **Continue**. You can type "SQL" in the search box to filter the connectors.

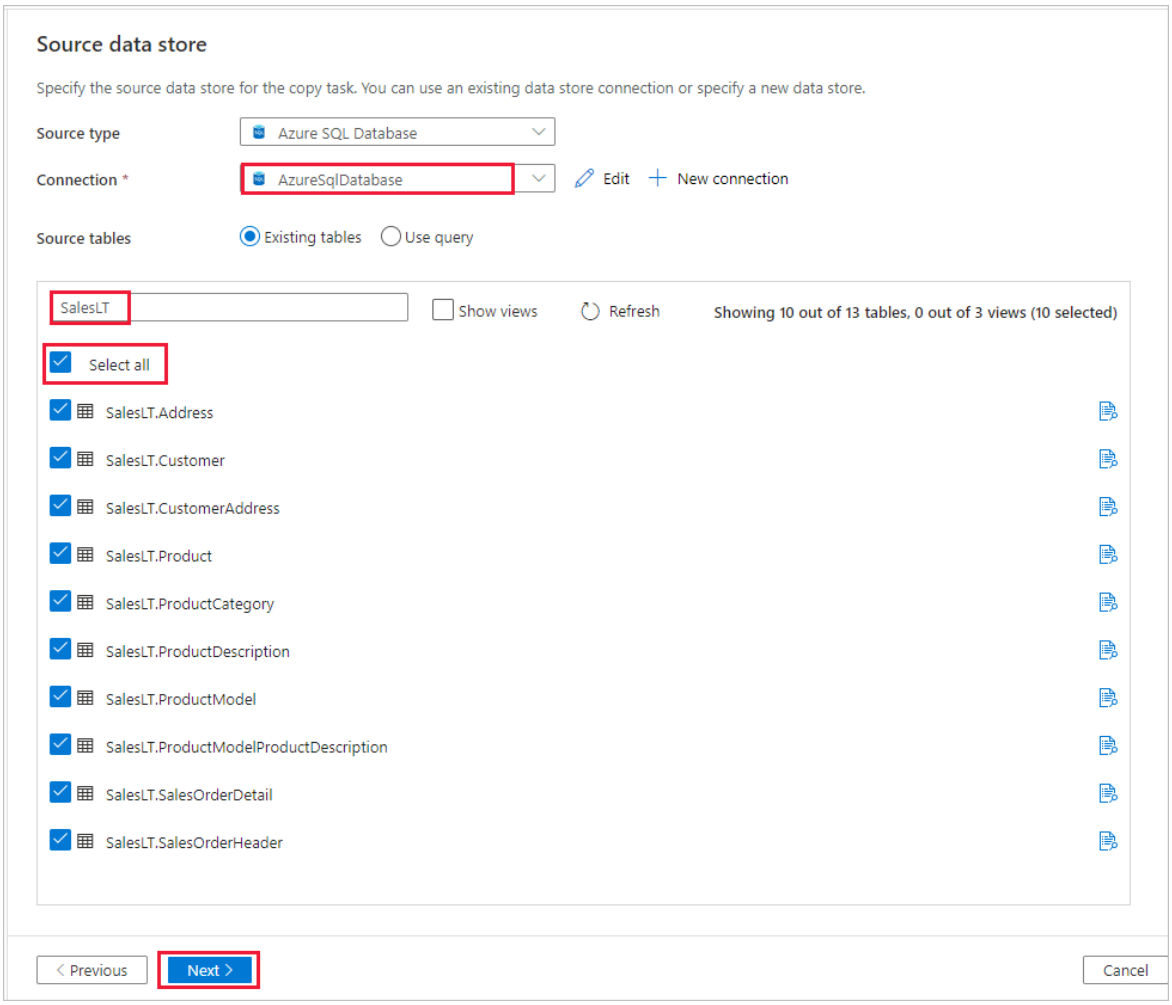


c. In the **New connection (Azure SQL Database)** page, select your server name and DB name from the dropdown list, and specify the username and password. Select **Test connection** to validate the settings, then select **Create**.

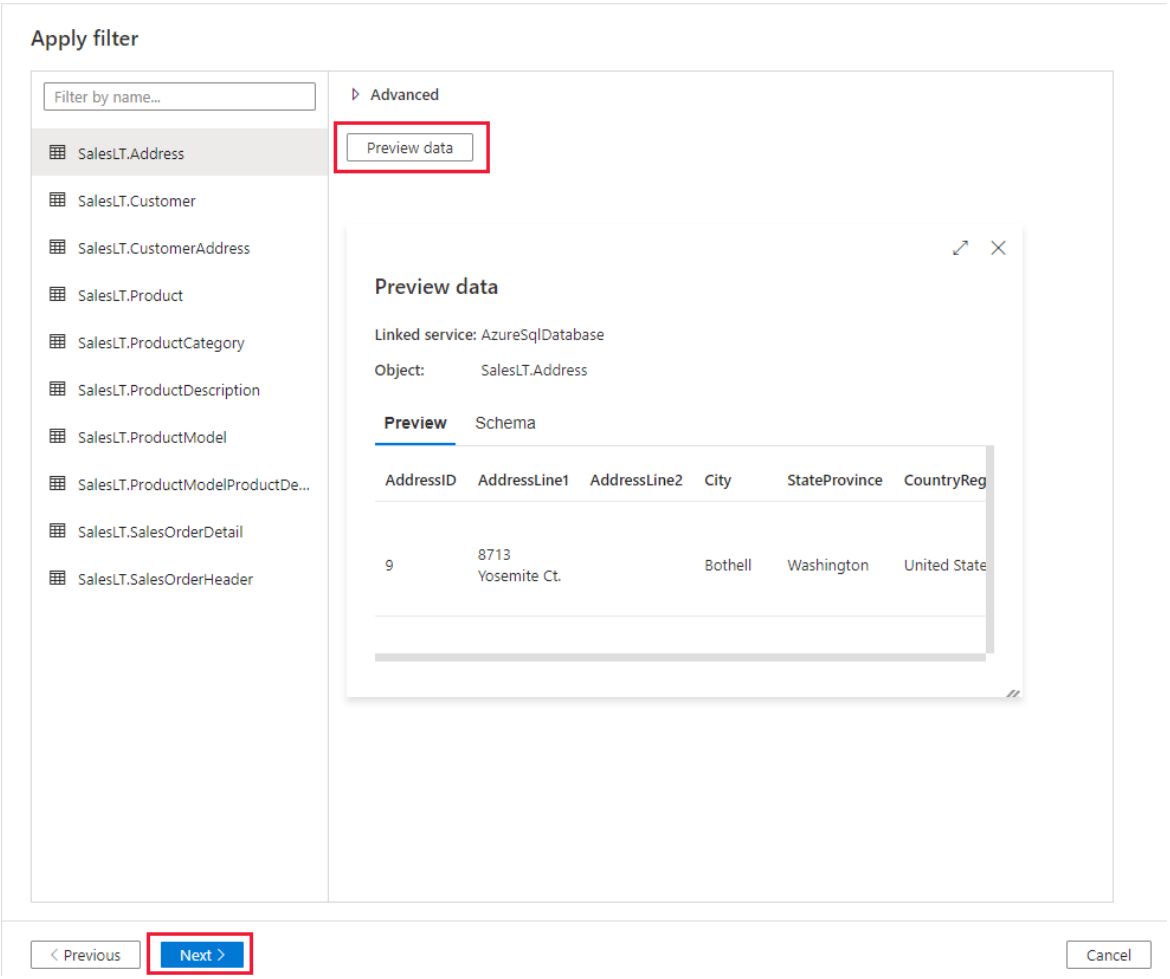


d. In the **Source data store** page, select the newly created connection as source in the **Connection** section.

e. In the **Source tables** section, enter **SalesLT** to filter the tables. Choose the **(Select all)** box to use all of the tables for the copy, and then select **Next**.



4. In the **Apply filter** page, specify your settings or select **Next**. You can preview data and view the schema of the input data by selecting **Preview data** button on this page.

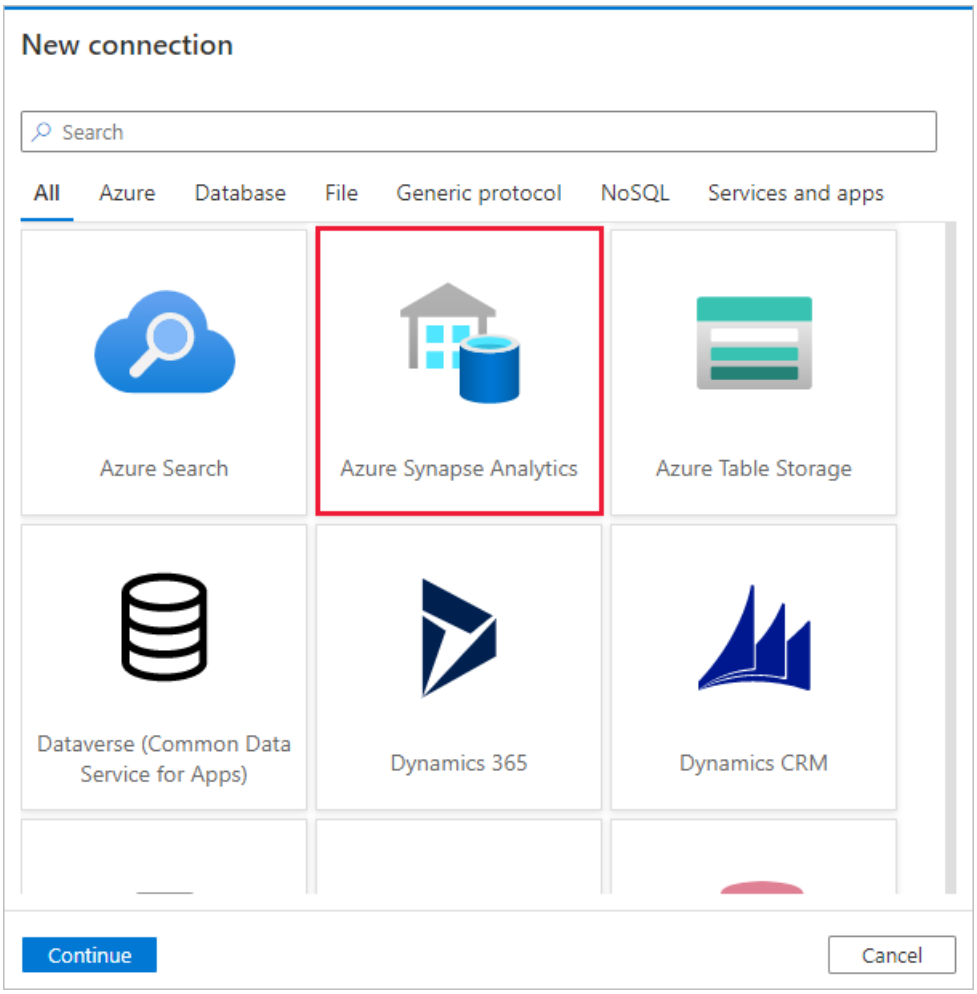


5. In the **Destination data store** page, complete the following steps:

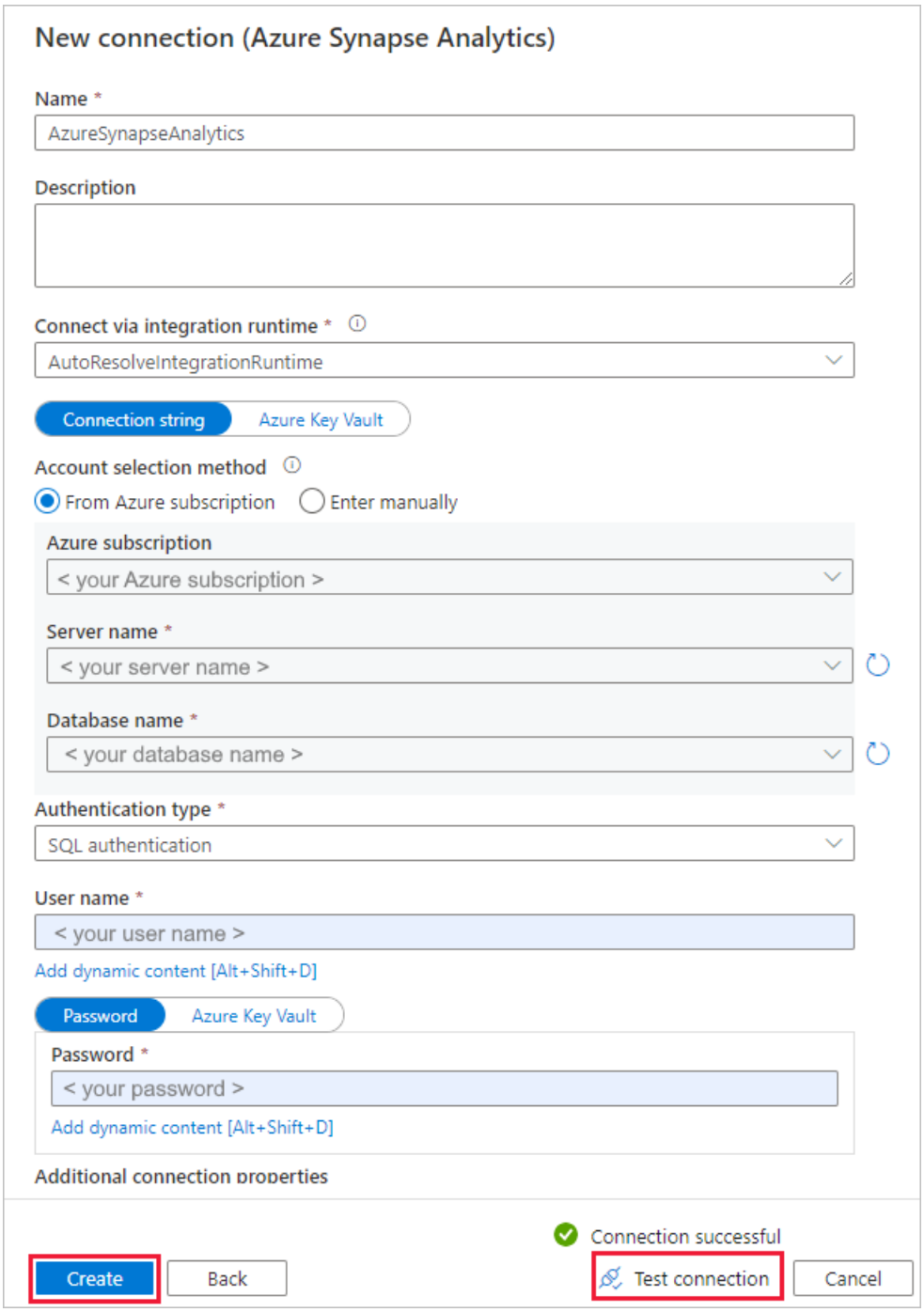
> **TIP**
>
> In this tutorial, you use *SQL authentication* as the authentication type for your destination data store, but you can choose other supported authentication methods:*Service Principal* and *Managed Identity* if needed. Refer to corresponding sections in this article for details. To store secrets for data stores securely, it's also recommended to use an Azure Key Vault. Refer to this article for detailed illustrations.

a. Select **+ New connection** to add a connection.

b. Select **Azure Synapse Analytics** from the gallery, and select **Continue**.

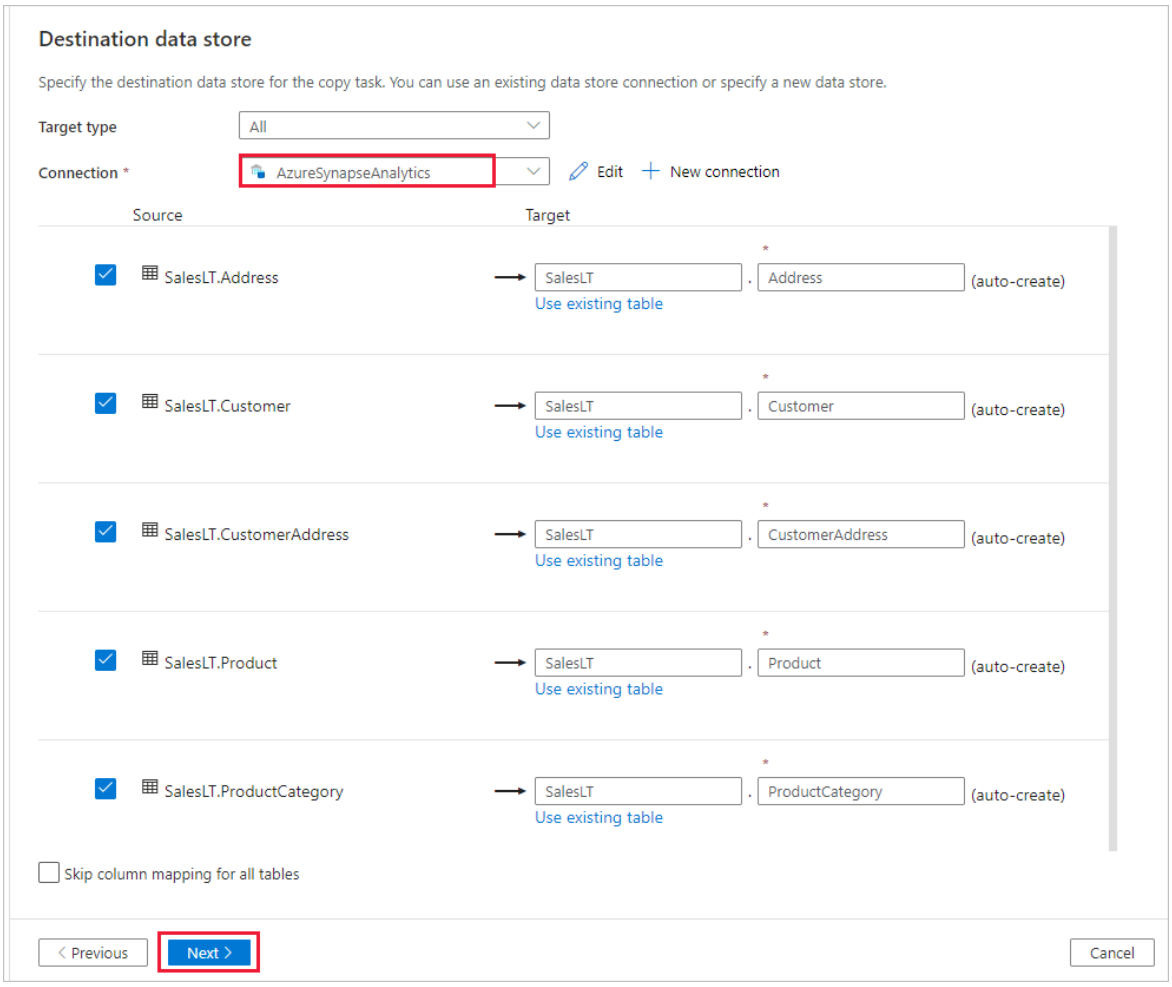


c. In the **New connection (Azure Synapse Analytics)** page, select your server name and DB name from the dropdown list, and specify the username and password. Select **Test connection** to validate the settings, then select **Create**.

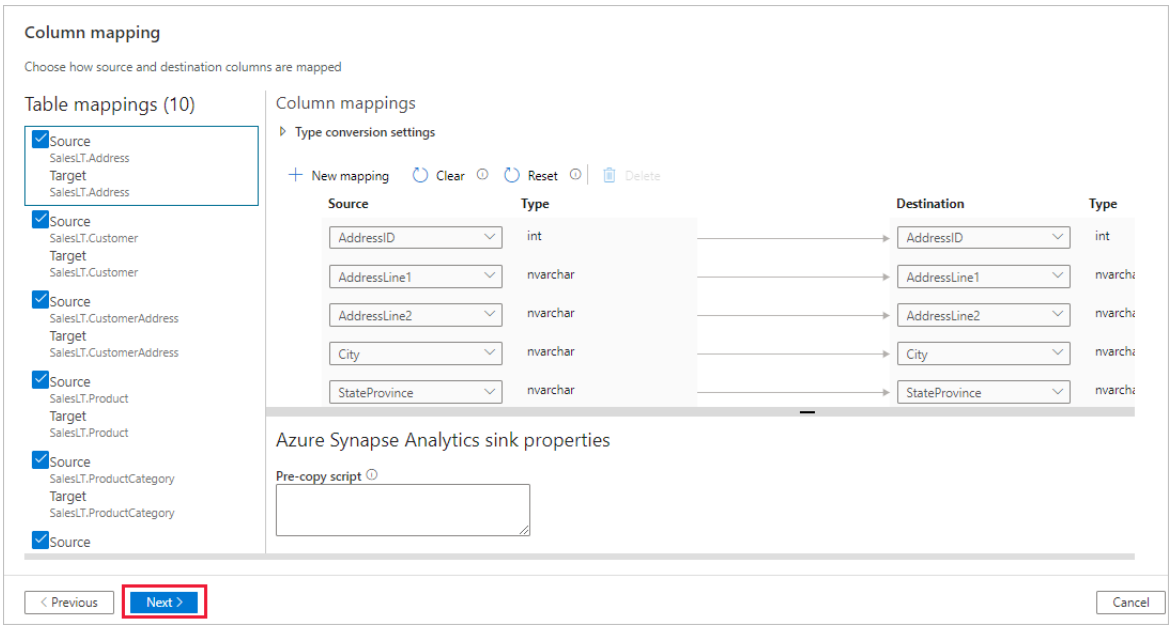


d. In the **Destination data store** page, select the newly created connection as sink in the **Connection** section.

6. In the **Table mapping** section, review the content and select **Next**. An intelligent table mapping displays. The source tables are mapped to the destination tables based on the table names. If a source table doesn't exist in the destination, the service creates a destination table with the same name by default. You can also map a source table to an existing destination table.



7. In the **Column mapping** page, review the content, and select **Next**. The intelligent table mapping is based on the column name. If you let the service automatically create the tables, data type conversion can occur when there are incompatibilities between the source and destination stores. If there's an unsupported data type conversion between the source and destination column, you see an error message next to the corresponding table.



8. In the **Settings** page, complete the following steps:

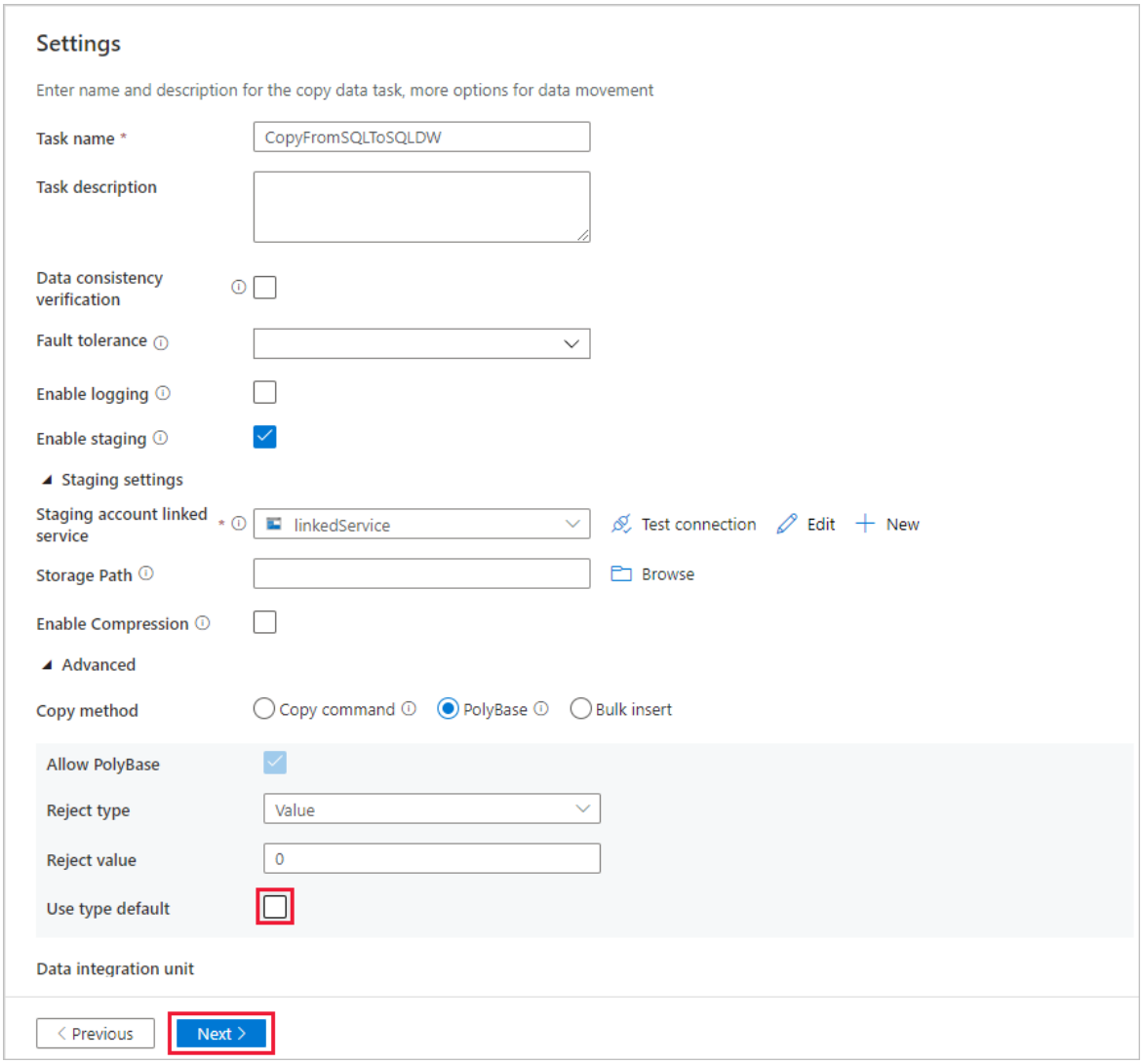
   a. Specify **CopyFromSQLToSQLDW** for the **Task name** field.

   b. In **Staging settings** section, select **+ New** to new a staging storage. The storage is used for staging the data before it loads into Azure Synapse Analytics by using PolyBase. After the copy is

complete, the interim data in Azure Blob Storage is automatically cleaned up.

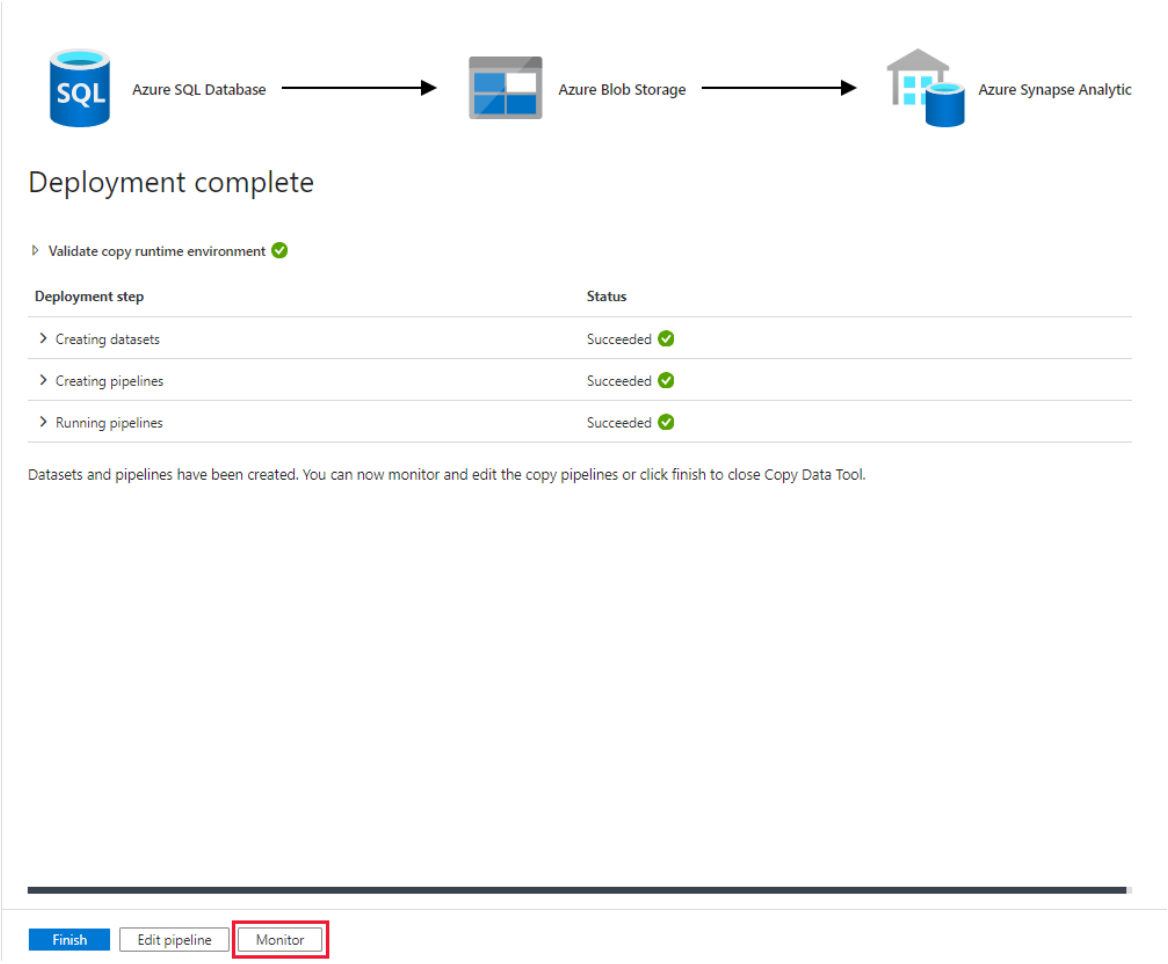
c. In the **New linked service** page, select your storage account, and select **Create** to deploy the linked service.

d. Deselect the **Use type default** option, and then select **Next**.



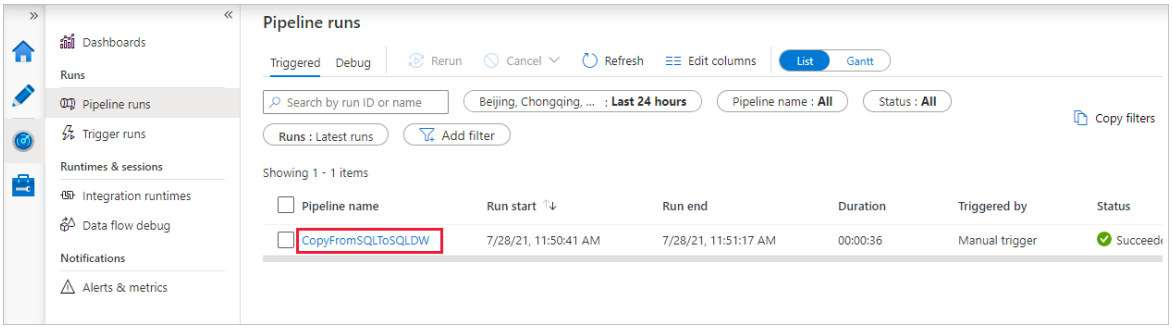
9. In the **Summary** page, review the settings, and select **Next**.

10. On the **Deployment page**, select **Monitor** to monitor the pipeline (task).

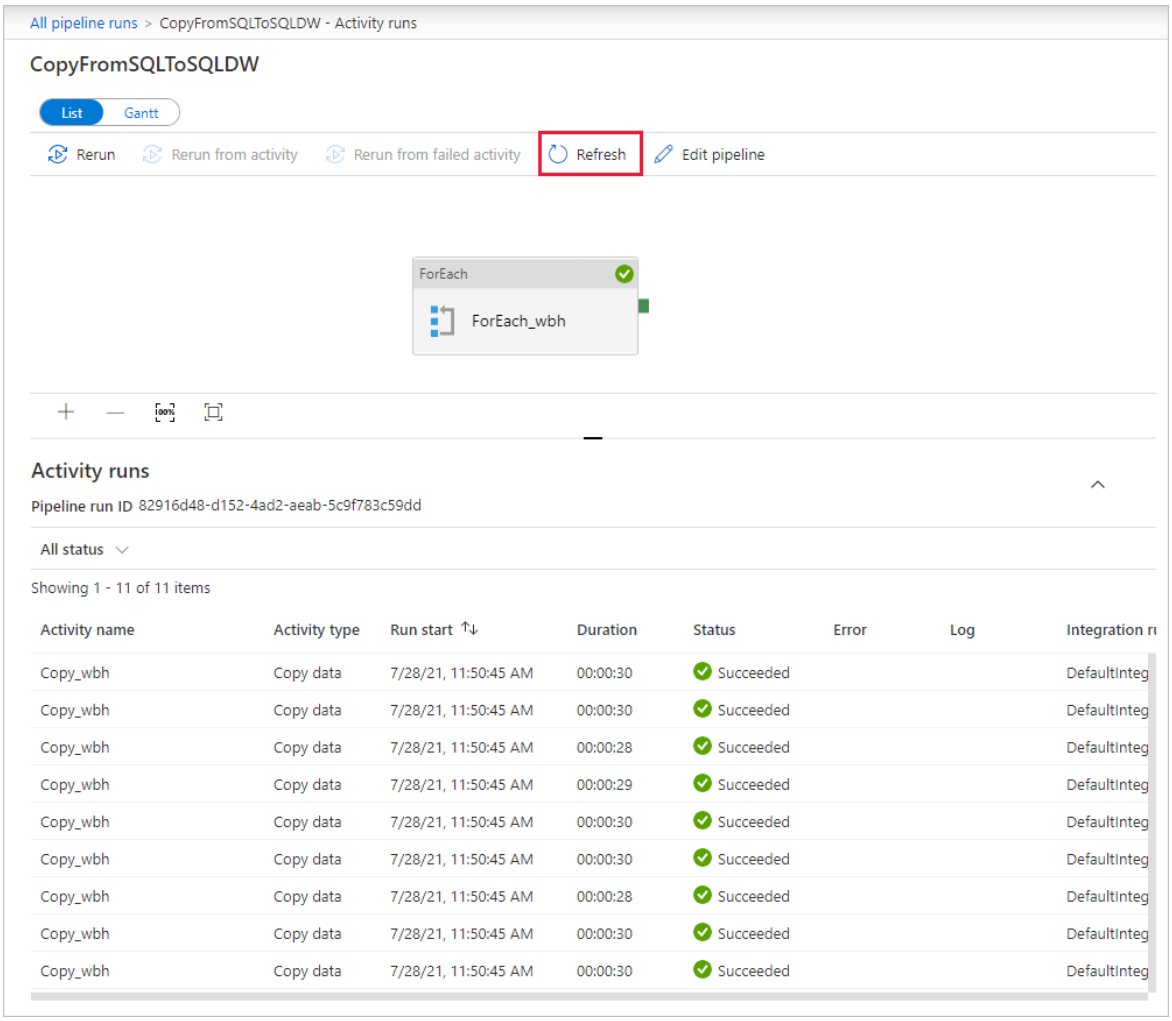


11. Notice that the **Monitor** tab on the left is automatically selected. When the pipeline run completes successfully, select the **CopyFromSQLToSQLDW** link under the **Pipeline name** column to view activity run details or to rerun the pipeline.

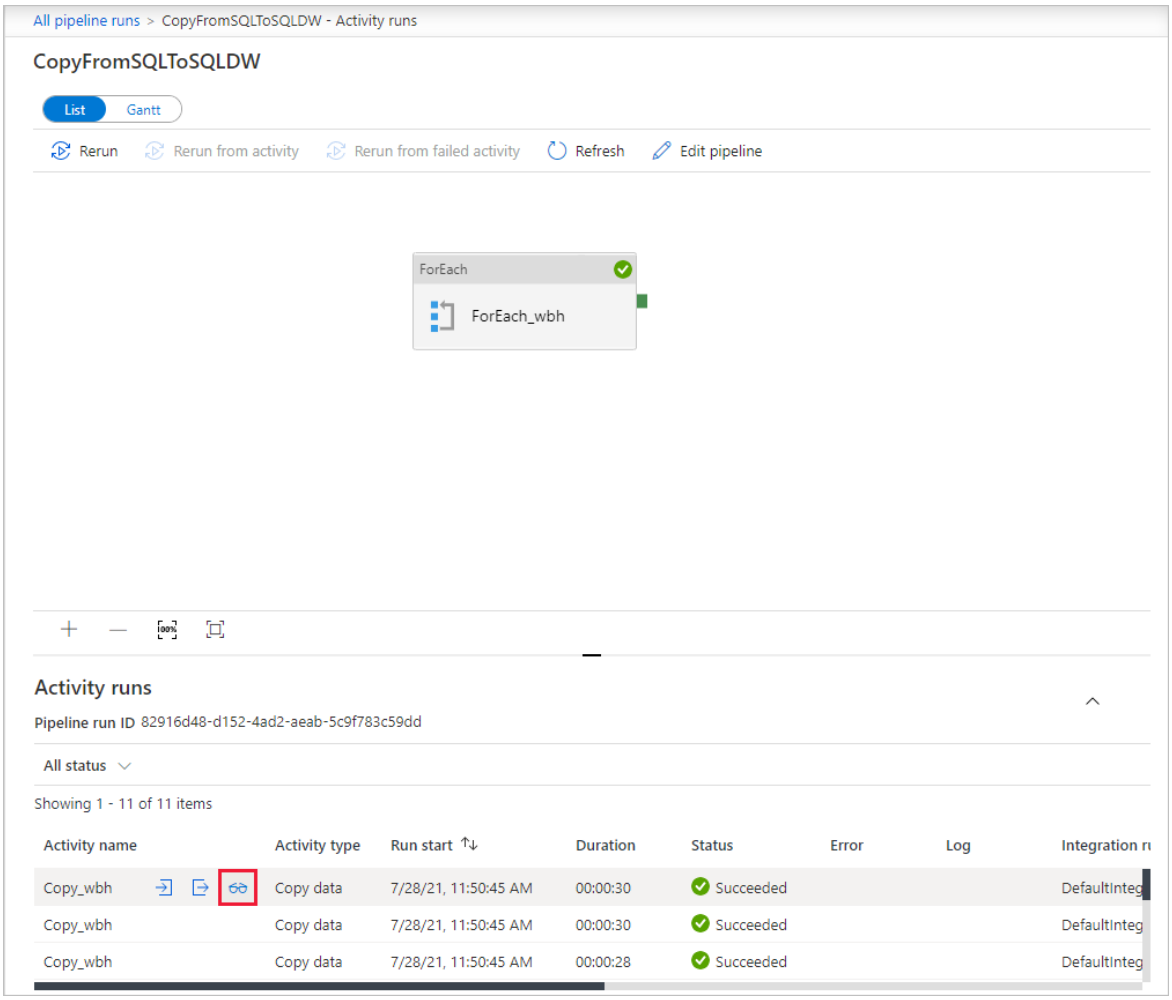
    - Azure Data Factory
    - Azure Synapse

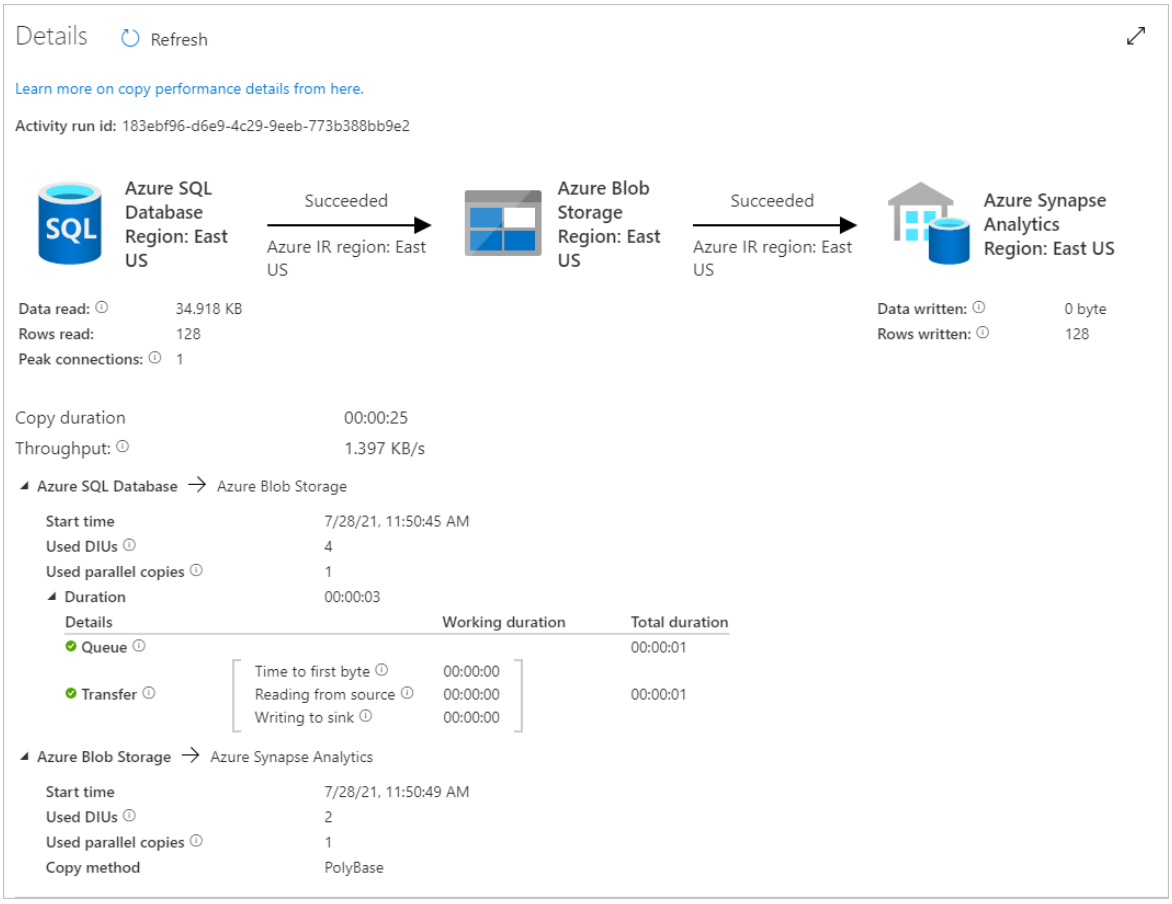


12. To switch back to the pipeline runs view, select the **All pipeline runs** link at the top. Select **Refresh** to refresh the list.



13. To monitor the execution details for each copy activity, select the **Details** link (eyeglasses icon) under **Activity name** in the activity runs view. You can monitor details like the volume of data copied from the source to the sink, data throughput, execution steps with corresponding duration, and used configurations.





Next steps

Advance to the following article to learn about Azure Synapse Analytics support:

Tutorial: Load the New York Taxicab dataset

8/19/2022 • 8 minutes to read • Edit Online

This tutorial uses the COPY statement to load New York Taxicab dataset from an Azure Blob Storage account. The tutorial uses the Azure portal and SQL Server Management Studio (SSMS) to:

- Create a user designated for loading data
- Create the tables for the sample dataset
- Use the COPY T-SQL statement to load data into your data warehouse
- View the progress of data as it is loading

If you don't have an Azure subscription, create a free account before you begin.

Before you begin

Before you begin this tutorial, download and install the newest version of SQL Server Management Studio (SSMS).

This tutorial assumes you have already created a SQL dedicated pool from the following tutorial.

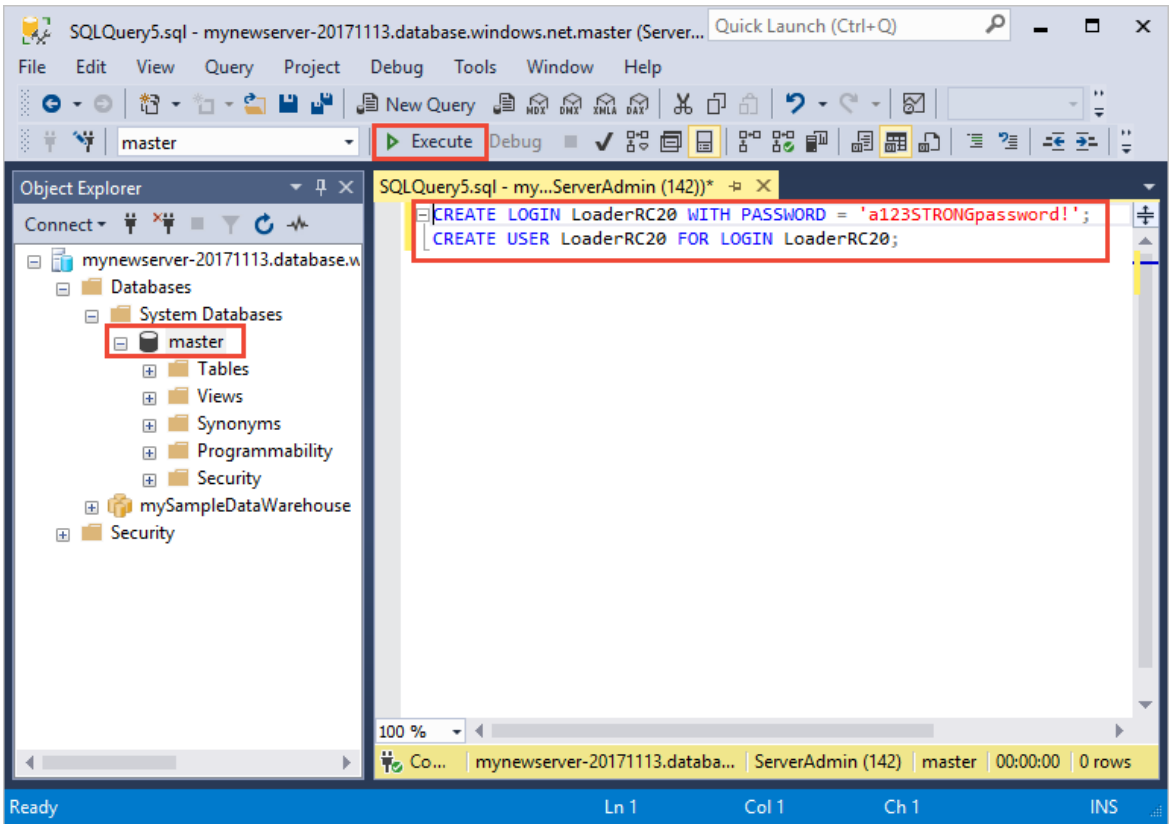
Create a user for loading data

The server admin account is meant to perform management operations, and is not suited for running queries on user data. Loading data is a memory-intensive operation. Memory maximums are defined according to the data warehouse units and resource class configured.

It's best to create a login and user that is dedicated for loading data. Then add the loading user to a resource class that enables an appropriate maximum memory allocation.

Connect as the server admin so you can create logins and users. Use these steps to create a login and user called **LoaderRC20**. Then assign the user to the **staticrc20** resource class.

1. In SSMS, right-select **master** to show a drop-down menu, and choose **New Query**. A new query window opens.

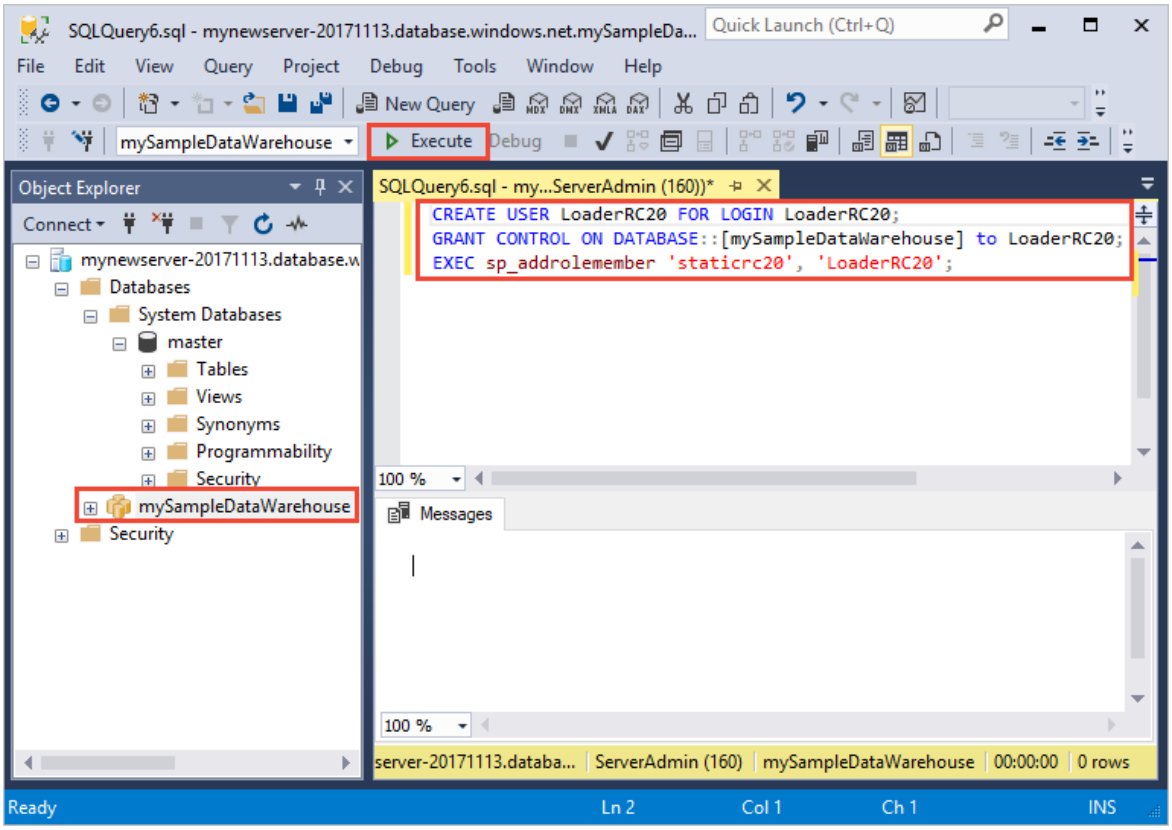


2. In the query window, enter these T-SQL commands to create a login and user named LoaderRC20, substituting your own password for 'a123STRONGpassword!'.

```
CREATE LOGIN LoaderRC20 WITH PASSWORD = 'a123STRONGpassword!';
CREATE USER LoaderRC20 FOR LOGIN LoaderRC20;
```

3. Select **Execute**.

4. Right-click **mySampleDataWarehouse**, and choose **New Query**. A new query Window opens.



5. Enter the following T-SQL commands to create a database user named LoaderRC20 for the LoaderRC20 login. The second line grants the new user CONTROL permissions on the new data warehouse. These permissions are similar to making the user the owner of the database. The third line adds the new user as a member of the staticrc20 resource class.

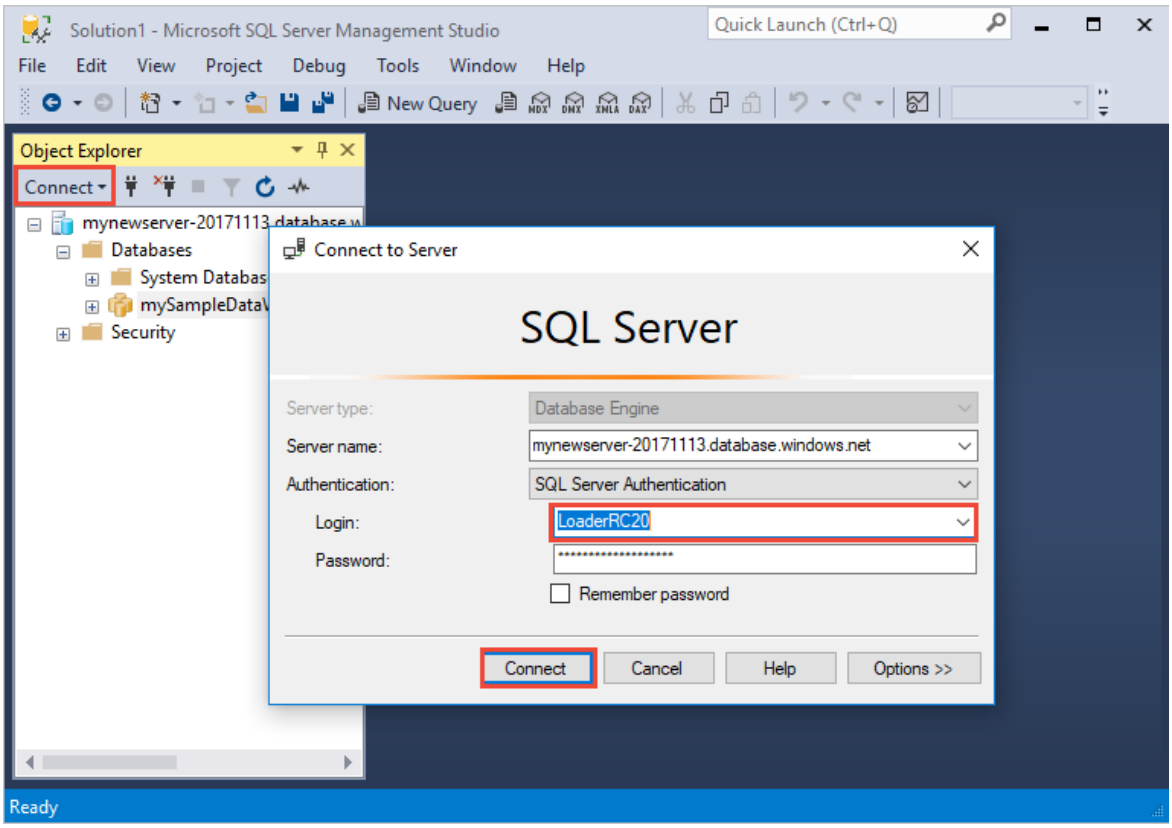
```
CREATE USER LoaderRC20 FOR LOGIN LoaderRC20;
GRANT CONTROL ON DATABASE::[mySampleDataWarehouse] to LoaderRC20;
EXEC sp_addrolemember 'staticrc20', 'LoaderRC20';
```

6. Select **Execute**.

Connect to the server as the loading user

The first step toward loading data is to login as LoaderRC20.

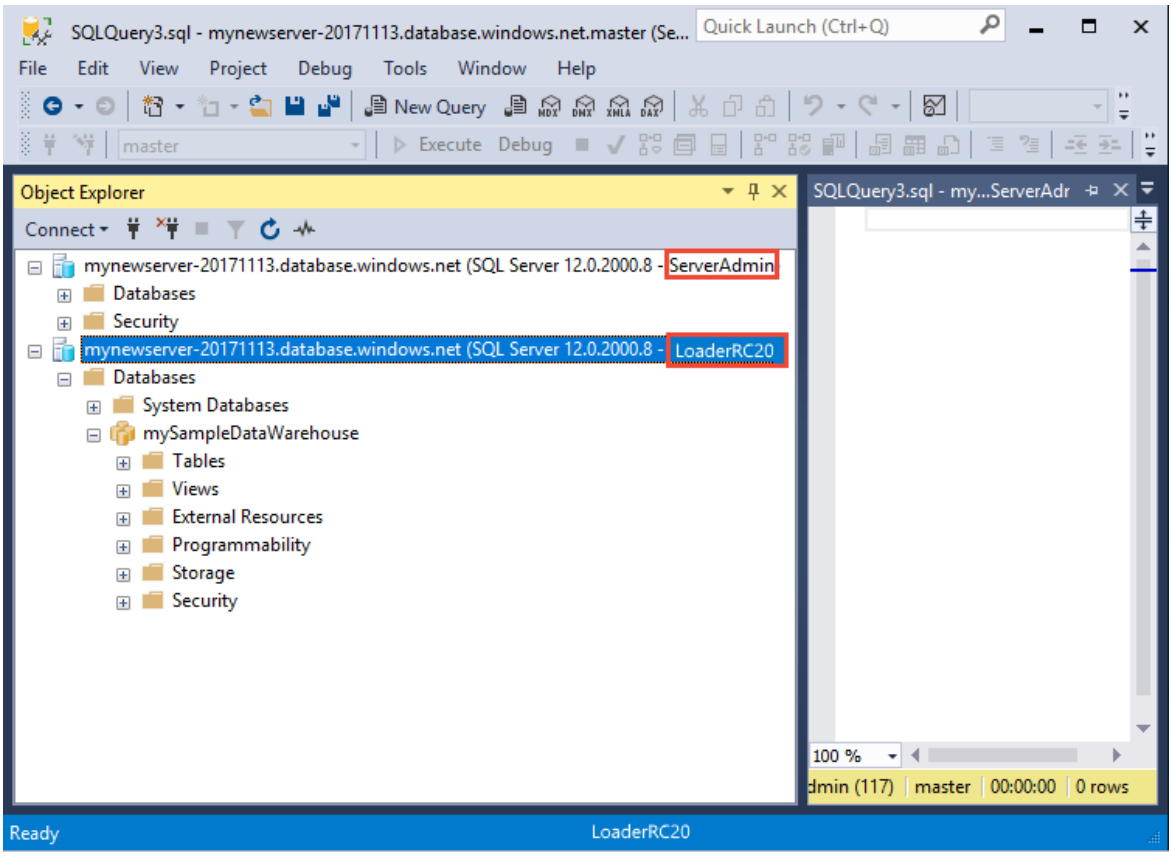
1. In Object Explorer, select the **Connect** drop down menu and select **Database Engine**. The **Connect to Server** dialog box appears.



2. Enter the fully qualified server name, and enter **LoaderRC20** as the Login. Enter your password for LoaderRC20.

3. Select **Connect**.

4. When your connection is ready, you will see two server connections in Object Explorer. One connection as ServerAdmin and one connection as MedRCLogin.

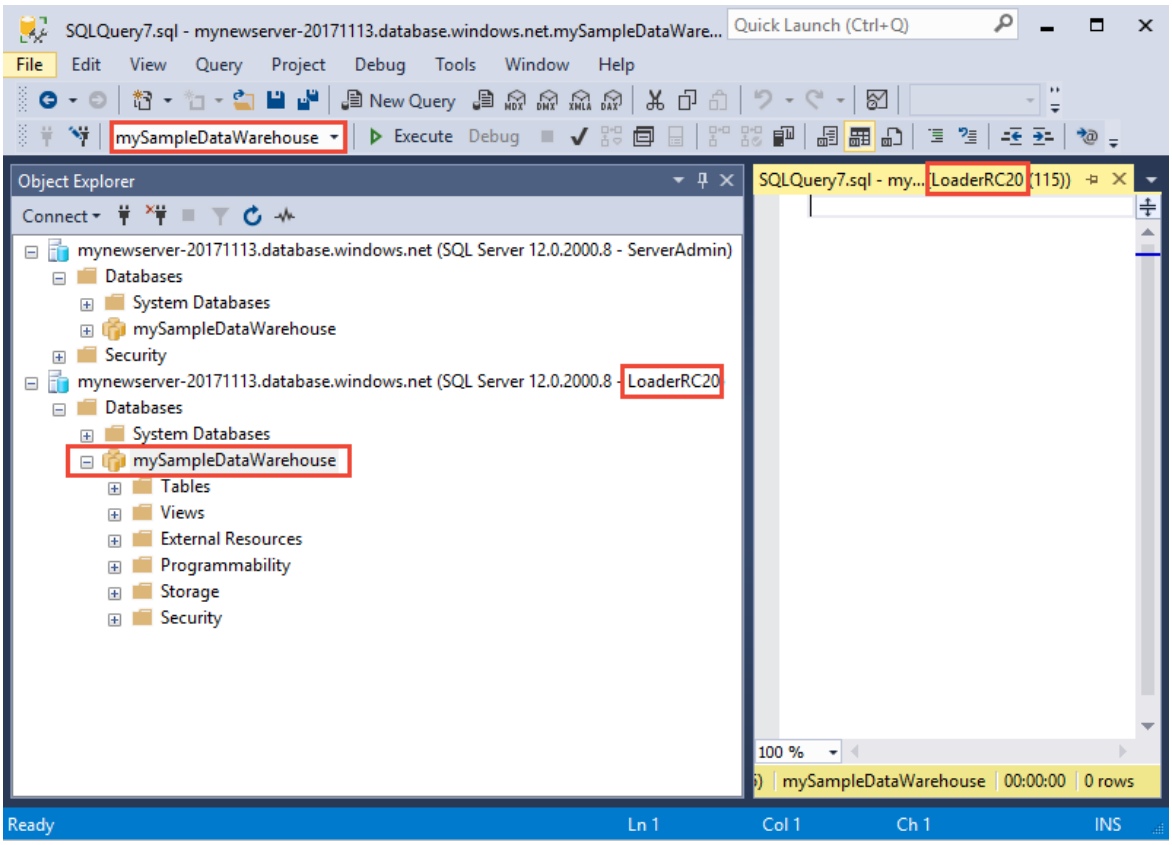


Create tables for the sample data

You are ready to begin the process of loading data into your new data warehouse. This part of the tutorial shows you how to use the COPY statement to load the New York City taxi cab dataset from an Azure Storage blob. For future reference, to learn how to get your data to Azure Blob Storage or to load it directly from your source, see the loading overview.

Run the following SQL scripts and specify information about the data you wish to load. This information includes where the data is located, the format of the contents of the data, and the table definition for the data.

1. In the previous section, you logged into your data warehouse as LoaderRC20. In SSMS, right-click your LoaderRC20 connection and select **New Query**. A new query window appears.



2. Compare your query window to the previous image. Verify your new query window is running as LoaderRC20 and performing queries on your MySampleDataWarehouse database. Use this query window to perform all of the loading steps.

3. Run the following T-SQL statements to create the tables:

```
CREATE TABLE [dbo].[Date]
(
    [DateID] int NOT NULL,
    [Date] datetime NULL,
    [DateBKey] char(10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfMonth] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DaySuffix] varchar(4) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayName] varchar(9) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfWeek] char(1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfWeekInMonth] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfWeekInYear] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfQuarter] varchar(3) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfYear] varchar(3) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [WeekOfMonth] varchar(1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [WeekOfQuarter] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [WeekOfYear] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [Month] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [MonthName] varchar(9) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [MonthOfQuarter] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [Quarter] char(1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [QuarterName] varchar(9) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [Year] char(4) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [YearName] char(7) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [MonthYear] char(10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [MMYYYY] char(6) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [FirstDayOfMonth] date NULL,
    [LastDayOfMonth] date NULL,
    [FirstDayOfQuarter] date NULL,
    [LastDayOfQuarter] date NULL,
    [FirstDayOfYear] date NULL,
    [LastDayOfYear] date NULL,
    [IsHolidayUSA] bit NULL,
    [IsWeekday] bit NULL
```

```
    [IsWeekday] bit NULL,
    [HolidayUSA] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
);

CREATE TABLE [dbo].[Geography]
(
    [GeographyID] int NOT NULL,
    [ZipCodeBKey] varchar(10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [County] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [City] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [State] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [Country] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [ZipCode] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
);

CREATE TABLE [dbo].[HackneyLicense]
(
    [HackneyLicenseID] int NOT NULL,
    [HackneyLicenseBKey] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [HackneyLicenseCode] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
);

CREATE TABLE [dbo].[Medallion]
(
    [MedallionID] int NOT NULL,
    [MedallionBKey] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [MedallionCode] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
);

CREATE TABLE [dbo].[Time]
(
    [TimeID] int NOT NULL,
    [TimeBKey] varchar(8) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [HourNumber] tinyint NOT NULL,
    [MinuteNumber] tinyint NOT NULL,
    [SecondNumber] tinyint NOT NULL,
    [TimeInSecond] int NOT NULL,
    [HourlyBucket] varchar(15) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [DayTimeBucketGroupKey] int NOT NULL,
    [DayTimeBucket] varchar(100) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
);

CREATE TABLE [dbo].[Trip]
(
    [DateID] int NOT NULL,
```

```
    [DateID] int NOT NULL,
    [MedallionID] int NOT NULL,
    [HackneyLicenseID] int NOT NULL,
    [PickupTimeID] int NOT NULL,
    [DropoffTimeID] int NOT NULL,
    [PickupGeographyID] int NULL,
    [DropoffGeographyID] int NULL,
    [PickupLatitude] float NULL,
    [PickupLongitude] float NULL,
    [PickupLatLong] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DropoffLatitude] float NULL,
    [DropoffLongitude] float NULL,
    [DropoffLatLong] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [PassengerCount] int NULL,
    [TripDurationSeconds] int NULL,
    [TripDistanceMiles] float NULL,
    [PaymentType] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [FareAmount] money NULL,
    [SurchargeAmount] money NULL,
    [TaxAmount] money NULL,
    [TipAmount] money NULL,
    [TollsAmount] money NULL,
    [TotalAmount] money NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
);

CREATE TABLE [dbo].[Weather]
(
    [DateID] int NOT NULL,
    [GeographyID] int NOT NULL,
    [PrecipitationInches] float NOT NULL,
    [AvgTemperatureFahrenheit] float NOT NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
);
```

Load the data into your data warehouse

This section uses the COPY statement to load the sample data from Azure Storage Blob.

> **NOTE**
>
> This tutorial loads the data directly into the final table. You would typically load into a staging table for your production workloads. While data is in the staging table you can perform any necessary transformations.

1. Run the following statements to load the data:

```
COPY INTO [dbo].[Date]
FROM 'https://nytaxiblob.blob.core.windows.net/2013/Date'
WITH
(
    FILE_TYPE = 'CSV',
 FIELDTERMINATOR = ',',
 FIELDQUOTE = ''
)
OPTION (LABEL = 'COPY : Load [dbo].[Date] - Taxi dataset');
```

```
COPY INTO [dbo].[Geography]
FROM 'https://nytaxiblob.blob.core.windows.net/2013/Geography'
WITH
(
    FILE_TYPE = 'CSV',
 FIELDTERMINATOR = ',',
 FIELDQUOTE = ''
)
OPTION (LABEL = 'COPY : Load [dbo].[Geography] - Taxi dataset');

COPY INTO [dbo].[HackneyLicense]
FROM 'https://nytaxiblob.blob.core.windows.net/2013/HackneyLicense'
WITH
(
    FILE_TYPE = 'CSV',
 FIELDTERMINATOR = ',',
 FIELDQUOTE = ''
)
OPTION (LABEL = 'COPY : Load [dbo].[HackneyLicense] - Taxi dataset');

COPY INTO [dbo].[Medallion]
FROM 'https://nytaxiblob.blob.core.windows.net/2013/Medallion'
WITH
(
    FILE_TYPE = 'CSV',
 FIELDTERMINATOR = ',',
 FIELDQUOTE = ''
)
OPTION (LABEL = 'COPY : Load [dbo].[Medallion] - Taxi dataset');

COPY INTO [dbo].[Time]
FROM 'https://nytaxiblob.blob.core.windows.net/2013/Time'
WITH
(
    FILE_TYPE = 'CSV',
 FIELDTERMINATOR = ',',
 FIELDQUOTE = ''
)
OPTION (LABEL = 'COPY : Load [dbo].[Time] - Taxi dataset');

COPY INTO [dbo].[Weather]
FROM 'https://nytaxiblob.blob.core.windows.net/2013/Weather'
WITH
(
    FILE_TYPE = 'CSV',
 FIELDTERMINATOR = ',',
 FIELDQUOTE = '',
 ROWTERMINATOR='0X0A'
)
OPTION (LABEL = 'COPY : Load [dbo].[Weather] - Taxi dataset');

COPY INTO [dbo].[Trip]
FROM 'https://nytaxiblob.blob.core.windows.net/2013/Trip2013'
WITH
(
    FILE_TYPE = 'CSV',
 FIELDTERMINATOR = '|',
 FIELDQUOTE = '',
 ROWTERMINATOR='0X0A',
 COMPRESSION = 'GZIP'
)
OPTION (LABEL = 'COPY : Load [dbo].[Trip] - Taxi dataset');
```

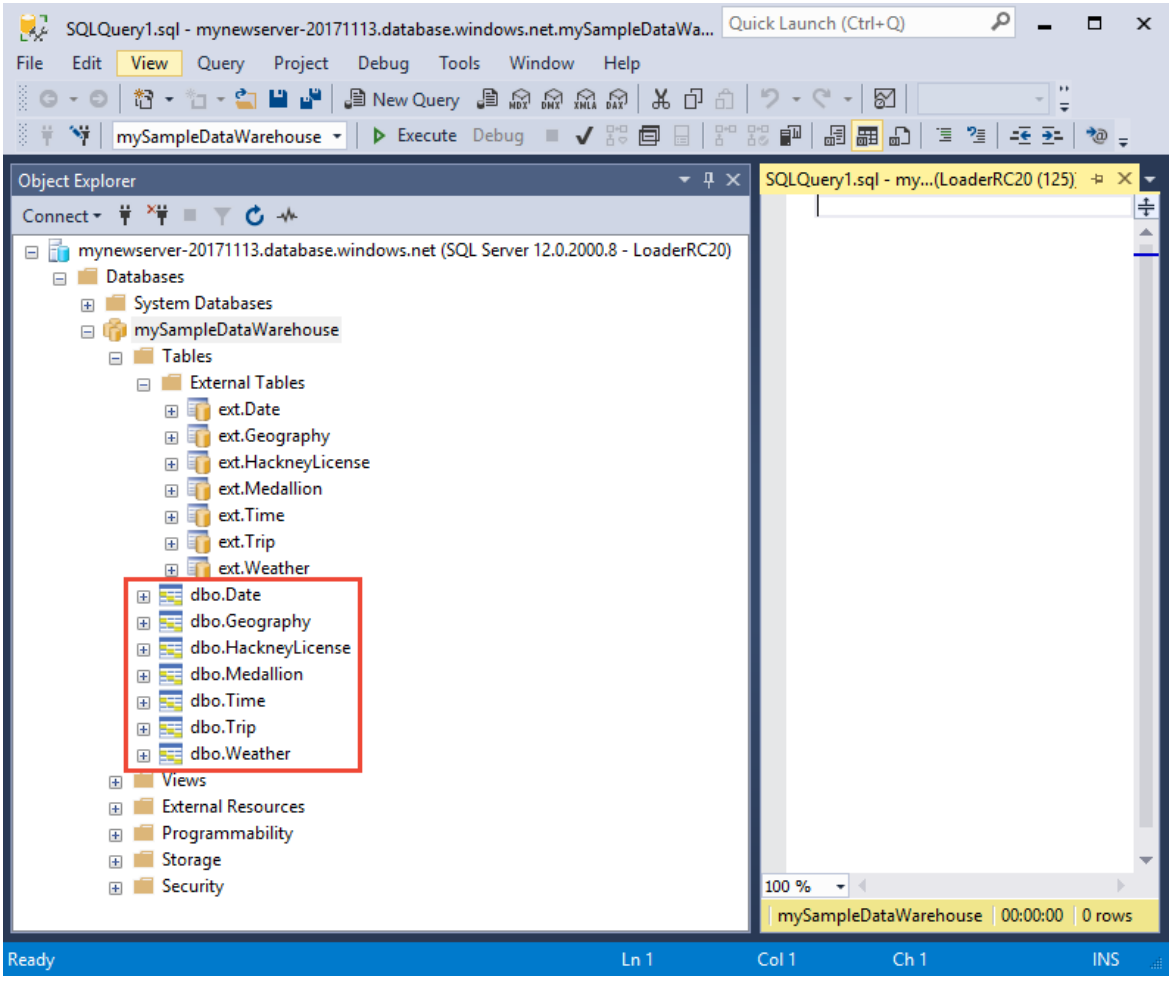
2. View your data as it loads. You're loading several GBs of data and compressing it into highly performant clustered columnstore indexes. Run the following query that uses a dynamic management views (DMVs) to show the status of the load.

```
SELECT  r.[request_id]
,       r.[status]
,       r.resource_class
,       r.command
,       sum(bytes_processed) AS bytes_processed
,       sum(rows_processed) AS rows_processed
FROM    sys.dm_pdw_exec_requests r
              JOIN sys.dm_pdw_dms_workers w
                     ON r.[request_id] = w.request_id
WHERE [label] = 'COPY : Load [dbo].[Date] - Taxi dataset' OR
    [label] = 'COPY : Load [dbo].[Geography] - Taxi dataset' OR
    [label] = 'COPY : Load [dbo].[HackneyLicense] - Taxi dataset' OR
    [label] = 'COPY : Load [dbo].[Medallion] - Taxi dataset' OR
    [label] = 'COPY : Load [dbo].[Time] - Taxi dataset' OR
    [label] = 'COPY : Load [dbo].[Weather] - Taxi dataset' OR
    [label] = 'COPY : Load [dbo].[Trip] - Taxi dataset'
and session_id <> session_id() and type = 'WRITER'
GROUP BY r.[request_id]
,       r.[status]
,       r.resource_class
,       r.command;
```

3. View all system queries.

```
SELECT * FROM sys.dm_pdw_exec_requests;
```

4. Enjoy seeing your data nicely loaded into your data warehouse.



Clean up resources

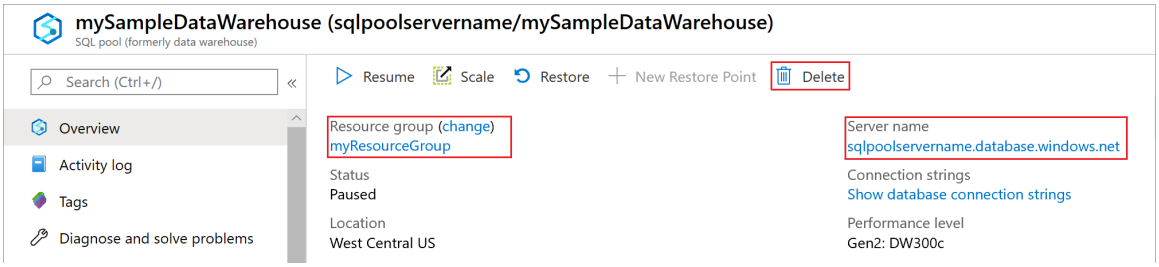
You are being charged for compute resources and data that you loaded into your data warehouse. These are

billed separately.

- If you want to keep the data in storage, you can pause compute when you aren't using the data warehouse. By pausing compute you will only be charge for data storage and you can resume the compute whenever you are ready to work with the data.
- If you want to remove future charges, you can delete the data warehouse.

Follow these steps to clean up resources as you desire.

1. Log in to the Azure portal, select your data warehouse.



2. To pause compute, select the **Pause** button. When the data warehouse is paused, you will see a **Start** button. To resume compute, select **Start**.
3. To remove the data warehouse so you won't be charged for compute or storage, select **Delete**.
4. To remove the server you created, select **mynewserver-20180430.database.windows.net** in the previous image, and then select **Delete**. Be careful with this as deleting the server will delete all databases assigned to the server.
5. To remove the resource group, select **myResourceGroup**, and then select **Delete resource group**.

Next steps

In this tutorial, you learned how to create a data warehouse and create a user for loading data. You used the simple COPY statement to load data into your data warehouse.

You did these things:

- Created a data warehouse in the Azure portal
- Set up a server-level firewall rule in the Azure portal
- Connected to the data warehouse with SSMS
- Created a user designated for loading data
- Created the tables for the sample data
- Used the COPY T-SQL statement to load data into your data warehouse
- Viewed the progress of data as it is loading

Advance to the development overview to learn how to migrate an existing database to Azure Synapse Analytics:

Design decisions to migrate an existing database to Azure Synapse Analytics

For more loading examples and references, view the following documentation:

- COPY statement reference documentation
- COPY examples for each authentication method
- COPY quickstart for a single table

Load Contoso retail data into dedicated SQL pools in Azure Synapse Analytics

8/19/2022 • 8 minutes to read • Edit Online

In this tutorial, you learn to use PolyBase and T-SQL commands to load two tables from the Contoso retail data into dedicated SQL pools.

In this tutorial you will:

1. Configure PolyBase to load from Azure blob storage
2. Load public data into your database
3. Perform optimizations after the load is finished.

Before you begin

To run this tutorial, you need an Azure account that already has a dedicated SQL pool. If you don't have a data warehouse provisioned, see Create a data warehouse and set server-level firewall rule.

Configure the data source

PolyBase uses T-SQL external objects to define the location and attributes of the external data. The external object definitions are stored in dedicated SQL pools. The data is stored externally.

Create a credential

Skip this step if you're loading the Contoso public data. You don't need secure access to the public data since it's already accessible to anyone.

Don't skip this step if you're using this tutorial as a template for loading your own data. To access data through a credential, use the following script to create a database-scoped credential. Then use it when defining the location of the data source.

```
-- A: Create a master key.
-- Only necessary if one does not already exist.
-- Required to encrypt the credential secret in the next step.

CREATE MASTER KEY;


-- B: Create a database scoped credential
-- IDENTITY: Provide any string, it is not used for authentication to Azure storage.
-- SECRET: Provide your Azure storage account key.


CREATE DATABASE SCOPED CREDENTIAL AzureStorageCredential
WITH
    IDENTITY = 'user',
    SECRET = '<azure_storage_account_key>'
;


-- C: Create an external data source
-- TYPE: HADOOP - PolyBase uses Hadoop APIs to access data in Azure blob storage.
-- LOCATION: Provide Azure storage account name and blob container name.
-- CREDENTIAL: Provide the credential created in the previous step.

CREATE EXTERNAL DATA SOURCE AzureStorage
WITH (
    TYPE = HADOOP,
    LOCATION = 'wasbs://<blob_container_name>@<azure_storage_account_name>.blob.core.windows.net',
    CREDENTIAL = AzureStorageCredential
);
```

Create the external data source

Use this CREATE EXTERNAL DATA SOURCE command to store the location of the data, and the data type.

```
CREATE EXTERNAL DATA SOURCE AzureStorage_west_public
WITH
(
    TYPE = Hadoop
,   LOCATION = 'wasbs://contosoretaildw-tables@contosoretaildw.blob.core.windows.net/'
);
```

> **IMPORTANT**
>
> If you choose to make your azure blob storage containers public, remember that as the data owner you will be charged for data egress charges when data leaves the data center.

Configure the data format

The data is stored in text files in Azure blob storage, and each field is separated with a delimiter. In SSMS, run the following CREATE EXTERNAL FILE FORMAT command to specify the format of the data in the text files. The Contoso data is uncompressed and pipe delimited.

```
CREATE EXTERNAL FILE FORMAT TextFileFormat
WITH
(   FORMAT_TYPE = DELIMITEDTEXT
,    FORMAT_OPTIONS    (   FIELD_TERMINATOR = '|'
                     ,    STRING_DELIMITER = ''
                     ,    DATE_FORMAT         = 'yyyy-MM-dd HH:mm:ss.fff'
                     ,    USE_TYPE_DEFAULT = FALSE
                     )
);
```

Create the schema for the external tables

Now that you've specified the data source and file format, you're ready to create the schema for the external tables.

To create a place to store the Contoso data in your database, create a schema.

```
CREATE SCHEMA [asb]
GO
```

Create the external tables

Run the following script to create the DimProduct and FactOnlineSales external tables. All you're doing here is defining column names and data types, and binding them to the location and format of the Azure blob storage files. The definition is stored in the data warehouse and the data is still in the Azure Storage Blob.

The **LOCATION** parameter is the folder under the root folder in the Azure Storage Blob. Each table is in a different folder.

```
--DimProduct
CREATE EXTERNAL TABLE [asb].DimProduct (
    [ProductKey] [int] NOT NULL,
    [ProductLabel] [nvarchar](255) NULL,
    [ProductName] [nvarchar](500) NULL,
    [ProductDescription] [nvarchar](400) NULL,
    [ProductSubcategoryKey] [int] NULL,
    [Manufacturer] [nvarchar](50) NULL,
    [BrandName] [nvarchar](50) NULL,
    [ClassID] [nvarchar](10) NULL,
    [ClassName] [nvarchar](20) NULL,
    [StyleID] [nvarchar](10) NULL,
    [StyleName] [nvarchar](20) NULL,
    [ColorID] [nvarchar](10) NULL,
    [ColorName] [nvarchar](20) NOT NULL,
    [Size] [nvarchar](50) NULL,
    [SizeRange] [nvarchar](50) NULL,
    [SizeUnitMeasureID] [nvarchar](20) NULL,
    [Weight] [float] NULL,
    [WeightUnitMeasureID] [nvarchar](20) NULL,
    [UnitOfMeasureID] [nvarchar](10) NULL,
    [UnitOfMeasureName] [nvarchar](40) NULL,
    [StockTypeID] [nvarchar](10) NULL,
    [StockTypeName] [nvarchar](40) NULL,
    [UnitCost] [money] NULL,
    [UnitPrice] [money] NULL,
    [AvailableForSaleDate] [datetime] NULL,
    [StopSaleDate] [datetime] NULL,
    [Status] [nvarchar](7) NULL,
    [ImageURL] [nvarchar](150) NULL,
    [ProductURL] [nvarchar](150) NULL,
    [ETLLoadID] [int] NULL,
```

```
    [LoadDate] [datetime] NULL,
    [UpdateDate] [datetime] NULL
)
WITH
(
    LOCATION='/DimProduct/'
,   DATA_SOURCE = AzureStorage_west_public
,   FILE_FORMAT = TextFileFormat
,   REJECT_TYPE = VALUE
,   REJECT_VALUE = 0
)
;

--FactOnlineSales
CREATE EXTERNAL TABLE [asb].FactOnlineSales
(
    [OnlineSalesKey] [int]  NOT NULL,
    [DateKey] [datetime] NOT NULL,
    [StoreKey] [int] NOT NULL,
    [ProductKey] [int] NOT NULL,
    [PromotionKey] [int] NOT NULL,
    [CurrencyKey] [int] NOT NULL,
    [CustomerKey] [int] NOT NULL,
    [SalesOrderNumber] [nvarchar](20) NOT NULL,
    [SalesOrderLineNumber] [int] NULL,
    [SalesQuantity] [int] NOT NULL,
    [SalesAmount] [money] NOT NULL,
    [ReturnQuantity] [int] NOT NULL,
    [ReturnAmount] [money] NULL,
    [DiscountQuantity] [int] NULL,
    [DiscountAmount] [money] NULL,
    [TotalCost] [money] NOT NULL,
    [UnitCost] [money] NULL,
    [UnitPrice] [money] NULL,
    [ETLLoadID] [int] NULL,
    [LoadDate] [datetime] NULL,
    [UpdateDate] [datetime] NULL
)
WITH
(
    LOCATION='/FactOnlineSales/'
,   DATA_SOURCE = AzureStorage_west_public
,   FILE_FORMAT = TextFileFormat
,   REJECT_TYPE = VALUE
,   REJECT_VALUE = 0
)
;
```

Load the data

There are different ways to access external data. You can query data directly from the external tables, load the data into new tables in the data warehouse, or add external data to existing data warehouse tables.

Create a new schema

CTAS creates a new table that contains data. First, create a schema for the contoso data.

```
CREATE SCHEMA [cso]
GO
```

Load the data into new tables

To load data from Azure blob storage into the data warehouse table, use the CREATE TABLE AS SELECT (Transact-SQL) statement. Loading with CTAS leverages the strongly typed external tables you've created. To load the data

into new tables, use one CTAS statement per table.

CTAS creates a new table and populates it with the results of a select statement. CTAS defines the new table to have the same columns and data types as the results of the select statement. If you select all the columns from an external table, the new table will be a replica of the columns and data types in the external table.

In this example, we create both the dimension and the fact table as hash distributed tables.

```
SELECT GETDATE();
GO

CREATE TABLE [cso].[DimProduct]            WITH (DISTRIBUTION = HASH([ProductKey]  ) ) AS SELECT * FROM
[asb].[DimProduct]             OPTION (LABEL = 'CTAS : Load [cso].[DimProduct]             ');
CREATE TABLE [cso].[FactOnlineSales]       WITH (DISTRIBUTION = HASH([ProductKey]  ) ) AS SELECT * FROM
[asb].[FactOnlineSales]        OPTION (LABEL = 'CTAS : Load [cso].[FactOnlineSales]        ');
```

Track the load progress

You can track the progress of your load using dynamic management views (DMVs).

```
-- To see all requests
SELECT * FROM sys.dm_pdw_exec_requests;

-- To see a particular request identified by its label
SELECT * FROM sys.dm_pdw_exec_requests as r
WHERE r.[label] = 'CTAS : Load [cso].[DimProduct]             '
      OR r.[label] = 'CTAS : Load [cso].[FactOnlineSales]        '
;

-- To track bytes and files
SELECT
    r.command,
    s.request_id,
    r.status,
    count(distinct input_name) as nbr_files,
    sum(s.bytes_processed)/1024/1024/1024 as gb_processed
FROM
    sys.dm_pdw_exec_requests r
    inner join sys.dm_pdw_dms_external_work s
        on r.request_id = s.request_id
WHERE
    r.[label] = 'CTAS : Load [cso].[DimProduct]             '
    OR r.[label] = 'CTAS : Load [cso].[FactOnlineSales]        '
GROUP BY
    r.command,
    s.request_id,
    r.status
ORDER BY
    nbr_files desc,
    gb_processed desc;
```

Optimize columnstore compression

By default, dedicated SQL pools store the table as a clustered columnstore index. After a load completes, some of the data rows might not be compressed into the columnstore. There are different reasons why this can happen. To learn more, see manage columnstore indexes.

To optimize query performance and columnstore compression after a load, rebuild the table to force the columnstore index to compress all the rows.

```
SELECT GETDATE();
GO

ALTER INDEX ALL ON [cso].[DimProduct]               REBUILD;
ALTER INDEX ALL ON [cso].[FactOnlineSales]          REBUILD;
```

For more information on maintaining columnstore indexes, see the manage columnstore indexes article.

Optimize statistics

It's best to create single-column statistics immediately after a load. If you know certain columns aren't going to be in query predicates, you can skip creating statistics on those columns. If you create single-column statistics on every column, it might take a long time to rebuild all the statistics.

If you decide to create single-column statistics on every column of every table, you can use the stored procedure code sample `prc_sqldw_create_stats` in the statistics article.

The following example is a good starting point for creating statistics. It creates single-column statistics on each column in the dimension table, and on each joining column in the fact tables. You can always add single or multi-column statistics to other fact table columns later on.

```
CREATE STATISTICS [stat_cso_DimProduct_AvailableForSaleDate] ON [cso].[DimProduct]([AvailableForSaleDate]);
CREATE STATISTICS [stat_cso_DimProduct_BrandName] ON [cso].[DimProduct]([BrandName]);
CREATE STATISTICS [stat_cso_DimProduct_ClassID] ON [cso].[DimProduct]([ClassID]);
CREATE STATISTICS [stat_cso_DimProduct_ClassName] ON [cso].[DimProduct]([ClassName]);
CREATE STATISTICS [stat_cso_DimProduct_ColorID] ON [cso].[DimProduct]([ColorID]);
CREATE STATISTICS [stat_cso_DimProduct_ColorName] ON [cso].[DimProduct]([ColorName]);
CREATE STATISTICS [stat_cso_DimProduct_ETLLoadID] ON [cso].[DimProduct]([ETLLoadID]);
CREATE STATISTICS [stat_cso_DimProduct_ImageURL] ON [cso].[DimProduct]([ImageURL]);
CREATE STATISTICS [stat_cso_DimProduct_LoadDate] ON [cso].[DimProduct]([LoadDate]);
CREATE STATISTICS [stat_cso_DimProduct_Manufacturer] ON [cso].[DimProduct]([Manufacturer]);
CREATE STATISTICS [stat_cso_DimProduct_ProductDescription] ON [cso].[DimProduct]([ProductDescription]);
CREATE STATISTICS [stat_cso_DimProduct_ProductKey] ON [cso].[DimProduct]([ProductKey]);
CREATE STATISTICS [stat_cso_DimProduct_ProductLabel] ON [cso].[DimProduct]([ProductLabel]);
CREATE STATISTICS [stat_cso_DimProduct_ProductName] ON [cso].[DimProduct]([ProductName]);
CREATE STATISTICS [stat_cso_DimProduct_ProductSubcategoryKey] ON [cso].[DimProduct]
([ProductSubcategoryKey]);
CREATE STATISTICS [stat_cso_DimProduct_ProductURL] ON [cso].[DimProduct]([ProductURL]);
CREATE STATISTICS [stat_cso_DimProduct_Size] ON [cso].[DimProduct]([Size]);
CREATE STATISTICS [stat_cso_DimProduct_SizeRange] ON [cso].[DimProduct]([SizeRange]);
CREATE STATISTICS [stat_cso_DimProduct_SizeUnitMeasureID] ON [cso].[DimProduct]([SizeUnitMeasureID]);
CREATE STATISTICS [stat_cso_DimProduct_Status] ON [cso].[DimProduct]([Status]);
CREATE STATISTICS [stat_cso_DimProduct_StockTypeID] ON [cso].[DimProduct]([StockTypeID]);
CREATE STATISTICS [stat_cso_DimProduct_StockTypeName] ON [cso].[DimProduct]([StockTypeName]);
CREATE STATISTICS [stat_cso_DimProduct_StopSaleDate] ON [cso].[DimProduct]([StopSaleDate]);
CREATE STATISTICS [stat_cso_DimProduct_StyleID] ON [cso].[DimProduct]([StyleID]);
CREATE STATISTICS [stat_cso_DimProduct_StyleName] ON [cso].[DimProduct]([StyleName]);
CREATE STATISTICS [stat_cso_DimProduct_UnitCost] ON [cso].[DimProduct]([UnitCost]);
CREATE STATISTICS [stat_cso_DimProduct_UnitOfMeasureID] ON [cso].[DimProduct]([UnitOfMeasureID]);
CREATE STATISTICS [stat_cso_DimProduct_UnitOfMeasureName] ON [cso].[DimProduct]([UnitOfMeasureName]);
CREATE STATISTICS [stat_cso_DimProduct_UnitPrice] ON [cso].[DimProduct]([UnitPrice]);
CREATE STATISTICS [stat_cso_DimProduct_UpdateDate] ON [cso].[DimProduct]([UpdateDate]);
CREATE STATISTICS [stat_cso_DimProduct_Weight] ON [cso].[DimProduct]([Weight]);
CREATE STATISTICS [stat_cso_DimProduct_WeightUnitMeasureID] ON [cso].[DimProduct]([WeightUnitMeasureID]);
CREATE STATISTICS [stat_cso_FactOnlineSales_CurrencyKey] ON [cso].[FactOnlineSales]([CurrencyKey]);
CREATE STATISTICS [stat_cso_FactOnlineSales_CustomerKey] ON [cso].[FactOnlineSales]([CustomerKey]);
CREATE STATISTICS [stat_cso_FactOnlineSales_DateKey] ON [cso].[FactOnlineSales]([DateKey]);
CREATE STATISTICS [stat_cso_FactOnlineSales_OnlineSalesKey] ON [cso].[FactOnlineSales]([OnlineSalesKey]);
CREATE STATISTICS [stat_cso_FactOnlineSales_ProductKey] ON [cso].[FactOnlineSales]([ProductKey]);
CREATE STATISTICS [stat_cso_FactOnlineSales_PromotionKey] ON [cso].[FactOnlineSales]([PromotionKey]);
CREATE STATISTICS [stat_cso_FactOnlineSales_StoreKey] ON [cso].[FactOnlineSales]([StoreKey]);
```

Achievement unlocked!

You have successfully loaded public data into your data warehouse. Great job!

You can now start querying the tables to explore your data. Run the following query to find out total sales per brand:

```
SELECT  SUM(f.[SalesAmount]) AS [sales_by_brand_amount]
,       p.[BrandName]
FROM    [cso].[FactOnlineSales] AS f
JOIN    [cso].[DimProduct]      AS p ON f.[ProductKey] = p.[ProductKey]
GROUP BY p.[BrandName]
```

Next steps

To load the full data set, run the example load the full Contoso retail data warehouse from the Microsoft SQL Server samples repository. For more development tips, see Design decisions and coding techniques for data warehouses.

Tutorial: Load data to Azure Synapse Analytics SQL pool

8/19/2022 • 26 minutes to read • Edit Online

This tutorial uses PolyBase to load the WideWorldImportersDW data warehouse from Azure Blob storage to your data warehouse in Azure Synapse Analytics SQL pool. The tutorial uses the Azure portal and SQL Server Management Studio (SSMS) to:

- Create a user designated for loading data
- Create external tables that use Azure blob as the data source
- Use the CTAS T-SQL statement to load data into your data warehouse
- View the progress of data as it is loading
- Generate a year of data in the date dimension and sales fact tables
- Create statistics on the newly loaded data

If you don't have an Azure subscription, create a free account before you begin.

Before you begin

Before you begin this tutorial, download and install the newest version of SQL Server Management Studio (SSMS).

This tutorial assumes you have already created a SQL dedicated pool from the following tutorial.

> **NOTE**
> It is recommended to use at least a DW1000c for this tutorial.

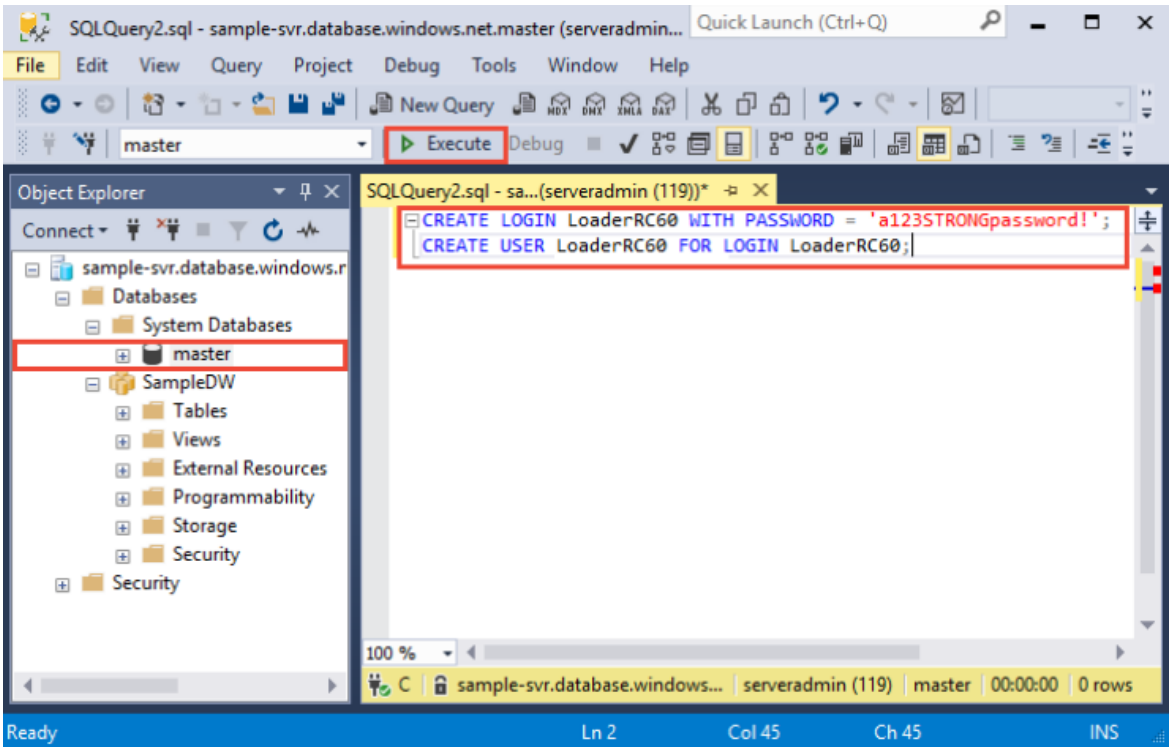
Create a user for loading data

The server admin account is meant to perform management operations, and is not suited for running queries on user data. Loading data is a memory-intensive operation. Memory maximums are defined according to the Generation of SQL pool you're using, data warehouse units, and resource class.

It's best to create a login and user that is dedicated for loading data. Then add the loading user to a resource class that enables an appropriate maximum memory allocation.

Since you are currently connected as the server admin, you can create logins and users. Use these steps to create a login and user called **LoaderRC60**. Then assign the user to the **staticrc60** resource class.

1. In SSMS, right-click **master** to show a drop-down menu, and choose **New Query**. A new query window opens.

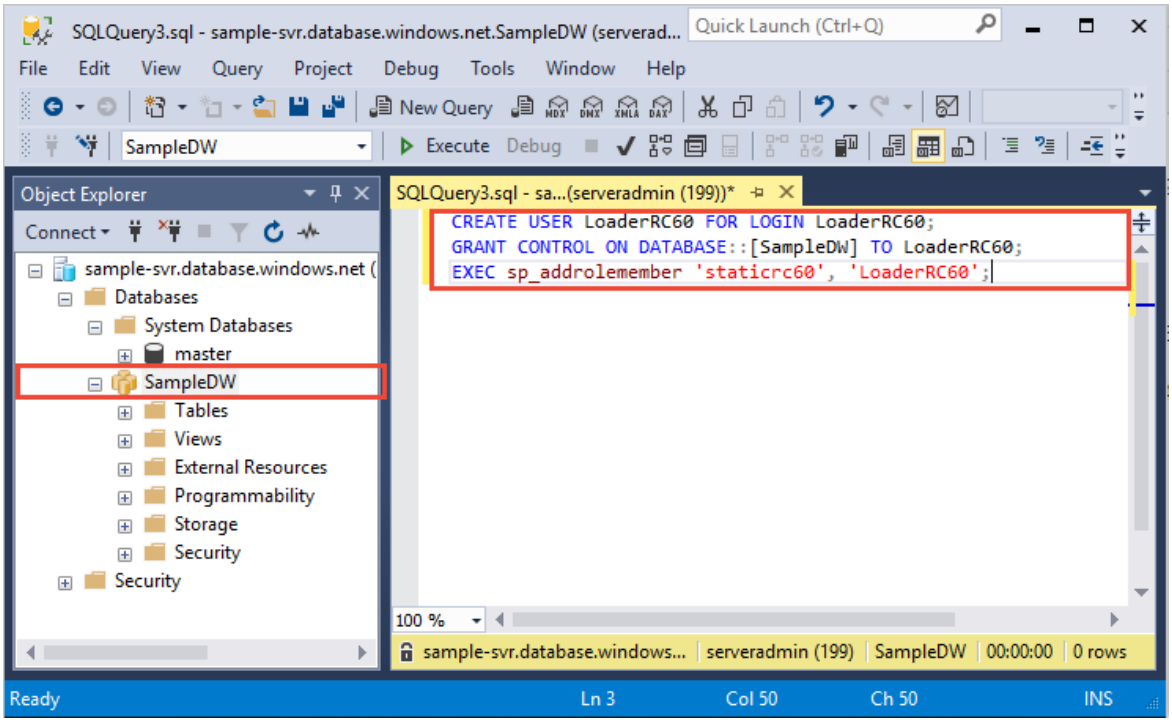


2. In the query window, enter these T-SQL commands to create a login and user named LoaderRC60, substituting your own password for 'a123STRONGpassword!'.

```
CREATE LOGIN LoaderRC60 WITH PASSWORD = 'a123STRONGpassword!';
CREATE USER LoaderRC60 FOR LOGIN LoaderRC60;
```

3. Click **Execute**.

4. Right-click **SampleDW**, and choose **New Query**. A new query Window opens.



5. Enter the following T-SQL commands to create a database user named LoaderRC60 for the LoaderRC60 login. The second line grants the new user CONTROL permissions on the new data warehouse. These permissions are similar to making the user the owner of the database. The third line adds the new user as a member of the `staticrc60` resource class.

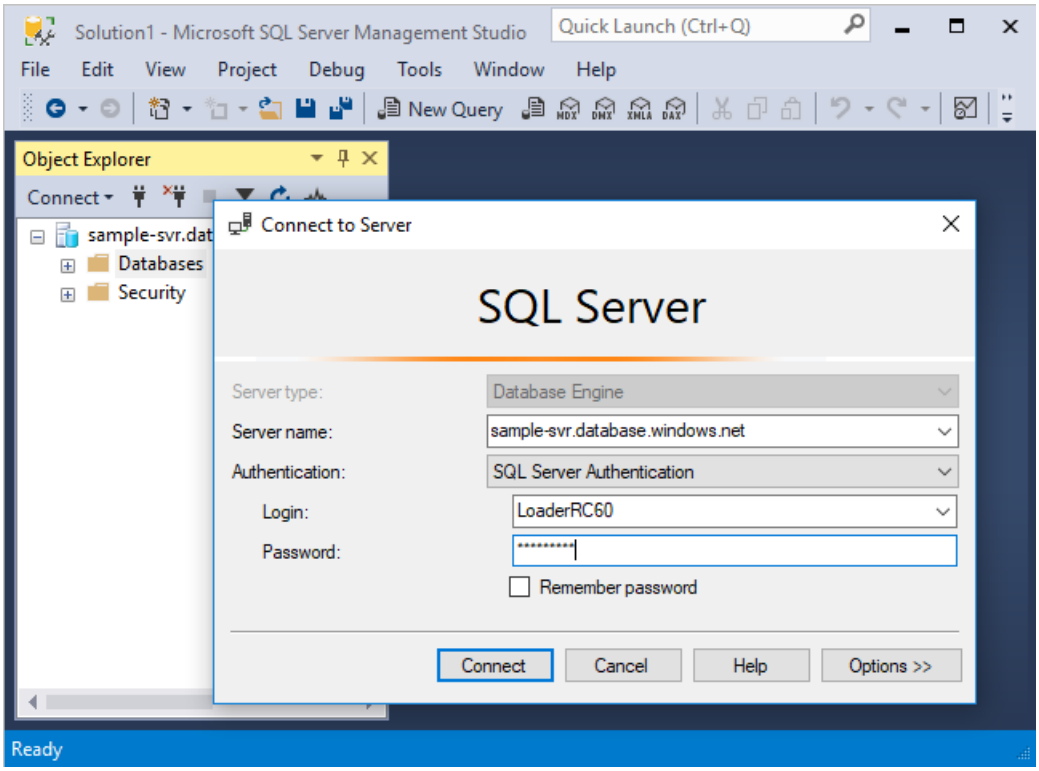
```
CREATE USER LoaderRC60 FOR LOGIN LoaderRC60;
GRANT CONTROL ON DATABASE::[SampleDW] to LoaderRC60;
EXEC sp_addrolemember 'staticrc60', 'LoaderRC60';
```

6. Click **Execute**.

Connect to the server as the loading user

The first step toward loading data is to login as LoaderRC60.

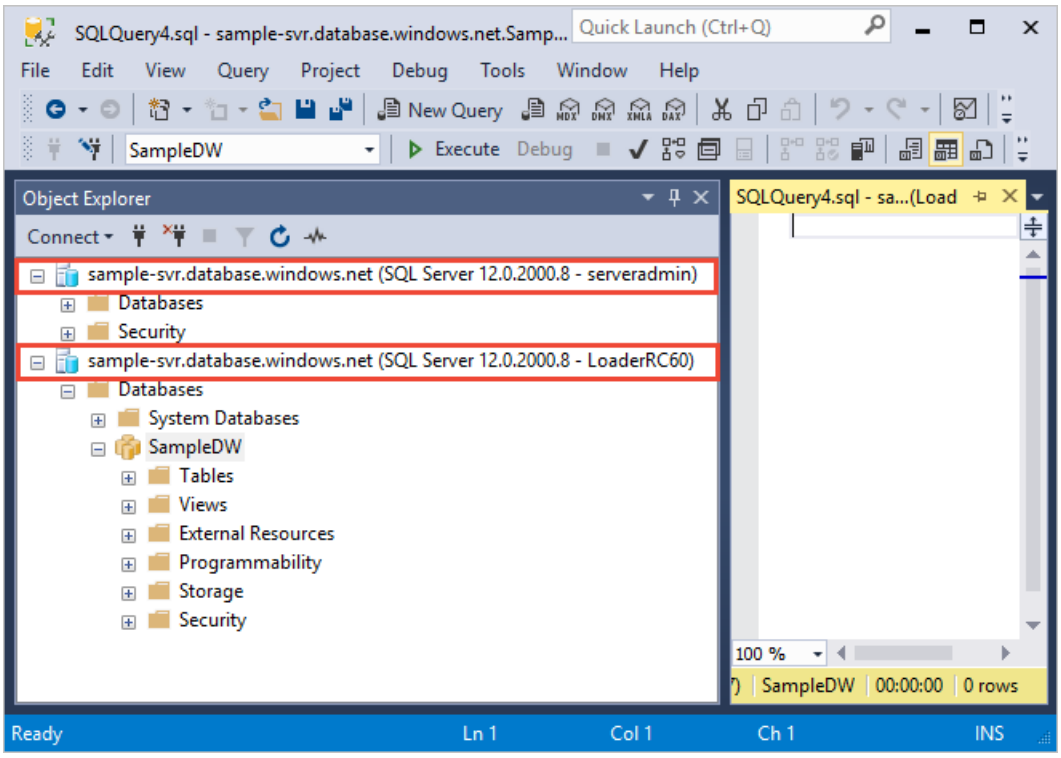
1. In Object Explorer, click the **Connect** drop down menu and select **Database Engine**. The **Connect to Server** dialog box appears.



2. Enter the fully qualified server name, and enter **LoaderRC60** as the Login. Enter your password for LoaderRC60.

3. Click **Connect**.

4. When your connection is ready, you will see two server connections in Object Explorer. One connection as ServerAdmin and one connection as LoaderRC60.

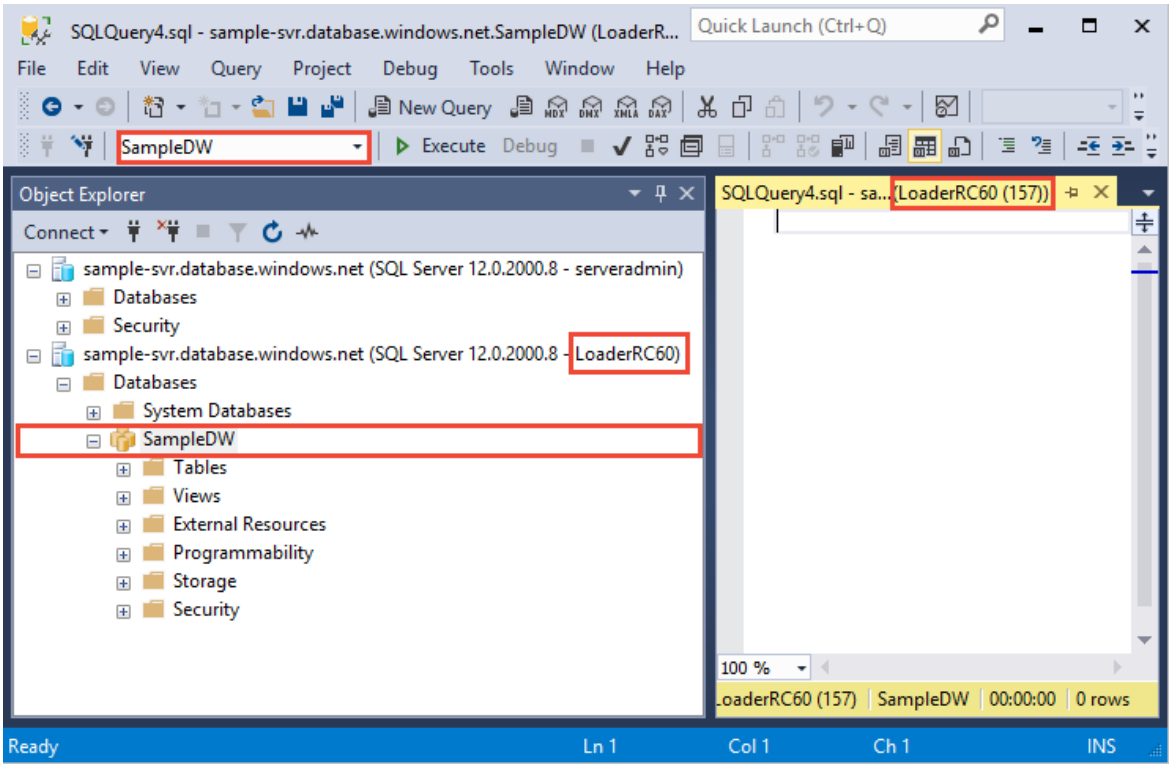


Create external tables and objects

You are ready to begin the process of loading data into your new data warehouse. For future reference, to learn how to get your data to Azure Blob storage or to load it directly from your source into SQL pool, see the loading overview.

Run the following SQL scripts to specify information about the data you wish to load. This information includes where the data is located, the format of the contents of the data, and the table definition for the data. The data is located in a global Azure Blob.

1. In the previous section, you logged into your data warehouse as LoaderRC60. In SSMS, right-click **SampleDW** under your LoaderRC60 connection and select **New Query**. A new query window appears.



2. Compare your query window to the previous image. Verify your new query window is running as

LoaderRC60 and performing queries on your SampleDW database. Use this query window to perform all of the loading steps.

3. Create a master key for the SampleDW database. You only need to create a master key once per database.

```
CREATE MASTER KEY;
```

4. Run the following CREATE EXTERNAL DATA SOURCE statement to define the location of the Azure blob. This is the location of the external worldwide importers data. To run a command that you have appended to the query window, highlight the commands you wish to run and click **Execute**.

```
CREATE EXTERNAL DATA SOURCE WWIStorage
WITH
(
    TYPE = Hadoop,
    LOCATION = 'wasbs://wideworldimporters@sqldwholdata.blob.core.windows.net'
);
```

5. Run the following CREATE EXTERNAL FILE FORMAT T-SQL statement to specify the formatting characteristics and options for the external data file. This statement specifies the external data is stored as text and the values are separated by the pipe ('|') character.

```
CREATE EXTERNAL FILE FORMAT TextFileFormat
WITH
(
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS
    (
        FIELD_TERMINATOR = '|',
        USE_TYPE_DEFAULT = FALSE
    )
);
```

6. Run the following CREATE SCHEMA statements to create a schema for your external file format. The ext schema provides a way to organize the external tables you are about to create. The wwi schema organizes the standard tables that will contain the data.

```
CREATE SCHEMA ext;
GO
CREATE SCHEMA wwi;
```

7. Create the external tables. The table definitions are stored in the database, but the tables reference data that is stored in Azure blob storage. Run the following T-SQL commands to create several external tables that all point to the Azure blob you defined previously in the external data source.

```
CREATE EXTERNAL TABLE [ext].[dimension_City](
    [City Key] [int] NOT NULL,
    [WWI City ID] [int] NOT NULL,
    [City] [nvarchar](50) NOT NULL,
    [State Province] [nvarchar](50) NOT NULL,
    [Country] [nvarchar](60) NOT NULL,
    [Continent] [nvarchar](30) NOT NULL,
    [Sales Territory] [nvarchar](50) NOT NULL,
    [Region] [nvarchar](30) NOT NULL,
    [Subregion] [nvarchar](30) NOT NULL,
    [Location] [nvarchar](76) NULL,
    [Latest Recorded Population] [bigint] NOT NULL,
    [Valid From] [datetime2](7) NOT NULL,
```

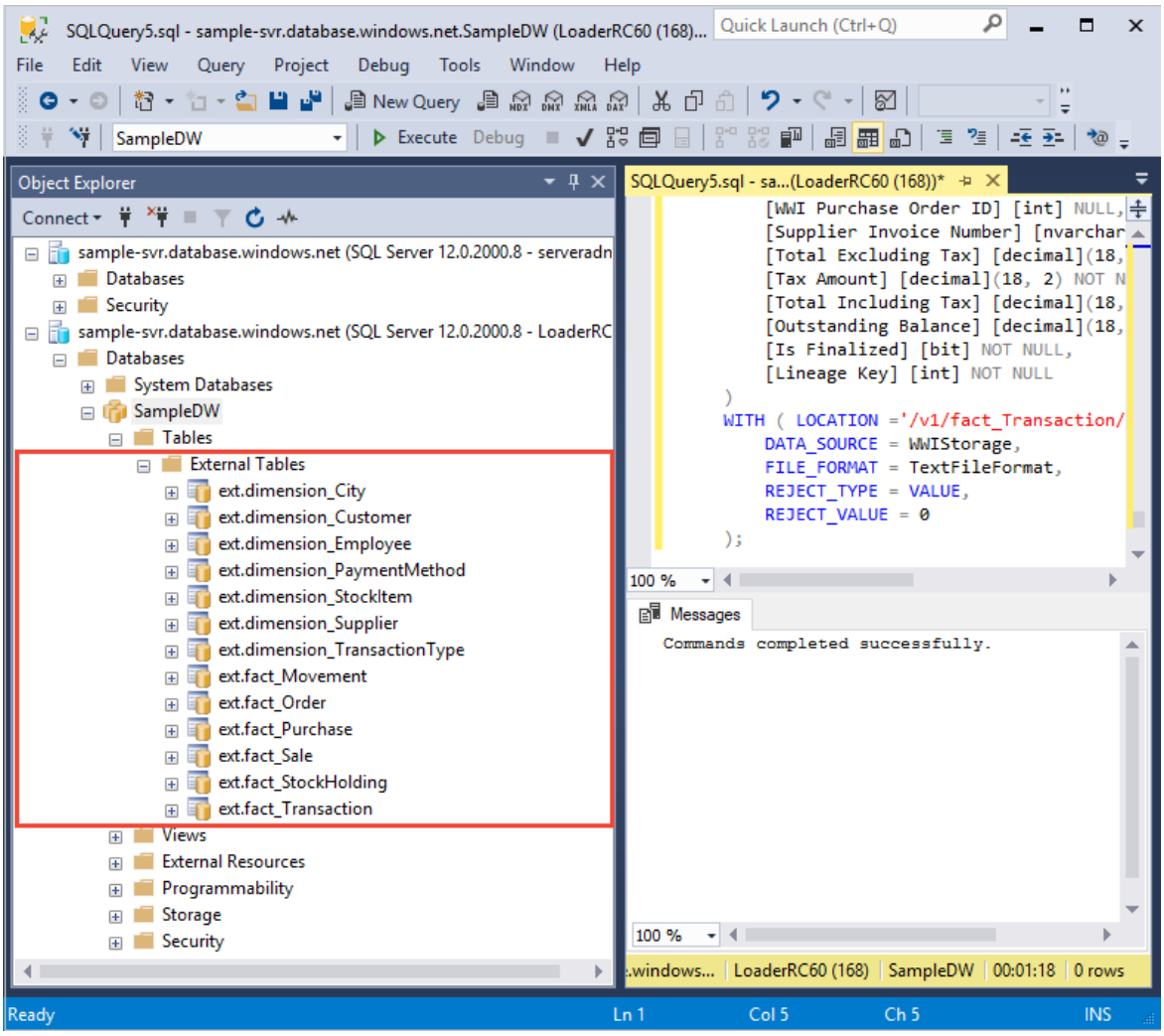
```
    [Valid To] [datetime2](7) NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH (LOCATION='/v1/dimension_City/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[dimension_Customer] (
    [Customer Key] [int] NOT NULL,
    [WWI Customer ID] [int] NOT NULL,
    [Customer] [nvarchar](100) NOT NULL,
    [Bill To Customer] [nvarchar](100) NOT NULL,
       [Category] [nvarchar](50) NOT NULL,
    [Buying Group] [nvarchar](50) NOT NULL,
    [Primary Contact] [nvarchar](50) NOT NULL,
    [Postal Code] [nvarchar](10) NOT NULL,
    [Valid From] [datetime2](7) NOT NULL,
    [Valid To] [datetime2](7) NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH (LOCATION='/v1/dimension_Customer/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[dimension_Employee] (
    [Employee Key] [int] NOT NULL,
    [WWI Employee ID] [int] NOT NULL,
    [Employee] [nvarchar](50) NOT NULL,
    [Preferred Name] [nvarchar](50) NOT NULL,
    [Is Salesperson] [bit] NOT NULL,
    [Photo] [varbinary](300) NULL,
    [Valid From] [datetime2](7) NOT NULL,
    [Valid To] [datetime2](7) NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH ( LOCATION='/v1/dimension_Employee/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[dimension_PaymentMethod] (
    [Payment Method Key] [int] NOT NULL,
    [WWI Payment Method ID] [int] NOT NULL,
    [Payment Method] [nvarchar](50) NOT NULL,
    [Valid From] [datetime2](7) NOT NULL,
    [Valid To] [datetime2](7) NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH ( LOCATION ='/v1/dimension_PaymentMethod/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[dimension_StockItem](
    [Stock Item Key] [int] NOT NULL,
    [WWI Stock Item ID] [int] NOT NULL,
    [Stock Item] [nvarchar](100) NOT NULL,
    [Color] [nvarchar](20) NOT NULL,
    [Selling Package] [nvarchar](50) NOT NULL,
    [Buying Package] [nvarchar](50) NOT NULL,
    [Brand] [nvarchar](50) NOT NULL,
    [Size] [nvarchar](20) NOT NULL,
    [Lead Time Days] [int] NOT NULL,
```

```
    [Lead Time Days] [int] NOT NULL,
    [Quantity Per Outer] [int] NOT NULL,
    [Is Chiller Stock] [bit] NOT NULL,
    [Barcode] [nvarchar](50) NULL,
    [Tax Rate] [decimal](18, 3) NOT NULL,
    [Unit Price] [decimal](18, 2) NOT NULL,
    [Recommended Retail Price] [decimal](18, 2) NULL,
    [Typical Weight Per Unit] [decimal](18, 3) NOT NULL,
    [Photo] [varbinary](300) NULL,
    [Valid From] [datetime2](7) NOT NULL,
    [Valid To] [datetime2](7) NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH ( LOCATION ='/v1/dimension_StockItem/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[dimension_Supplier](
    [Supplier Key] [int] NOT NULL,
    [WWI Supplier ID] [int] NOT NULL,
    [Supplier] [nvarchar](100) NOT NULL,
    [Category] [nvarchar](50) NOT NULL,
    [Primary Contact] [nvarchar](50) NOT NULL,
    [Supplier Reference] [nvarchar](20) NULL,
    [Payment Days] [int] NOT NULL,
    [Postal Code] [nvarchar](10) NOT NULL,
    [Valid From] [datetime2](7) NOT NULL,
    [Valid To] [datetime2](7) NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH ( LOCATION ='/v1/dimension_Supplier/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[dimension_TransactionType](
    [Transaction Type Key] [int] NOT NULL,
    [WWI Transaction Type ID] [int] NOT NULL,
    [Transaction Type] [nvarchar](50) NOT NULL,
    [Valid From] [datetime2](7) NOT NULL,
    [Valid To] [datetime2](7) NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH ( LOCATION ='/v1/dimension_TransactionType/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[fact_Movement] (
    [Movement Key] [bigint] NOT NULL,
    [Date Key] [date] NOT NULL,
    [Stock Item Key] [int] NOT NULL,
    [Customer Key] [int] NULL,
    [Supplier Key] [int] NULL,
    [Transaction Type Key] [int] NOT NULL,
    [WWI Stock Item Transaction ID] [int] NOT NULL,
    [WWI Invoice ID] [int] NULL,
    [WWI Purchase Order ID] [int] NULL,
    [Quantity] [int] NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH ( LOCATION ='/v1/fact_Movement/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
```

```
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[fact_Order] (
    [Order Key] [bigint] NOT NULL,
    [City Key] [int] NOT NULL,
    [Customer Key] [int] NOT NULL,
    [Stock Item Key] [int] NOT NULL,
    [Order Date Key] [date] NOT NULL,
    [Picked Date Key] [date] NULL,
    [Salesperson Key] [int] NOT NULL,
    [Picker Key] [int] NULL,
    [WWI Order ID] [int] NOT NULL,
    [WWI Backorder ID] [int] NULL,
    [Description] [nvarchar](100) NOT NULL,
    [Package] [nvarchar](50) NOT NULL,
    [Quantity] [int] NOT NULL,
    [Unit Price] [decimal](18, 2) NOT NULL,
    [Tax Rate] [decimal](18, 3) NOT NULL,
    [Total Excluding Tax] [decimal](18, 2) NOT NULL,
    [Tax Amount] [decimal](18, 2) NOT NULL,
    [Total Including Tax] [decimal](18, 2) NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH ( LOCATION ='/v1/fact_Order/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[fact_Purchase] (
    [Purchase Key] [bigint] NOT NULL,
    [Date Key] [date] NOT NULL,
    [Supplier Key] [int] NOT NULL,
    [Stock Item Key] [int] NOT NULL,
    [WWI Purchase Order ID] [int] NULL,
    [Ordered Outers] [int] NOT NULL,
    [Ordered Quantity] [int] NOT NULL,
    [Received Outers] [int] NOT NULL,
    [Package] [nvarchar](50) NOT NULL,
    [Is Order Finalized] [bit] NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH ( LOCATION ='/v1/fact_Purchase/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[fact_Sale] (
    [Sale Key] [bigint] NOT NULL,
    [City Key] [int] NOT NULL,
    [Customer Key] [int] NOT NULL,
    [Bill To Customer Key] [int] NOT NULL,
    [Stock Item Key] [int] NOT NULL,
    [Invoice Date Key] [date] NOT NULL,
    [Delivery Date Key] [date] NULL,
    [Salesperson Key] [int] NOT NULL,
    [WWI Invoice ID] [int] NOT NULL,
    [Description] [nvarchar](100) NOT NULL,
    [Package] [nvarchar](50) NOT NULL,
    [Quantity] [int] NOT NULL,
    [Unit Price] [decimal](18, 2) NOT NULL,
    [Tax Rate] [decimal](18, 3) NOT NULL,
    [Total Excluding Tax] [decimal](18, 2) NOT NULL,
    [Tax Amount] [decimal](18, 2) NOT NULL,
    [Profit] [decimal](18, 2) NOT NULL,
    [Total Including Tax] [decimal](18, 2) NOT NULL,
    [Total Dry Items] [int] NOT NULL,
    [Total Chiller Items] [int] NOT NULL,
    [
```

```
    [Lineage Key] [int] NOT NULL
)
WITH ( LOCATION ='/v1/fact_Sale/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[fact_StockHolding] (
    [Stock Holding Key] [bigint] NOT NULL,
    [Stock Item Key] [int] NOT NULL,
    [Quantity On Hand] [int] NOT NULL,
    [Bin Location] [nvarchar](20) NOT NULL,
    [Last Stocktake Quantity] [int] NOT NULL,
    [Last Cost Price] [decimal](18, 2) NOT NULL,
    [Reorder Level] [int] NOT NULL,
    [Target Stock Level] [int] NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH ( LOCATION ='/v1/fact_StockHolding/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[fact_Transaction] (
    [Transaction Key] [bigint] NOT NULL,
    [Date Key] [date] NOT NULL,
    [Customer Key] [int] NULL,
    [Bill To Customer Key] [int] NULL,
    [Supplier Key] [int] NULL,
    [Transaction Type Key] [int] NOT NULL,
    [Payment Method Key] [int] NULL,
    [WWI Customer Transaction ID] [int] NULL,
    [WWI Supplier Transaction ID] [int] NULL,
    [WWI Invoice ID] [int] NULL,
    [WWI Purchase Order ID] [int] NULL,
    [Supplier Invoice Number] [nvarchar](20) NULL,
    [Total Excluding Tax] [decimal](18, 2) NOT NULL,
    [Tax Amount] [decimal](18, 2) NOT NULL,
    [Total Including Tax] [decimal](18, 2) NOT NULL,
    [Outstanding Balance] [decimal](18, 2) NOT NULL,
    [Is Finalized] [bit] NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH ( LOCATION ='/v1/fact_Transaction/',
    DATA_SOURCE = WWIStorage,
    FILE_FORMAT = TextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
```

8. In Object Explorer, expand SampleDW to see the list of external tables you created.



Load the data into SQL pool

This section uses the external tables you defined to load the sample data from Azure Blob to SQL pool.

> **NOTE**
>
> This tutorial loads the data directly into the final table. In a production environment, you will usually use CREATE TABLE AS SELECT to load into a staging table. While data is in the staging table you can perform any necessary transformations. To append the data in the staging table to a production table, you can use the INSERT...SELECT statement. For more information, see Inserting data into a production table.

The script uses the CREATE TABLE AS SELECT (CTAS) T-SQL statement to load the data from Azure Storage Blob into new tables in your data warehouse. CTAS creates a new table based on the results of a select statement. The new table has the same columns and data types as the results of the select statement. When the select statement selects from an external table, the data is imported into a relational table in the data warehouse.

This script does not load data into the wwi.dimension_Date and wwi.fact_Sale tables. These tables are generated in a later step in order to make the tables have a sizeable number of rows.

1. Run the following script to load the data into new tables in your data warehouse.

```
CREATE TABLE [wwi].[dimension_City]
WITH
(
    DISTRIBUTION = REPLICATE,
    CLUSTERED COLUMNSTORE INDEX
)
AS
```

```
AS
SELECT * FROM [ext].[dimension_City]
OPTION (LABEL = 'CTAS : Load [wwi].[dimension_City]')
;

CREATE TABLE [wwi].[dimension_Customer]
WITH
(
    DISTRIBUTION = REPLICATE,
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[dimension_Customer]
OPTION (LABEL = 'CTAS : Load [wwi].[dimension_Customer]')
;

CREATE TABLE [wwi].[dimension_Employee]
WITH
(
    DISTRIBUTION = REPLICATE,
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[dimension_Employee]
OPTION (LABEL = 'CTAS : Load [wwi].[dimension_Employee]')
;

CREATE TABLE [wwi].[dimension_PaymentMethod]
WITH
(
    DISTRIBUTION = REPLICATE,
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[dimension_PaymentMethod]
OPTION (LABEL = 'CTAS : Load [wwi].[dimension_PaymentMethod]')
;

CREATE TABLE [wwi].[dimension_StockItem]
WITH
(
    DISTRIBUTION = REPLICATE,
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[dimension_StockItem]
OPTION (LABEL = 'CTAS : Load [wwi].[dimension_StockItem]')
;

CREATE TABLE [wwi].[dimension_Supplier]
WITH
(
    DISTRIBUTION = REPLICATE,
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[dimension_Supplier]
OPTION (LABEL = 'CTAS : Load [wwi].[dimension_Supplier]')
;

CREATE TABLE [wwi].[dimension_TransactionType]
WITH
(
    DISTRIBUTION = REPLICATE,
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[dimension_TransactionType]
OPTION (LABEL = 'CTAS : Load [wwi].[dimension_TransactionType]')
;
```

```
;

CREATE TABLE [wwi].[fact_Movement]
WITH
(
    DISTRIBUTION = HASH([Movement Key]),
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[fact_Movement]
OPTION (LABEL = 'CTAS : Load [wwi].[fact_Movement]')
;

CREATE TABLE [wwi].[fact_Order]
WITH
(
    DISTRIBUTION = HASH([Order Key]),
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[fact_Order]
OPTION (LABEL = 'CTAS : Load [wwi].[fact_Order]')
;

CREATE TABLE [wwi].[fact_Purchase]
WITH
(
    DISTRIBUTION = HASH([Purchase Key]),
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[fact_Purchase]
OPTION (LABEL = 'CTAS : Load [wwi].[fact_Purchase]')
;

CREATE TABLE [wwi].[seed_Sale]
WITH
(
    DISTRIBUTION = HASH([WWI Invoice ID]),
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[fact_Sale]
OPTION (LABEL = 'CTAS : Load [wwi].[seed_Sale]')
;

CREATE TABLE [wwi].[fact_StockHolding]
WITH
(
    DISTRIBUTION = HASH([Stock Holding Key]),
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[fact_StockHolding]
OPTION (LABEL = 'CTAS : Load [wwi].[fact_StockHolding]')
;

CREATE TABLE [wwi].[fact_Transaction]
WITH
(
    DISTRIBUTION = HASH([Transaction Key]),
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[fact_Transaction]
OPTION (LABEL = 'CTAS : Load [wwi].[fact_Transaction]')
;
```

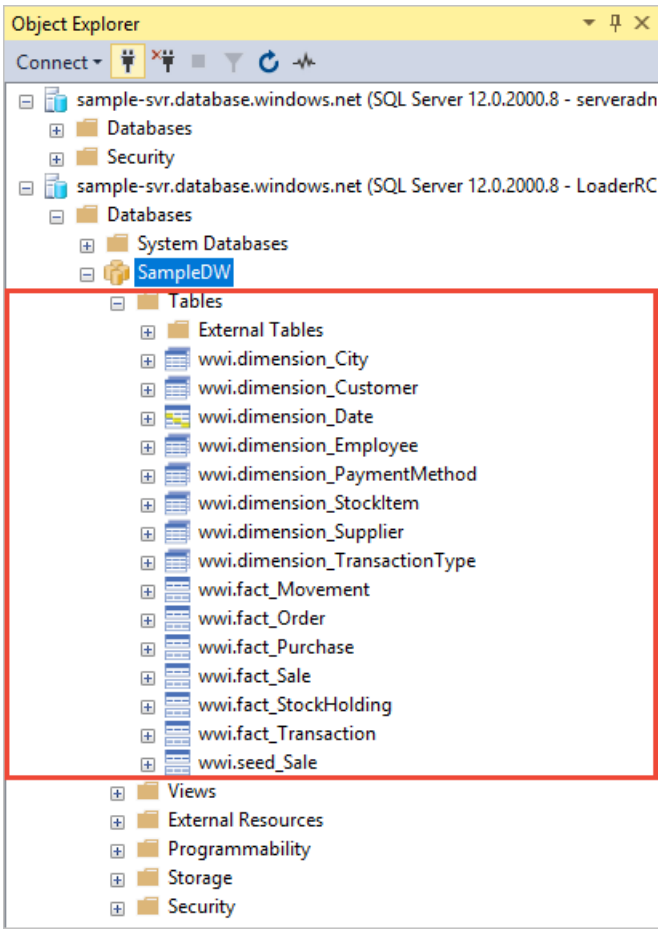
2. View your data as it loads. You're loading several GBs of data and compressing it into highly performant clustered columnstore indexes. Open a new query window on SampleDW, and run the following query to show the status of the load. After starting the query, grab a coffee and a snack while SQL pool does some heavy lifting.

```
SELECT
    r.command,
    s.request_id,
    r.status,
    count(distinct input_name) as nbr_files,
    sum(s.bytes_processed)/1024/1024/1024 as gb_processed
FROM
    sys.dm_pdw_exec_requests r
    INNER JOIN sys.dm_pdw_dms_external_work s
    ON r.request_id = s.request_id
WHERE
    r.[label] = 'CTAS : Load [wwi].[dimension_City]' OR
    r.[label] = 'CTAS : Load [wwi].[dimension_Customer]' OR
    r.[label] = 'CTAS : Load [wwi].[dimension_Employee]' OR
    r.[label] = 'CTAS : Load [wwi].[dimension_PaymentMethod]' OR
    r.[label] = 'CTAS : Load [wwi].[dimension_StockItem]' OR
    r.[label] = 'CTAS : Load [wwi].[dimension_Supplier]' OR
    r.[label] = 'CTAS : Load [wwi].[dimension_TransactionType]' OR
    r.[label] = 'CTAS : Load [wwi].[fact_Movement]' OR
    r.[label] = 'CTAS : Load [wwi].[fact_Order]' OR
    r.[label] = 'CTAS : Load [wwi].[fact_Purchase]' OR
    r.[label] = 'CTAS : Load [wwi].[fact_StockHolding]' OR
    r.[label] = 'CTAS : Load [wwi].[fact_Transaction]'
GROUP BY
    r.command,
    s.request_id,
    r.status
ORDER BY
    nbr_files desc,
    gb_processed desc;
```

3. View all system queries.

```
SELECT * FROM sys.dm_pdw_exec_requests;
```

4. Enjoy seeing your data nicely loaded into your data warehouse.



Create tables and procedures to generate the Date and Sales tables

This section creates the wwi.dimension_Date and wwi.fact_Sale tables. It also creates stored procedures that can generate millions of rows in the wwi.dimension_Date and wwi.fact_Sale tables.

1. Create the dimension_Date and fact_Sale tables.

```
CREATE TABLE [wwi].[dimension_Date]
(
    [Date] [datetime] NOT NULL,
    [Day Number] [int] NOT NULL,
    [Day] [nvarchar](10) NOT NULL,
    [Month] [nvarchar](10) NOT NULL,
    [Short Month] [nvarchar](3) NOT NULL,
    [Calendar Month Number] [int] NOT NULL,
    [Calendar Month Label] [nvarchar](20) NOT NULL,
    [Calendar Year] [int] NOT NULL,
    [Calendar Year Label] [nvarchar](10) NOT NULL,
    [Fiscal Month Number] [int] NOT NULL,
    [Fiscal Month Label] [nvarchar](20) NOT NULL,
    [Fiscal Year] [int] NOT NULL,
    [Fiscal Year Label] [nvarchar](10) NOT NULL,
    [ISO Week Number] [int] NOT NULL
)
WITH
(
    DISTRIBUTION = REPLICATE,
    CLUSTERED INDEX ([Date])
);
CREATE TABLE [wwi].[fact_Sale]
(
    [Sale Key] [bigint] IDENTITY(1,1) NOT NULL,
    [City Key] [int] NOT NULL,
    [Customer Key] [int] NOT NULL,
    [Bill To Customer Key] [int] NOT NULL,
    [Stock Item Key] [int] NOT NULL,
    [Invoice Date Key] [date] NOT NULL,
    [Delivery Date Key] [date] NULL,
    [Salesperson Key] [int] NOT NULL,
    [WWI Invoice ID] [int] NOT NULL,
    [Description] [nvarchar](100) NOT NULL,
    [Package] [nvarchar](50) NOT NULL,
    [Quantity] [int] NOT NULL,
    [Unit Price] [decimal](18, 2) NOT NULL,
    [Tax Rate] [decimal](18, 3) NOT NULL,
    [Total Excluding Tax] [decimal](18, 2) NOT NULL,
    [Tax Amount] [decimal](18, 2) NOT NULL,
    [Profit] [decimal](18, 2) NOT NULL,
    [Total Including Tax] [decimal](18, 2) NOT NULL,
    [Total Dry Items] [int] NOT NULL,
    [Total Chiller Items] [int] NOT NULL,
    [Lineage Key] [int] NOT NULL
)
WITH
(
    DISTRIBUTION = HASH ( [WWI Invoice ID] ),
    CLUSTERED COLUMNSTORE INDEX
)
```

2. Create [wwi].[InitialSalesDataPopulation] to increase the number of rows in [wwi].[seed_Sale] by a factor of eight.

```
CREATE PROCEDURE [wwi].[InitialSalesDataPopulation] AS
BEGIN
    INSERT INTO [wwi].[seed_Sale] (
        [Sale Key], [City Key], [Customer Key], [Bill To Customer Key], [Stock Item Key], [Invoice
Date Key], [Delivery Date Key], [Salesperson Key], [WWI Invoice ID], [Description], [Package],
[Quantity], [Unit Price], [Tax Rate], [Total Excluding Tax], [Tax Amount], [Profit], [Total Including
Tax], [Total Dry Items], [Total Chiller Items], [Lineage Key]
    )
    SELECT
        [Sale Key], [City Key], [Customer Key], [Bill To Customer Key], [Stock Item Key], [Invoice
Date Key], [Delivery Date Key], [Salesperson Key], [WWI Invoice ID], [Description], [Package],
[Quantity], [Unit Price], [Tax Rate], [Total Excluding Tax], [Tax Amount], [Profit], [Total Including
Tax], [Total Dry Items], [Total Chiller Items], [Lineage Key]
    FROM [wwi].[seed_Sale]

    INSERT INTO [wwi].[seed_Sale] (
        [Sale Key], [City Key], [Customer Key], [Bill To Customer Key], [Stock Item Key], [Invoice
Date Key], [Delivery Date Key], [Salesperson Key], [WWI Invoice ID], [Description], [Package],
[Quantity], [Unit Price], [Tax Rate], [Total Excluding Tax], [Tax Amount], [Profit], [Total Including
Tax], [Total Dry Items], [Total Chiller Items], [Lineage Key]
    )
    SELECT
        [Sale Key], [City Key], [Customer Key], [Bill To Customer Key], [Stock Item Key], [Invoice
Date Key], [Delivery Date Key], [Salesperson Key], [WWI Invoice ID], [Description], [Package],
[Quantity], [Unit Price], [Tax Rate], [Total Excluding Tax], [Tax Amount], [Profit], [Total Including
Tax], [Total Dry Items], [Total Chiller Items], [Lineage Key]
    FROM [wwi].[seed_Sale]

    INSERT INTO [wwi].[seed_Sale] (
        [Sale Key], [City Key], [Customer Key], [Bill To Customer Key], [Stock Item Key], [Invoice
Date Key], [Delivery Date Key], [Salesperson Key], [WWI Invoice ID], [Description], [Package],
[Quantity], [Unit Price], [Tax Rate], [Total Excluding Tax], [Tax Amount], [Profit], [Total Including
Tax], [Total Dry Items], [Total Chiller Items], [Lineage Key]
    )
    SELECT
        [Sale Key], [City Key], [Customer Key], [Bill To Customer Key], [Stock Item Key], [Invoice
Date Key], [Delivery Date Key], [Salesperson Key], [WWI Invoice ID], [Description], [Package],
[Quantity], [Unit Price], [Tax Rate], [Total Excluding Tax], [Tax Amount], [Profit], [Total Including
Tax], [Total Dry Items], [Total Chiller Items], [Lineage Key]
    FROM [wwi].[seed_Sale]
END
```

3. Create this stored procedure that populates rows into wwi.dimension_Date.

```
CREATE PROCEDURE [wwi].[PopulateDateDimensionForYear] @Year [int] AS
BEGIN
    IF OBJECT_ID('tempdb..#month', 'U') IS NOT NULL
        DROP TABLE #month
    CREATE TABLE #month (
        monthnum int,
        numofdays int
    )
    WITH ( DISTRIBUTION = ROUND_ROBIN, heap )
    INSERT INTO #month
        SELECT 1, 31 UNION SELECT 2, CASE WHEN (@YEAR % 4 = 0 AND @YEAR % 100 <> 0) OR @YEAR % 400 =
0 THEN 29 ELSE 28 END UNION SELECT 3,31 UNION SELECT 4,30 UNION SELECT 5,31 UNION SELECT 6,30 UNION
SELECT 7,31 UNION SELECT 8,31 UNION SELECT 9,30 UNION SELECT 10,31 UNION SELECT 11,30 UNION SELECT
12,31

    IF OBJECT_ID('tempdb..#days', 'U') IS NOT NULL
        DROP TABLE #days
    CREATE TABLE #days (days int)
    WITH (DISTRIBUTION = ROUND_ROBIN, HEAP)

    INSERT INTO #days
        SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5 UNION SELECT 6 UNION
```

```
SELECT 7 UNION SELECT 8 UNION SELECT 9 UNION SELECT 10 UNION SELECT 11 UNION SELECT 12 UNION SELECT
13 UNION SELECT 14 UNION SELECT 15 UNION SELECT 16 UNION SELECT 17 UNION SELECT 18 UNION SELECT 19
UNION SELECT 20    UNION SELECT 21 UNION SELECT 22 UNION SELECT 23 UNION SELECT 24 UNION SELECT 25
UNION SELECT 26 UNION SELECT 27 UNION SELECT 28 UNION SELECT 29 UNION SELECT 30 UNION SELECT 31

    INSERT [wwi].[dimension_Date] (
        [Date], [Day Number], [Day], [Month], [Short Month], [Calendar Month Number], [Calendar Month
Label], [Calendar Year], [Calendar Year Label], [Fiscal Month Number], [Fiscal Month Label], [Fiscal
Year], [Fiscal Year Label], [ISO Week Number]
    )
    SELECT
        CAST(CAST(monthnum AS VARCHAR(2)) + '/' + CAST([days] AS VARCHAR(3)) + '/' + CAST(@year AS
CHAR(4)) AS DATE) AS [Date]
        ,DAY(CAST(CAST(monthnum AS VARCHAR(2)) + '/' + CAST([days] AS VARCHAR(3)) + '/' + CAST(@year
AS CHAR(4)) AS DATE)) AS [Day Number]
        ,CAST(DATENAME(day, CAST(CAST(monthnum AS VARCHAR(2)) + '/' + CAST([days] AS VARCHAR(3)) +
'/' + CAST(@year AS CHAR(4)) AS DATE)) AS NVARCHAR(10)) AS [Day]
        ,CAST(DATENAME(month, CAST(CAST(monthnum AS VARCHAR(2)) + '/' + CAST([days] AS VARCHAR(3)) +
'/' + CAST(@year as char(4)) AS DATE)) AS nvarchar(10)) AS [Month]
        ,CAST(SUBSTRING(DATENAME(month, CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as
varchar(3)) + '/' + CAST(@year as char(4)) AS DATE)), 1, 3) AS nvarchar(3)) AS [Short Month]
        ,MONTH(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' +
CAST(@year as char(4)) AS DATE)) AS [Calendar Month Number]
        ,CAST(N'CY' + CAST(YEAR(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3))
+ '/' + CAST(@year as char(4)) AS DATE)) AS nvarchar(4)) + N'-' + SUBSTRING(DATENAME(month,
CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' + CAST(@year as char(4))
AS DATE)), 1, 3) AS nvarchar(10)) AS [Calendar Month Label]
        ,YEAR(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' + CAST(@year
as char(4)) AS DATE)) AS [Calendar Year]
        ,CAST(N'CY' + CAST(YEAR(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3))
+ '/' + CAST(@year as char(4)) AS DATE)) AS nvarchar(4)) AS nvarchar(10)) AS [Calendar Year Label]
        ,CASE WHEN MONTH(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' +
CAST(@year as char(4)) AS DATE)) IN (11, 12)
        THEN MONTH(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' +
CAST(@year as char(4)) AS DATE)) - 10
        ELSE MONTH(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' +
CAST(@year as char(4)) AS DATE)) + 2 END AS [Fiscal Month Number]
        ,CAST(N'FY' + CAST(CASE WHEN MONTH(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as
varchar(3)) + '/' + CAST(@year as char(4)) AS DATE)) IN (11, 12)
        THEN YEAR(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' +
CAST(@year as char(4)) AS DATE)) + 1
        ELSE YEAR(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' +
CAST(@year as char(4)) AS DATE)) END AS nvarchar(4)) + N'-' + SUBSTRING(DATENAME(month,
CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' + CAST(@year as char(4))
AS DATE)), 1, 3) AS nvarchar(20)) AS [Fiscal Month Label]
        ,CASE WHEN MONTH(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' +
CAST(@year as char(4)) AS DATE)) IN (11, 12)
        THEN YEAR(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' +
CAST(@year as char(4)) AS DATE)) + 1
        ELSE YEAR(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' +
CAST(@year as char(4)) AS DATE)) END AS [Fiscal Year]
        ,CAST(N'FY' + CAST(CASE WHEN MONTH(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as
varchar(3)) + '/' + CAST(@year as char(4)) AS DATE)) IN (11, 12)
        THEN YEAR(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' +
CAST(@year as char(4)) AS DATE)) + 1
        ELSE YEAR(CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) + '/' +
CAST(@year as char(4)) AS DATE))END AS nvarchar(4)) AS nvarchar(10)) AS [Fiscal Year Label]
        , DATEPART(ISO_WEEK, CAST(CAST(monthnum as varchar(2)) + '/' + CAST([days] as varchar(3)) +
'/' + CAST(@year as char(4)) AS DATE)) AS [ISO Week Number]
FROM #month m
    CROSS JOIN #days d
WHERE d.days <= m.numofdays

DROP table #month;
DROP table #days;
END;
```

4. Create this procedure that populates the wwi.dimension_Date and wwi.fact_Sale tables. It calls [wwi].[PopulateDateDimensionForYear] to populate wwi.dimension_Date.

```
CREATE PROCEDURE [wwi].[Configuration_PopulateLargeSaleTable] @EstimatedRowsPerDay [bigint],@Year
[int] AS
BEGIN
    SET NOCOUNT ON;
    SET XACT_ABORT ON;

    EXEC [wwi].[PopulateDateDimensionForYear] @Year;

    DECLARE @OrderCounter bigint = 0;
    DECLARE @NumberOfSalesPerDay bigint = @EstimatedRowsPerDay;
    DECLARE @DateCounter date;
    DECLARE @StartingSaleKey bigint;
    DECLARE @MaximumSaleKey bigint = (SELECT MAX([Sale Key]) FROM wwi.seed_Sale);
    DECLARE @MaxDate date;
    SET @MaxDate = (SELECT MAX([Invoice Date Key]) FROM wwi.fact_Sale)
    IF ( @MaxDate < CAST(@YEAR AS CHAR(4)) + '1231') AND (@MaxDate > CAST(@YEAR AS CHAR(4)) + '0101')
        SET @DateCounter = @MaxDate
    ELSE
        SET @DateCounter= CAST(@Year as char(4)) + '0101';

    PRINT 'Targeting ' + CAST(@NumberOfSalesPerDay AS varchar(20)) + ' sales per day.';

    DECLARE @OutputCounter varchar(20);
    DECLARE @variance DECIMAL(18,10);
    DECLARE @VariantNumberOfSalesPerDay BIGINT;

    WHILE @DateCounter < CAST(@YEAR AS CHAR(4)) + '1231'
    BEGIN
        SET @OutputCounter = CONVERT(varchar(20), @DateCounter, 112);
        RAISERROR(@OutputCounter, 0, 1);
        SET @variance = (SELECT RAND() * 10)*.01 + .95
        SET @VariantNumberOfSalesPerDay = FLOOR(@NumberOfSalesPerDay * @variance)

        SET @StartingSaleKey = @MaximumSaleKey - @VariantNumberOfSalesPerDay - FLOOR(RAND() * 20000);
        SET @OrderCounter = 0;

        INSERT [wwi].[fact_Sale] (
            [City Key], [Customer Key], [Bill To Customer Key], [Stock Item Key], [Invoice Date Key],
[Delivery Date Key], [Salesperson Key], [WWI Invoice ID], [Description], Package, Quantity, [Unit
Price], [Tax Rate], [Total Excluding Tax], [Tax Amount], Profit, [Total Including Tax], [Total Dry
Items], [Total Chiller Items], [Lineage Key]
        )
        SELECT TOP(@VariantNumberOfSalesPerDay)
            [City Key], [Customer Key], [Bill To Customer Key], [Stock Item Key], @DateCounter,
DATEADD(day, 1, @DateCounter), [Salesperson Key], [WWI Invoice ID], [Description], Package, Quantity,
[Unit Price], [Tax Rate], [Total Excluding Tax], [Tax Amount], Profit, [Total Including Tax], [Total
Dry Items], [Total Chiller Items], [Lineage Key]
        FROM [wwi].[seed_Sale]
        WHERE
             --[Sale Key] > @StartingSaleKey and /* IDENTITY DOES NOT WORK THE SAME IN SQLDW AND
CAN'T USE THIS METHOD FOR VARIANT */
            [Invoice Date Key] >=cast(@YEAR AS CHAR(4)) + '-01-01'
        ORDER BY [Sale Key];

        SET @DateCounter = DATEADD(day, 1, @DateCounter);
    END;

END;
```

Generate millions of rows

Use the stored procedures you created to generate millions of rows in the wwi.fact_Sale table, and

corresponding data in the wwi.dimension_Date table.

1. Run this procedure to seed the [wwi].[seed_Sale] with more rows.

```
EXEC [wwi].[InitialSalesDataPopulation]
```

2. Run this procedure to populate wwi.fact_Sale with 100,000 rows per day for each day in the year 2000.

```
EXEC [wwi].[Configuration_PopulateLargeSaleTable] 100000, 2000
```

3. The data generation in the previous step might take a while as it progresses through the year. To see which day the current process is on, open a new query and run this SQL command:

```
SELECT MAX([Invoice Date Key]) FROM wwi.fact_Sale;
```

4. Run the following command to see the space used.

```
EXEC sp_spaceused N'wwi.fact_Sale';
```

Populate the replicated table cache

SQL pool replicates a table by caching the data to each Compute node. The cache gets populated when a query runs against the table. Therefore, the first query on a replicated table might require extra time to populate the cache. After the cache is populated, queries on replicated tables run faster.

Run these SQL queries to populate the replicated table cache on the Compute nodes.

```
SELECT TOP 1 * FROM [wwi].[dimension_City];
SELECT TOP 1 * FROM [wwi].[dimension_Customer];
SELECT TOP 1 * FROM [wwi].[dimension_Date];
SELECT TOP 1 * FROM [wwi].[dimension_Employee];
SELECT TOP 1 * FROM [wwi].[dimension_PaymentMethod];
SELECT TOP 1 * FROM [wwi].[dimension_StockItem];
SELECT TOP 1 * FROM [wwi].[dimension_Supplier];
SELECT TOP 1 * FROM [wwi].[dimension_TransactionType];
```

Create statistics on newly loaded data

To achieve high query performance, it's important to create statistics on each column of each table after the first load. It's also important to update statistics after substantial changes in the data.

1. Create this stored procedure that updates statistics on all columns of all tables.

```
CREATE PROCEDURE    [dbo].[prc_sqldw_create_stats]
(   @create_type    tinyint -- 1 default 2 Fullscan 3 Sample
,   @sample_pct     tinyint
)
AS

IF @create_type IS NULL
BEGIN
    SET @create_type = 1;
END;

IF @create_type NOT IN (1,2,3)
BEGIN
```

```
BEGIN
    THROW 151000,'Invalid value for @stats_type parameter. Valid range 1 (default), 2 (fullscan) or 3
(sample).',1;
END;

IF @sample_pct IS NULL
BEGIN;
    SET @sample_pct = 20;
END;

IF OBJECT_ID('tempdb..#stats_ddl') IS NOT NULL
BEGIN;
    DROP TABLE #stats_ddl;
END;

CREATE TABLE #stats_ddl
WITH    (   DISTRIBUTION    = HASH([seq_nmbr])
        ,   LOCATION        = USER_DB
        )
AS
WITH T
AS
(
SELECT      t.[name]                        AS [table_name]
,           s.[name]                        AS [table_schema_name]
,           c.[name]                        AS [column_name]
,           c.[column_id]                   AS [column_id]
,           t.[object_id]                   AS [object_id]
,           ROW_NUMBER()
            OVER(ORDER BY (SELECT NULL))    AS [seq_nmbr]
FROM        sys.[tables] t
JOIN        sys.[schemas] s         ON  t.[schema_id]       = s.[schema_id]
JOIN        sys.[columns] c         ON  t.[object_id]       = c.[object_id]
LEFT JOIN   sys.[stats_columns] l   ON  l.[object_id]       = c.[object_id]
                                    AND l.[column_id]       = c.[column_id]
                                    AND l.[stats_column_id] = 1
LEFT JOIN    sys.[external_tables] e    ON    e.[object_id]        = t.[object_id]
WHERE       l.[object_id] IS NULL
AND            e.[object_id] IS NULL -- not an external table
)
SELECT  [table_schema_name]
,       [table_name]
,       [column_name]
,       [column_id]
,       [object_id]
,       [seq_nmbr]
,       CASE @create_type
        WHEN 1
        THEN    CAST('CREATE STATISTICS '+QUOTENAME('stat_'+table_schema_name+ '_' + table_name +
'_'+column_name)+' ON
'+QUOTENAME(table_schema_name)+'.'+QUOTENAME(table_name)+'('+QUOTENAME(column_name)+')' AS
VARCHAR(8000))
        WHEN 2
        THEN    CAST('CREATE STATISTICS '+QUOTENAME('stat_'+table_schema_name+ '_' + table_name +
'_'+column_name)+' ON
'+QUOTENAME(table_schema_name)+'.'+QUOTENAME(table_name)+'('+QUOTENAME(column_name)+') WITH FULLSCAN'
AS VARCHAR(8000))
        WHEN 3
        THEN    CAST('CREATE STATISTICS '+QUOTENAME('stat_'+table_schema_name+ '_' + table_name +
'_'+column_name)+' ON
'+QUOTENAME(table_schema_name)+'.'+QUOTENAME(table_name)+'('+QUOTENAME(column_name)+') WITH SAMPLE
'+CONVERT(varchar(4),@sample_pct)+' PERCENT' AS VARCHAR(8000))
        END AS create_stat_ddl
FROM T
;

DECLARE @i INT              = 1
,       @t INT              = (SELECT COUNT(*) FROM #stats_ddl)
,       @s NVARCHAR(4000)   = N''
```

```
;

WHILE @i <= @t
BEGIN
    SET @s=(SELECT create_stat_ddl FROM #stats_ddl WHERE seq_nmbr = @i);
    PRINT @s
    EXEC sp_executesql @s
    SET @i+=1;
END

DROP TABLE #stats_ddl;
```

2. Run this command to create statistics on all columns of all tables in the data warehouse.

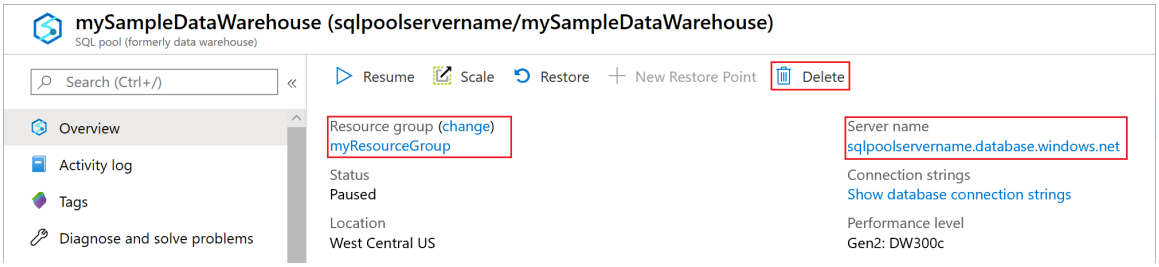
```
EXEC [dbo].[prc_sqldw_create_stats] 1, NULL;
```

Clean up resources

You are being charged for compute resources and data that you loaded into your data warehouse. These are billed separately.

Follow these steps to clean up resources as you desire.

1. Sign in to the Azure portal, click on your data warehouse.



2. If you want to keep the data in storage, you can pause compute when you aren't using the data warehouse. By pausing compute, you will only be charge for data storage and you can resume the compute whenever you are ready to work with the data. To pause compute, click the **Pause** button. When the data warehouse is paused, you will see a **Start** button. To resume compute, click **Start**.

3. If you want to remove future charges, you can delete the data warehouse. To remove the data warehouse so you won't be charged for compute or storage, click **Delete**.

4. To remove the server you created, click **sample-svr.database.windows.net** in the previous image, and then click **Delete**. Be careful with this as deleting the server will delete all databases assigned to the server.

5. To remove the resource group, click **SampleRG**, and then click **Delete resource group**.

Next steps

In this tutorial, you learned how to create a data warehouse and create a user for loading data. You created external tables to define the structure for data stored in Azure Storage Blob, and then used the PolyBase CREATE TABLE AS SELECT statement to load data into your data warehouse.

You did these things:

- Created a data warehouse using SQL pool in the Azure portal
- Set up a server-level firewall rule in the Azure portal
- Connected to the SQL pool with SSMS

- Created a user designated for loading data
- Created external tables for data in Azure Storage Blob
- Used the CTAS T-SQL statement to load data into your data warehouse
- Viewed the progress of data as it is loading
- Created statistics on the newly loaded data

Advance to the development overview to learn how to migrate an existing database to Azure Synapse SQL pool.

Design decisions to migrate an existing database to SQL pool

Ingest data into a dedicated SQL pool

8/19/2022 • 3 minutes to read • Edit Online

In this article, you'll learn how to ingest data from an Azure Data Lake Gen 2 storage account into a dedicated SQL pool in Azure Synapse Analytics.

Prerequisites

- **Azure subscription**: If you don't have an Azure subscription, create a free Azure account before you begin.
- **Azure storage account**: You use Azure Data Lake Storage Gen 2 as a *source* data store. If you don't have a storage account, see Create an Azure Storage account for steps to create one.
- **Azure Synapse Analytics**: You use a dedicated SQL pool as a *sink* data store.
  - If you don't have an existing Synapse workspace, see Creating a Synapse workspace
  - If you don't have an existing dedicated SQL pool. see Create a dedicated SQL pool.

Create linked services

In Azure Synapse Analytics, a linked service is where you define your connection information to other services. In this section, you'll add an Azure Synapse Analytics and Azure Data Lake Storage Gen2 linked service.

1. Open the Azure Synapse Analytics UX and go to the **Manage** tab.
2. Under **External connections**, select **Linked services**.
3. To add a linked service, select **New**.
4. Select the Azure Data Lake Storage Gen2 tile from the list and select **Continue**.
5. Enter your authentication credentials. Account key, service principal, and managed identity are currently supported authentication types. Select test connection to verify your credentials are correct. Select **Create** when finished.
6. Repeat steps 3-5, but instead of Azure Data Lake Storage Gen2, select the Azure Synapse Analytics tile and enter in the corresponding connection credentials. For Azure Synapse Analytics, SQL authentication, managed identity, and service principal are currently supported.

Create pipeline

A pipeline contains the logical flow for an execution of a set of activities. In this section, you'll create a pipeline containing a copy activity that ingests data from ADLS Gen2 into a dedicated SQL pool.

1. Go to the **Integrate** tab. Select on the plus icon next to the pipelines header and select **Pipeline**.
2. Under **Move and Transform** in the activities pane, drag **Copy data** onto the pipeline canvas.
3. Select on the copy activity and go to the **Source** tab. Select **New** to create a new source dataset.
4. Select Azure Data Lake Storage gen2 as your data store and select continue.
5. Select DelimitedText as your format and select continue.
6. In the set properties pane, select the ADLS linked service you created. Specify the file path of your source data and specify whether the first row has a header. You can import the schema from the file store or a sample file. Select OK when finished.
7. Go to the **Sink** tab. Select **New** to create a new sink dataset.
8. Select Azure Synapse Analytics as your data store and select continue.
9. In the set properties pane, select the Azure Synapse Analytics linked service you created. If you're writing to an existing table, select it from the dropdown. Otherwise, check **Edit** and enter in your new table name.

Select OK when finished

10. If you're creating a table, enable **Auto create table** in the table option field.

Debug and publish pipeline

Once you've finished configuring your pipeline, you can execute a debug run before you publish your artifacts to verify everything is correct.

1. To debug the pipeline, select **Debug** on the toolbar. You see the status of the pipeline run in the **Output** tab at the bottom of the window.
2. Once the pipeline can run successfully, in the top toolbar, select **Publish All**. This action publishes entities (datasets, and pipelines) you created to the Synapse Analytics service.
3. Wait until you see the **Successfully published** message. To see notification messages, select the bell button on the top-right.

Trigger and monitor the pipeline

In this step, you manually trigger the pipeline published in the previous step.

1. Select **Add Trigger** on the toolbar, and then select **Trigger Now**. On the **Pipeline Run** page, select **Finish**.
2. Go to the **Monitor** tab located in the left sidebar. You see a pipeline run that is triggered by a manual trigger. You can use links in the **Actions** column to view activity details and to rerun the pipeline.
3. To see activity runs associated with the pipeline run, select the **View Activity Runs** link in the **Actions** column. In this example, there's only one activity, so you see only one entry in the list. For details about the copy operation, select the **Details** link (eyeglasses icon) in the **Actions** column. Select **Pipeline Runs** at the top to go back to the Pipeline Runs view. To refresh the view, select **Refresh**.
4. Verify your data is correctly written in the dedicated SQL pool.

Next steps

For more information on data integration for Azure Synapse Analytics, see the Ingesting data into Azure Data Lake Storage Gen2 article.

Ingest data into Azure Data Lake Storage Gen2

8/19/2022 • 3 minutes to read • Edit Online

In this article, you'll learn how to ingest data from one location to another in an Azure Data Lake Gen 2 (Azure Data Lake Gen 2) storage account using Azure Synapse Analytics.

Prerequisites

- **Azure subscription**: If you don't have an Azure subscription, create a free Azure account before you begin.
- **Azure Storage account**: You use Azure Data Lake Gen 2 as a *source* data store. If you don't have a storage account, see Create an Azure Storage account for steps to create one.

Create linked services

In Azure Synapse Analytics, a linked service is where you define your connection information to other services. In this section, you'll add Azure Synapse Analytics and Azure Data Lake Gen 2 as linked services.

1. Open the Azure Synapse Analytics UX and go to the **Manage** tab.
2. Under **External connections**, select **Linked services**.
3. To add a linked service, select **New**.
4. Select the Azure Data Lake Storage Gen2 tile from the list and select **Continue**.
5. Enter your authentication credentials. Account key, service principal, and managed identity are currently supported authentication types. Select test connection to verify your credentials are correct.
6. Select **Create** when finished.

Create pipeline

A pipeline contains the logical flow for an execution of a set of activities. In this section, you'll create a pipeline containing a copy activity that ingests data from Azure Data Lake Gen 2 into a dedicated SQL pool.

1. Go to the **Orchestrate** tab. Select on the plus icon next to the pipelines header and select **Pipeline**.
2. Under **Move and Transform** in the activities pane, drag **Copy data** onto the pipeline canvas.
3. Select on the copy activity and go to the **Source** tab. Select **New** to create a new source dataset.
4. Select Azure Data Lake Storage Gen2 as your data store and select continue.
5. Select DelimitedText as your format and select continue.
6. In the set properties pane, select the ADLS linked service you created. Specify the file path of your source data and specify whether the first row has a header. You can import the schema from the file store or a sample file. Select OK when finished.
7. Go to the **Sink** tab. Select **New** to create a new sink dataset.
8. Select Azure Data Lake Storage gen2 as your data store and select continue.
9. Select DelimitedText as your format and select continue.
10. In the set properties pane, select the ADLS linked service you created. Specify the path of the folder where you wish to write data. Select OK when finished.

Debug and publish pipeline

Once you've finished configuring your pipeline, you can execute a debug run before you publish your artifacts to verify everything is correct.

1. To debug the pipeline, select **Debug** on the toolbar. You see the status of the pipeline run in the **Output** tab at the bottom of the window.
2. Once the pipeline can run successfully, in the top toolbar, select **Publish All**. This action publishes entities (datasets, and pipelines) you created to the Synapse Analytics service.
3. Wait until you see the **Successfully published** message. To see notification messages, select the bell button on the top-right.

Trigger and monitor the pipeline

In this step, you manually trigger the pipeline published in the previous step.

1. Select **Add Trigger** on the toolbar, and then select **Trigger Now**. On the **Pipeline Run** page, select **Finish**.
2. Go to the **Monitor** tab located in the left sidebar. You see a pipeline run that is triggered by a manual trigger. You can use links in the **Actions** column to view activity details and to rerun the pipeline.
3. To see activity runs associated with the pipeline run, select the **View Activity Runs** link in the **Actions** column. In this example, there's only one activity, so you see only one entry in the list. For details about the copy operation, select the **Details** link (eyeglasses icon) in the **Actions** column. Select **Pipeline Runs** at the top to go back to the Pipeline Runs view. To refresh the view, select **Refresh**.
4. Verify your data is correctly written in the dedicated SQL pool.

Next steps

For more information on data integration for Azure Synapse Analytics, see the Ingesting data into a dedicated SQL pool article.

Getting started with Visual Studio 2019

8/19/2022 • 2 minutes to read • Edit Online

Visual Studio **2019** SQL Server Data Tools (SSDT) is a single tool allowing you to do the following:

- Connect, query, and develop applications
- Leverage an object explorer to visually explore all objects in your data model including tables, views, stored procedures, and etc.
- Generate T-SQL data definition language (DDL) scripts for your objects
- Develop your data warehouse using a state-based approach with SSDT Database Projects
- Integrate your database project with source control systems such as Git with Azure Repos
- Set up continuous integration and deployment pipelines with automation servers such as Azure DevOps

Install Visual Studio 2019

See Download Visual Studio 2019 to download and install Visual Studio **16.3 and above**. During install, select the data storage and processing workload. Standalone SSDT installation is no longer required in Visual Studio 2019.

Unsupported features in SSDT

There are times when feature releases for Synapse SQL may not include support for SSDT. The following features are currently unsupported:

- Workload management - workload groups and classifiers
- Row-level security (including table-valued functions)
  - Submit a support ticket or vote to get the feature supported.
  - Submit a support ticket or vote to get the feature supported.
- Certain T-SQL features, such as:
  - *WITHIN GROUP* clause in the STRING_AGG string function.

Next steps

Now that you have the latest version of SSDT, you're ready to connect to your SQL pool.

Continuous integration and deployment for dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

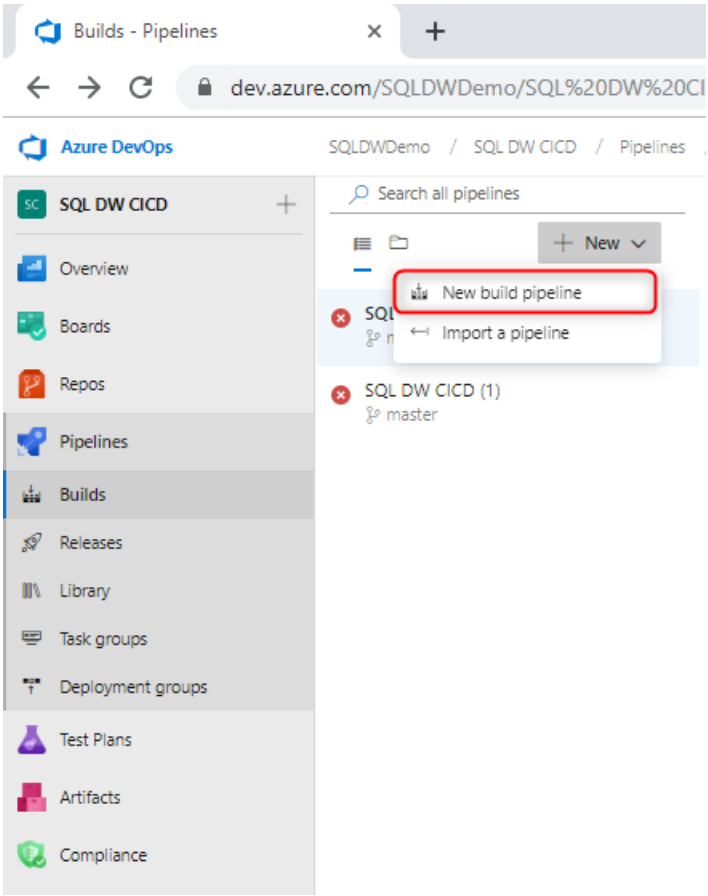
This simple tutorial outlines how to integrate your SQL Server Data tools (SSDT) database project with Azure DevOps and leverage Azure Pipelines to set up continuous integration and deployment. This tutorial is the second step in building your continuous integration and deployment pipeline for data warehousing.

Before you begin

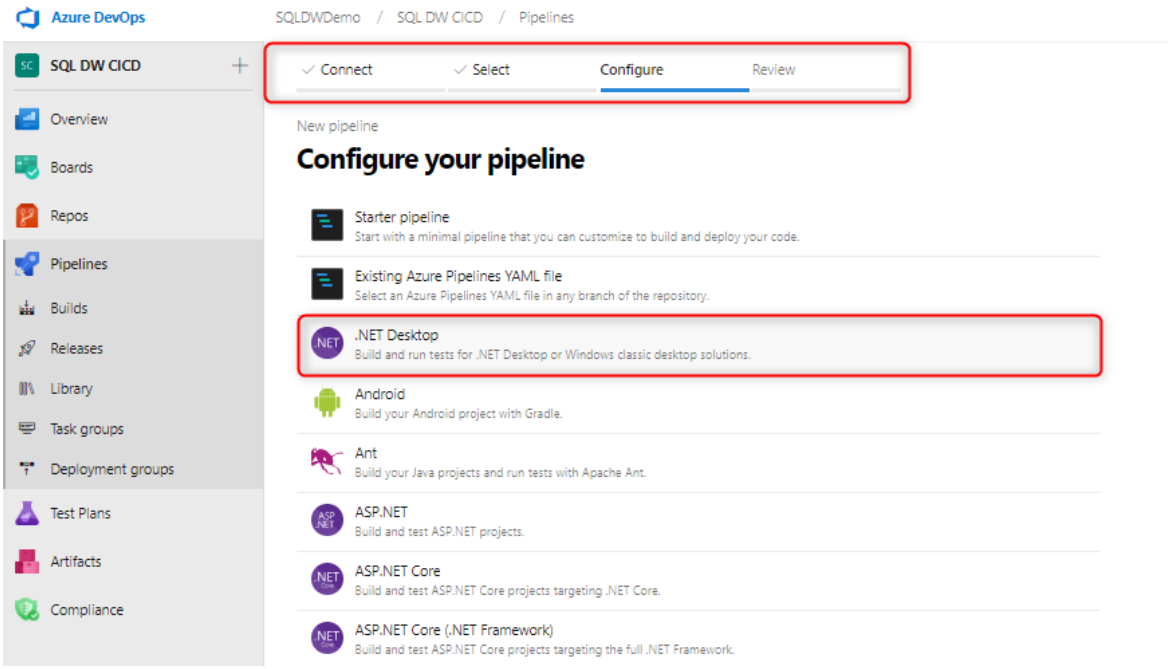
- Go through the source control integration tutorial
- Set up and connect to Azure DevOps

Continuous integration with Visual Studio build

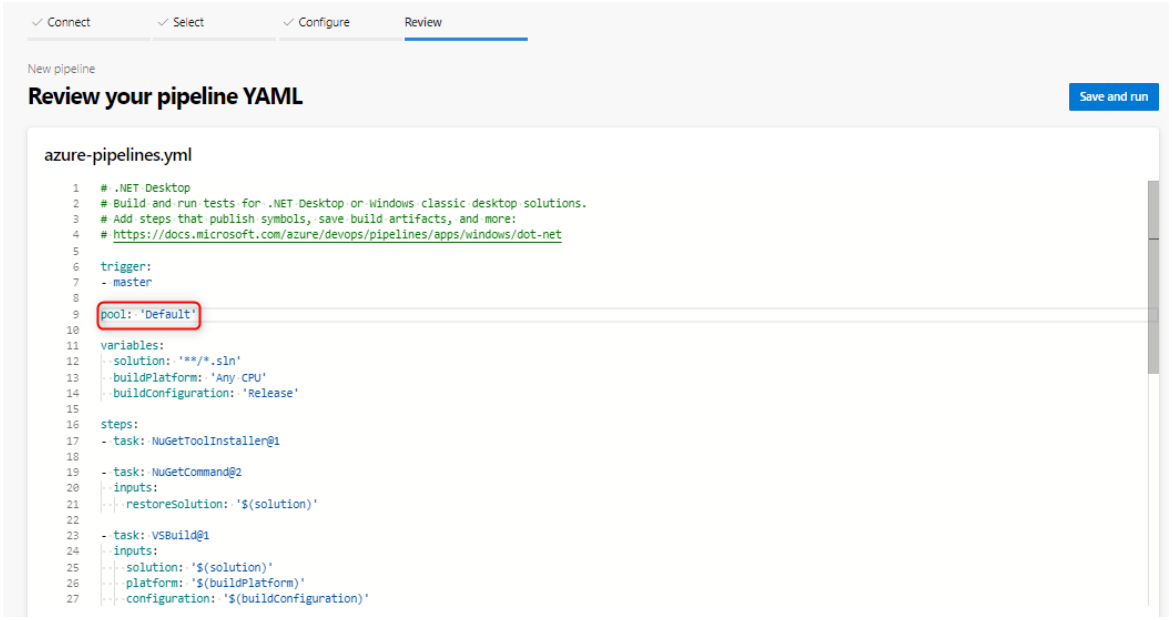
1. Navigate to Azure Pipelines and create a new build pipeline.



2. Select your source code repository (Azure Repos Git) and select the .NET Desktop app template.



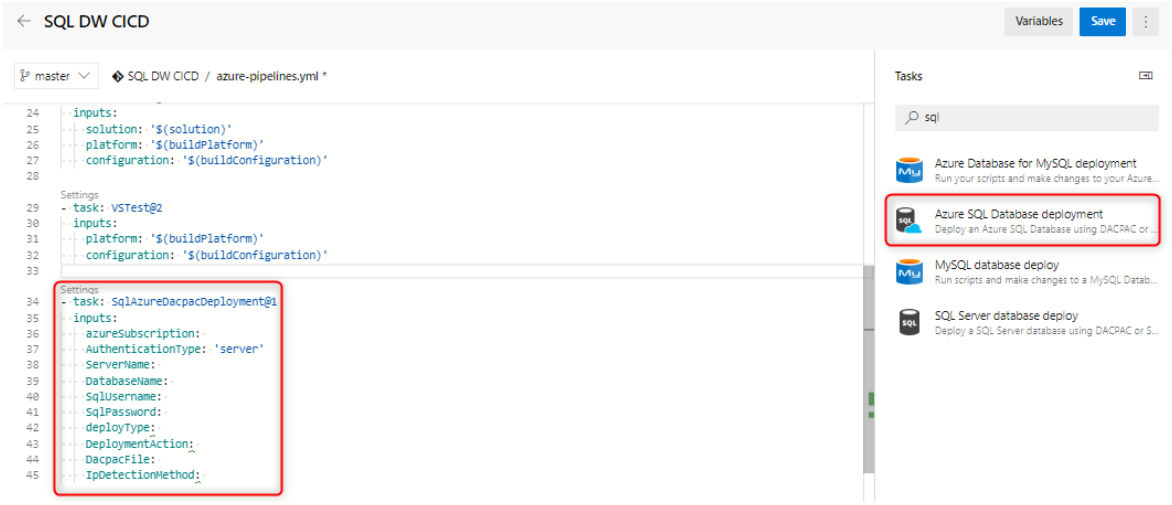
3. Edit your YAML file to use the proper pool of your agent. Your YAML file should look something like this:



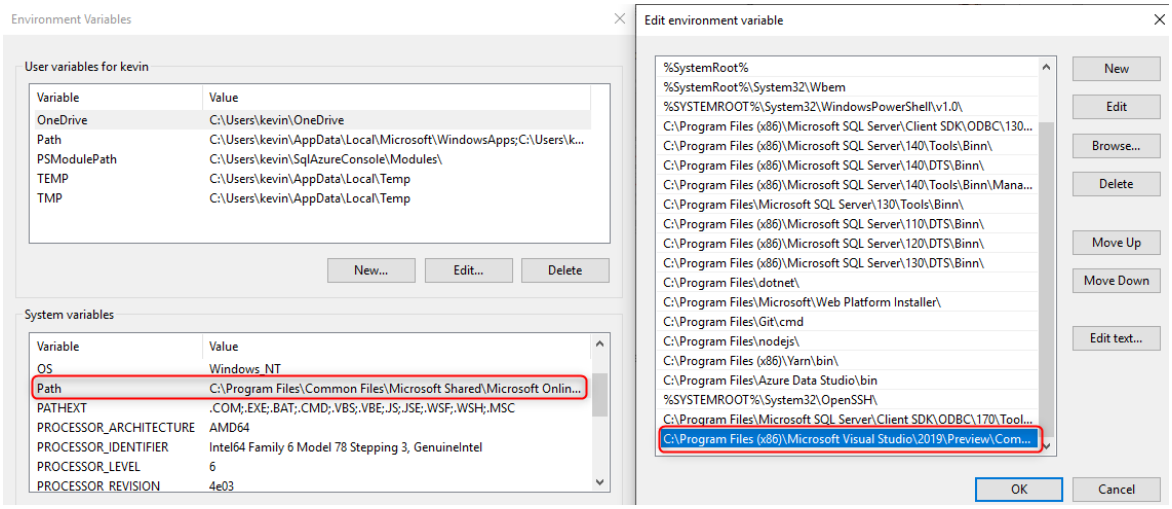
At this point, you have a simple environment where any check-in to your source control repository main branch should automatically trigger a successful Visual Studio build of your database project. Validate the automation is working end to end by making a change in your local database project and checking in that change to your main branch.

Continuous deployment with the Azure Synapse Analytics (or Database) deployment task

1. Add a new task using the Azure SQL Database deployment task and fill in the required fields to connect to your target data warehouse. When this task runs, the DACPAC generated from the previous build process is deployed to the target data warehouse. You can also use the Azure Synapse Analytics deployment task.



2. If you are using a self-hosted agent, make sure you set your environment variable to use the correct SqlPackage.exe for Azure Synapse Analytics. The path should look something like this:



C:\Program Files (x86)\Microsoft Visual Studio\2019\Preview\Common7\IDE\Extensions\Microsoft\SQLDB\DAC\150

Run and validate your pipeline. You can make changes locally and check in changes to your source control that should generate an automatic build and deployment.

Next steps

- Explore Dedicated SQL pool (formerly SQL DW) architecture
- Quickly create a dedicated SQL pool (formerly SQL DW)
- Load sample data
- Explore Videos

Deploy linked ARM templates with VSTS

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article describes how to deploy linked Azure Resource Manager (ARM) templates with Visual Studio Team Services (VSTS).

Overview

When dealing with deploying many components in Azure, a single ARM template might be challenging to manage and maintain. ARM linked templates allow you to make your deployment more modular and makes the templates easier to manage. When dealing with large deployments, it is highly recommended to consider breaking down your deployment into a main template and multiple linked templates representing different components of your deployment.

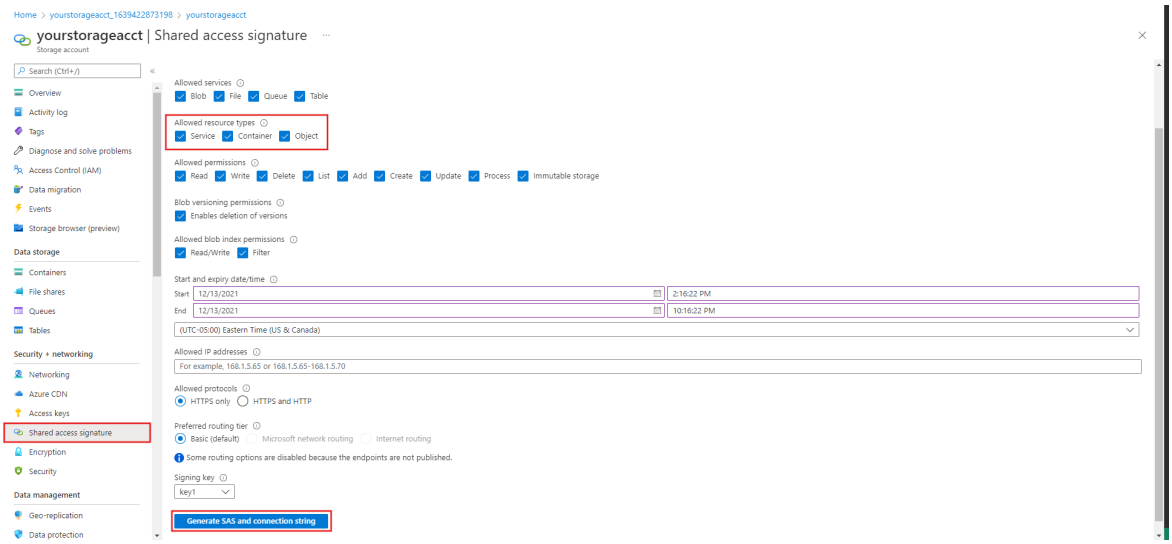
Deploying ARM templates can be performed using several different methods such as using PowerShell, Azure CLI, and Azure portal. A recommended approach however is to adopt one of DevOps practices, namely continuous deployment. VSTS is an application lifecycle management tool hosted in the cloud and offered as a service. One of the capabilities VSTS offers is release management.

This article describes how you can deploy linked ARM templates using the release management feature of VSTS. In order for the linked templates to be deployed properly, they will need to be stored in a location that can be reached by the Azure Resource Manager, such as Azure Storage; so we will show how Azure Storage can be used to stage the ARM template files. We will also show some recommended practices around keeping secrets protected using Azure Key Vault.

The scenario we walk through here is to deploy VNet with a Network Security Group (NSG) structured as linked templates. We will use VSTS to show how continuous deployment can be set up to enable teams to continuously update Azure with new changes each time there is a modification to the template.

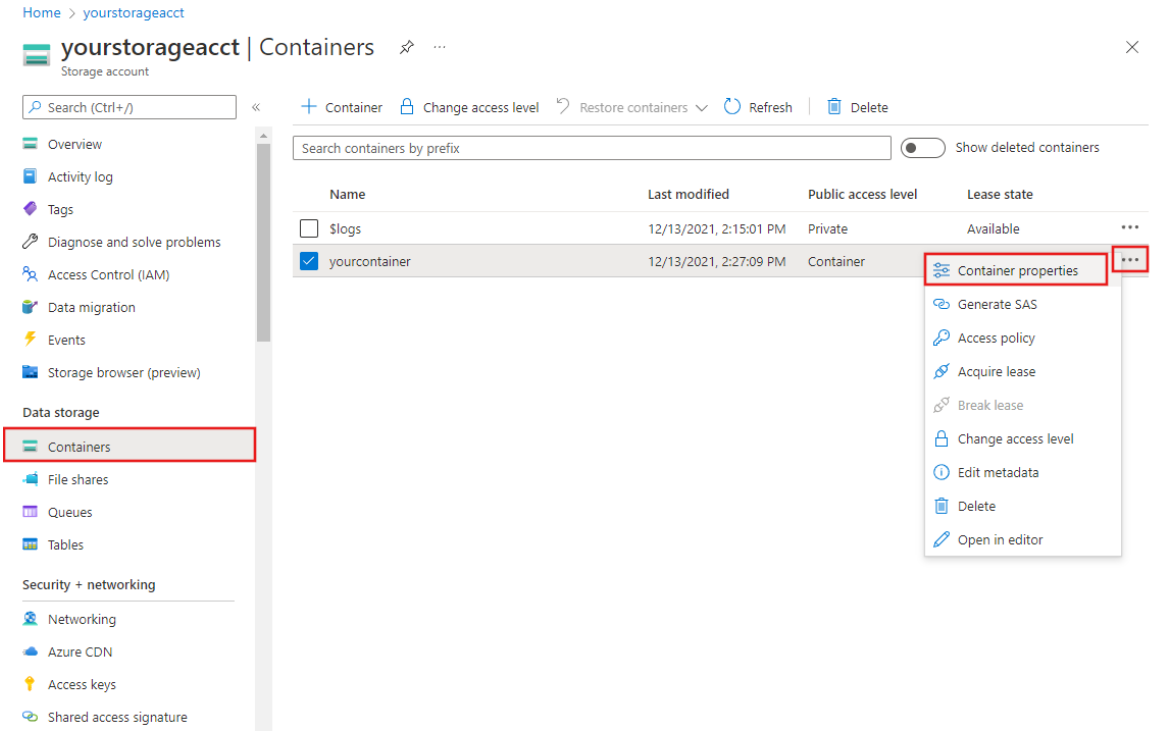
Create an Azure Storage account

1. Log in to the Azure portal and create an Azure Storage account following the steps documented here.
2. Once deployment is complete, navigate to the storage account and select **Shared access signature**. Select Service, Container, and Object for the **Allowed resource types**. Then select **Generate SAS and connection string**. Copy the SAS token and keep it available since we will use it later.



3. Select the storage account Containers page and create a new Container.

4. Select the new Container properties.



5. Copy the URL field and keep it handy. We will need it later along with the SAS token from above.

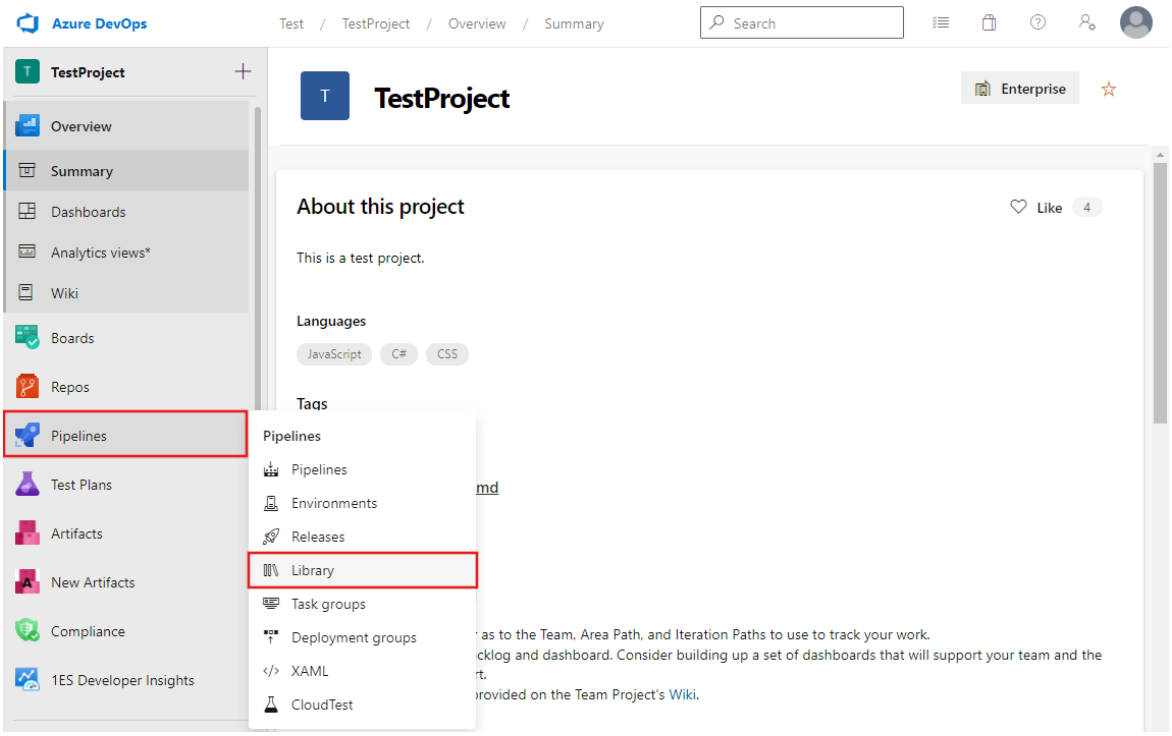
Protect secrets with Azure Key Vault

1. In the Azure portal, create an Azure Key Vault resource.
2. Select the Azure Key Vault you created above and then select Secrets.
3. Select Generate/Import to add the SAS Token.
4. For the Name property, enter `StorageSASToken` and then provide the Azure Storage shared access signature key you copied in a previous step for the Value.
5. Select Create.

Link Azure Key Vault to VSTS

1. Log in to your Azure DevOps organization and navigate to your project.

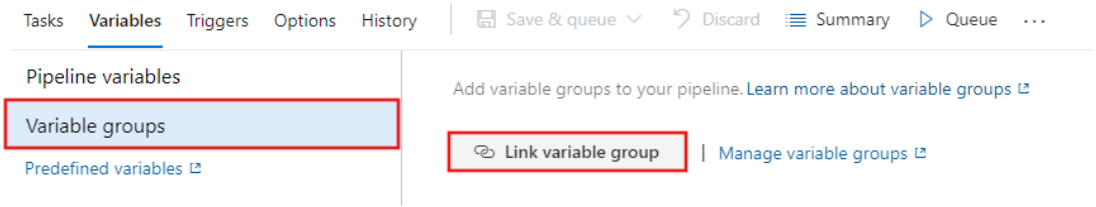
2. Go to **Library** under **Pipelines** in the navigation pane.



3. Under **Variable group**, create a new group and for **Variable group name** enter `AzureKeyVaultSecrets`.
4. Toggle **Link secrets from an Azure key vault as variables**.
5. Select your Azure subscription and then the Azure Key Vault you created earlier, and then select Authorize.
6. Once authorization is successful, you can add variables by clicking **Add** and will be presented with the option to add references to the secrets in the Azure Key Vault. Add a reference to the `StorageSASToken` created above, and save it.

Setup continuous deployment using VSTS

1. Follow steps listed in the article Automate continuous integration using Azure Pipelines releases.
2. A few changes are required from the above steps in order to use a linked ARM template deployment:
   - Under the release, link the variable group created earlier:



   - Linked ARM template:
     - For Template, point to ArmTemplate_master.json instead of ArmTemplateForFactory.json
     - For Template Parameters, point to 'ArmTemplateParameters_master.json' instead of 'ArmTemplateParametersForFactory.json'
   - Under override Template parameters update two additional parameters
     - **containerUri** – Paste the URL of container created above.
     - **containerSasToken** - If the secret's name is 'StorageSASToken', enter '$(StorageSASToken)' for this value.
3. Save the release pipeline and trigger a release.

Next steps

- Automate continuous integration using Azure Pipelines releases

Query storage files with serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 8 minutes to read • Edit Online

Serverless SQL pool enables you to query data in your data lake. It offers a T-SQL query surface area that accommodates semi-structured and unstructured data queries. For querying, the following T-SQL aspects are supported:

- Full SELECT surface area, including majority of SQL functions and operators.
- CREATE EXTERNAL TABLE AS SELECT (CETAS) creates an external table and then exports, in parallel, the results of a Transact-SQL SELECT statement to Azure Storage.

For more information on what is vs. what isn't currently supported, read the serverless SQL pool overview article, or the following articles:

- Develop storage access where you can learn how to use External table and OPENROWSET function to read data from storage.
- Control storage access where you can learn how to enable Synapse SQL to access storage using SAS authentication, or Managed Identity of the workspace.

Overview

To support a smooth experience for in place querying of data that's located in Azure Storage files, serverless SQL pool uses the OPENROWSET function with additional capabilities:

- Query multiple files or folders
- PARQUET file format
- Query CSV and delimited text (field terminator, row terminator, escape char)
- DELTA LAKE format
- Read a chosen subset of columns
- Schema inference
- filename function
- filepath function
- Work with complex types and nested or repeated data structures

Query PARQUET files

To query Parquet source data, use FORMAT = 'PARQUET':

```
SELECT * FROM
OPENROWSET( BULK N'https://myaccount.dfs.core.windows.net/mycontainer/mysubfolder/data.parquet', FORMAT =
'PARQUET')
WITH (C1 int, C2 varchar(20), C3 varchar(max)) as rows
```

Review the Query Parquet files article for usage examples.

Query CSV files

To query CSV source data, use FORMAT = 'CSV'. You can specify schema of the CSV file as part of `OPENROWSET`

function when you query CSV files:

```
SELECT * FROM
OPENROWSET( BULK N'https://myaccount.dfs.core.windows.net/mycontainer/mysubfolder/data.csv', FORMAT = 'CSV',
PARSER_VERSION='2.0')
WITH (C1 int, C2 varchar(20), C3 varchar(max)) as rows
```

There are some additional options that can be used to adjust parsing rules to custom CSv format:

- ESCAPE_CHAR = 'char' Specifies the character in the file that is used to escape itself and all delimiter values in the file. If the escape character is followed by a value other than itself, or any of the delimiter values, the escape character is dropped when reading the value. The ESCAPE_CHAR parameter will be applied whether the FIELDQUOTE is or isn't enabled. It won't be used to escape the quoting character. The quoting character must be escaped with another quoting character. Quoting character can appear within column value only if value is encapsulated with quoting characters.
- FIELDTERMINATOR='field_terminator' Specifies the field terminator to be used. The default field terminator is a comma (",")
- ROWTERMINATOR='row_terminator' Specifies the row terminator to be used. The default row terminator is a newline character:**\r\n**.

Query DELTA LAKE format

To query Delta Lake source data, use FORMAT = 'DELTA' and reference the root folder containing your Delta Lake files.

```
SELECT * FROM
OPENROWSET( BULK N'https://myaccount.dfs.core.windows.net/mycontainer/mysubfolder', FORMAT = 'DELTA')
WITH (C1 int, C2 varchar(20), C3 varchar(max)) as rows
```

The root folder must contain a subfolder called `_delta_log`. Review the query Delta Lake format article for usage examples.

File schema

SQL language in Synapse SQL enables you to define schema of the file as part of `OPENROWSET` function and read all or subset of columns, or it tries to automatically determine column types from the file using schema inference.

Read a chosen subset of columns

To specify columns that you want to read, you can provide an optional WITH clause within your `OPENROWSET` statement.

- If there are CSV data files, to read all the columns, provide column names and their data types. If you want a subset of columns, use ordinal numbers to pick the columns from the originating data files by ordinal. Columns will be bound by the ordinal designation.
- If there are Parquet data files, provide column names that match the column names in the originating data files. Columns will be bound by name.

```
SELECT * FROM
OPENROWSET( BULK N'https://myaccount.dfs.core.windows.net/mycontainer/mysubfolder/data.parquet', FORMAT =
'PARQUET')
WITH (
      C1 int,
      C2 varchar(20),
      C3 varchar(max)
) as rows
```

For every column, you need to specify column name and type in `WITH` clause. For samples, refer to Read CSV files without specifying all columns.

Schema inference

By omitting the WITH clause from the `OPENROWSET` statement, you can instruct the service to auto detect (infer) the schema from underlying files.

```
SELECT * FROM
OPENROWSET( BULK N'https://myaccount.dfs.core.windows.net/mycontainer/mysubfolder/data.parquet', FORMAT =
'PARQUET')
```

Make sure appropriate inferred data types are used for optimal performance.

Query multiple files or folders

To run a T-SQL query over a set of files within a folder or set of folders while treating them as a single entity or rowset, provide a path to a folder or a pattern (using wildcards) over a set of files or folders.

The following rules apply:

- Patterns can appear either in part of a directory path or in a filename.
- Several patterns can appear in the same directory step or file name.
- If there are multiple wildcards, then files within all matching paths will be included in the resulting file set.

```
SELECT * FROM
OPENROWSET( BULK N'https://myaccount.dfs.core.windows.net/myroot/*/mysubfolder/*.parquet', FORMAT =
'PARQUET' ) as rows
```

Refer to Query folders and multiple files for usage examples.

File metadata functions

Filename function

This function returns the file name that the row originates from.

To query specific files, read the Filename section in the Query specific files article.

Return data type is nvarchar(1024). For optimal performance, always cast result of filename function to appropriate data type. If you use character data type, make sure appropriate length is used.

Filepath function

This function returns a full path or a part of path:

- When called without parameter, returns the full file path that a row originates from.
- When called with parameter, it returns part of path that matches the wildcard on position specified in the

parameter. For example, parameter value 1 would return part of path that matches the first wildcard.

For additional information, read the Filepath section of the Query specific files article.

Return data type is nvarchar(1024). For optimal performance, always cast result of filepath function to appropriate data type. If you use character data type, make sure appropriate length is used.

Work with complex types and nested or repeated data structures

To enable a smooth experience with data stored in nested or repeated data types, such as in Parquet files, serverless SQL pool has added the extensions that follow.

Project nested or repeated data

To project data, run a SELECT statement over the Parquet file that contains columns of nested data types. On output, nested values will be serialized into JSON and returned as a varchar(8000) SQL data type.

```
SELECT * FROM
OPENROWSET
(   BULK 'unstructured_data_path' ,
    FORMAT = 'PARQUET' )
[AS alias]
```

For more detailed information, refer to the Project nested or repeated data section of the Query Parquet nested types article.

Access elements from nested columns

To access nested elements from a nested column, such as Struct, use "dot notation" to concatenate field names into the path. Provide the path as column_name in the WITH clause of the `OPENROWSET` function.

The syntax fragment example is as follows:

```
OPENROWSET
(   BULK 'unstructured_data_path' ,
    FORMAT = 'PARQUET' )
WITH ({'column_name' 'column_type',})
[AS alias]
'column_name' ::= '[field_name.] field_name'
```

By default, the `OPENROWSET` function matches the source field name and path with the column names provided in the WITH clause. Elements contained at different nesting levels within the same source Parquet file can be accessed via the WITH clause.

Return values

- Function returns a scalar value, such as int, decimal, and varchar, from the specified element, and on the specified path, for all Parquet types that aren't in the Nested Type group.
- If the path points to an element that is of a Nested Type, the function returns a JSON fragment starting from the top element on the specified path. The JSON fragment is of type varchar(8000).
- If the property can't be found at the specified column_name, the function returns an error.
- If the property can't be found at the specified column_path, depending on Path mode, the function returns an error when in strict mode or null when in lax mode.

For query samples, review the Access elements from nested columns section in the Query Parquet nested types article.

Access elements from repeated columns

To access elements from a repeated column, such as an element of an Array or Map, use the JSON_VALUE

function for every scalar element you need to project and provide:

- Nested or repeated column, as the first parameter
- A JSON path that specifies the element or property to access, as a second parameter

To access non-scalar elements from a repeated column, use the JSON_QUERY function for every non-scalar element you need to project and provide:

- Nested or repeated column, as the first parameter
- A JSON path that specifies the element or property to access, as a second parameter

See syntax fragment below:

```
SELECT
   { JSON_VALUE (column_name, path_to_sub_element), }
   { JSON_QUERY (column_name [ , path_to_sub_element ]), )
FROM
OPENROWSET
(   BULK 'unstructured_data_path' ,
    FORMAT = 'PARQUET' )
[AS alias]
```

You can find query samples for accessing elements from repeated columns in the Query Parquet nested types article.

Query samples

You can learn more about querying various types of data using the sample queries.

Tools

The tools you need to issue queries: - Azure Synapse Studio - Azure Data Studio - SQL Server Management Studio

Demo setup

Your first step is to **create a database** where you'll execute the queries. Then you'll initialize the objects by executing setup script on that database.

This setup script will create the data sources, database scoped credentials, and external file formats that are used to read data in these samples.

> **NOTE**
> Databases are only used for viewing metadata, not for actual data. Write down the database name that you use, you will need it later on.

```
CREATE DATABASE mydbname;
```

Provided demo data

Demo data contains the following data sets:

- NYC Taxi - Yellow Taxi Trip Records - part of public NYC data set in CSV and Parquet format
- Population data set in CSV format
- Sample Parquet files with nested columns
- Books in JSON format

FOLDER PATH	DESCRIPTION
/csv/	Parent folder for data in CSV format
/csv/population/ /csv/population-unix/ /csv/population-unix-hdr/ /csv/population-unix-hdr-escape /csv/population-unix-hdr-quoted	Folders with Population data files in different CSV formats.
/csv/taxi/	Folder with NYC public data files in CSV format
/parquet/	Parent folder for data in Parquet format
/parquet/taxi	NYC public data files in Parquet format, partitioned by year, and month using Hive/Hadoop partitioning scheme.
/parquet/nested/	Sample Parquet files with nested columns
/json/	Parent folder for data in JSON format
/json/books/	JSON files with books data

Next steps

For more information on how to query different file types, and to create and use views, see the following articles:

- Query CSV files
- Query Parquet files
- Query JSON files
- Query nested values
- Query folders and multiple CSV files
- Use file metadata in queries
- Create and use views

Query CSV files

8/19/2022 • 8 minutes to read • Edit Online

In this article, you'll learn how to query a single CSV file using serverless SQL pool in Azure Synapse Analytics. CSV files may have different formats:

- With and without a header row
- Comma and tab-delimited values
- Windows and Unix style line endings
- Non-quoted and quoted values, and escaping characters

All of the above variations will be covered below.

Quickstart example

`OPENROWSET` function enables you to read the content of CSV file by providing the URL to your file.

Read a csv file

The easiest way to see to the content of your `CSV` file is to provide file URL to `OPENROWSET` function, specify csv `FORMAT`, and 2.0 `PARSER_VERSION`. If the file is publicly available or if your Azure AD identity can access this file, you should be able to see the content of the file using the query like the one shown in the following example:

```
select top 10 *
from openrowset(
    bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-
19/ecdc_cases/latest/ecdc_cases.csv',
    format = 'csv',
    parser_version = '2.0',
    firstrow = 2 ) as rows
```

Option `firstrow` is used to skip the first row in the CSV file that represents header in this case. Make sure that you can access this file. If your file is protected with SAS key or custom identity, your would need to setup server level credential for sql login.

> **IMPORTANT**
>
> If your CSV file contains UTF-8 characters, make sure that you are using a UTF-8 database collation (for example `Latin1_General_100_CI_AS_SC_UTF8`). A mismatch between text encoding in the file and the collation might cause unexpected conversion errors. You can easily change default collation of the current database using the following T-SQL statement: `alter database current collate Latin1_General_100_CI_AI_SC_UTF8`

Data source usage

Previous example uses full path to the file. As an alternative, you can create an external data source with the location that points to the root folder of the storage:

```
create external data source covid
with ( location = 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases' );
```

Once you create a data source, you can use that data source and the relative path to the file in `OPENROWSET` function:

```
select top 10 *
from openrowset(
        bulk 'latest/ecdc_cases.csv',
        data_source = 'covid',
        format = 'csv',
        parser_version ='2.0',
        firstrow = 2
    ) as rows
```

If a data source is protected with SAS key or custom identity you can configure data source with database scoped credential.

Explicitly specify schema

`OPENROWSET` enables you to explicitly specify what columns you want to read from the file using `WITH` clause:

```
select top 10 *
from openrowset(
        bulk 'latest/ecdc_cases.csv',
        data_source = 'covid',
        format = 'csv',
        parser_version ='2.0',
        firstrow = 2
    ) with (
        date_rep date 1,
        cases int 5,
        geo_id varchar(6) 8
    ) as rows
```

The numbers after a data type in the `WITH` clause represent column index in the CSV file.

> **IMPORTANT**
> If your CSV file contains UTF-8 characters, make sure that you are explicilty specifying some UTF-8 collation (for example `Latin1_General_100_CI_AS_SC_UTF8`) for all columns in `WITH` clause or set some UTF-8 collation at database level. Mismatch between text encoding in the file and collation might cause unexpected conversion errors. You can easily change default collation of the current database using the following T-SQL statement: `alter database current collate Latin1_General_100_CI_AI_SC_UTF8` You can easily set collation on the colum types using the following definition: `geo_id varchar(6) collate Latin1_General_100_CI_AI_SC_UTF8 8`

In the following sections you can see how to query various types of CSV files.

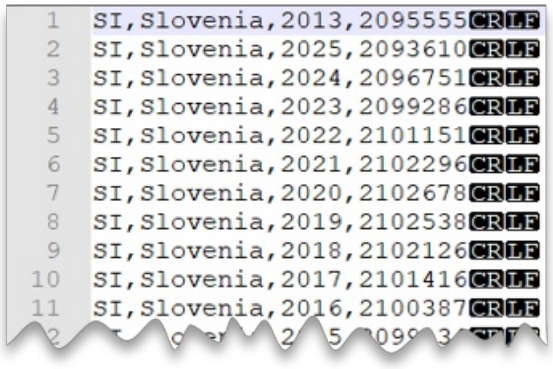
Prerequisites

Your first step is to **create a database** where the tables will be created. Then initialize the objects by executing setup script on that database. This setup script will create the data sources, database scoped credentials, and external file formats that are used in these samples.

Windows style new line

The following query shows how to read a CSV file without a header row, with a Windows-style new line, and comma-delimited columns.

File preview:




```
SELECT *
FROM OPENROWSET(
        BULK 'csv/population/population.csv',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '\n'
    )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017;
```

Unix-style new line

The following query shows how to read a file without a header row, with a Unix-style new line, and comma-delimited columns. Note the different location of the file as compared to the other examples.

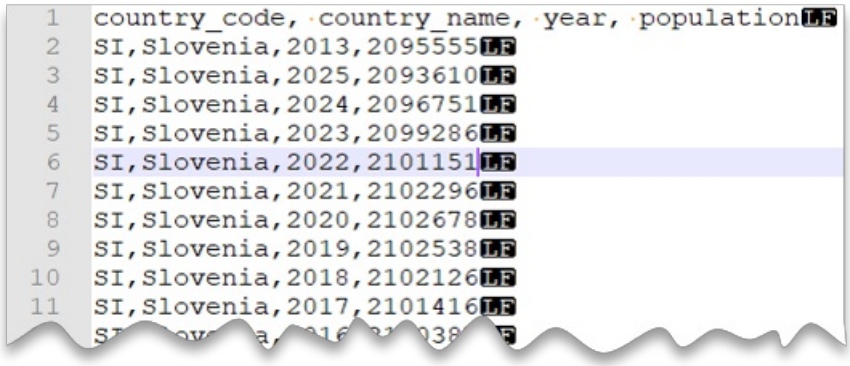
File preview:

```
SELECT *
FROM OPENROWSET(
        BULK 'csv/population-unix/population.csv',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '0x0a'
    )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017;
```

Header row

The following query shows how to a read file with a header row, with a Unix-style new line, and comma-delimited columns. Note the different location of the file as compared to the other examples.

File preview:



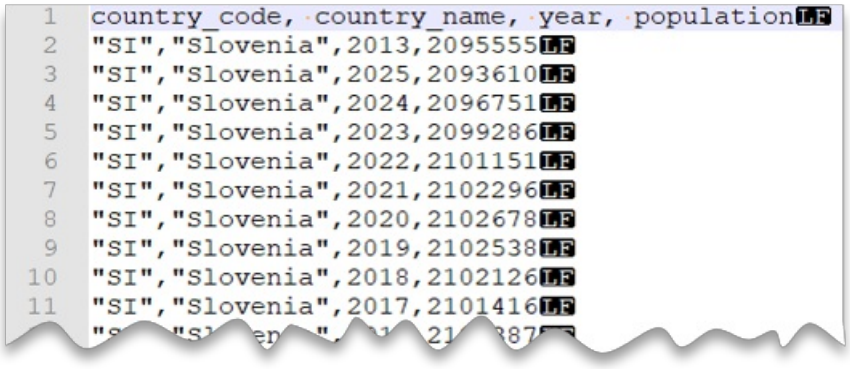

```
SELECT *
FROM OPENROWSET(
    BULK 'csv/population-unix-hdr/population.csv',
    DATA_SOURCE = 'SqlOnDemandDemo',
    FORMAT = 'CSV', PARSER_VERSION = '2.0',
    FIELDTERMINATOR =',',
    HEADER_ROW = TRUE
    ) AS [r]
```

Option `HEADER_ROW = TRUE` will result in reading column names from the header row in file. It is great for exploration purposes when you are not familiar with file content. For best performance see Use appropriate data types section in Best practices. Also, you can read more about OPENROWSET syntax here.

Custom quote character

The following query shows how to read a file with a header row, with a Unix-style new line, comma-delimited columns, and quoted values. Note the different location of the file as compared to the other examples.

File preview:




```
SELECT *
FROM OPENROWSET(
        BULK 'csv/population-unix-hdr-quoted/population.csv',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '0x0a',
        FIRSTROW = 2,
        FIELDQUOTE = '"'
    )
    WITH (
        [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
        [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
        [year] smallint,
        [population] bigint
    ) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017;
```

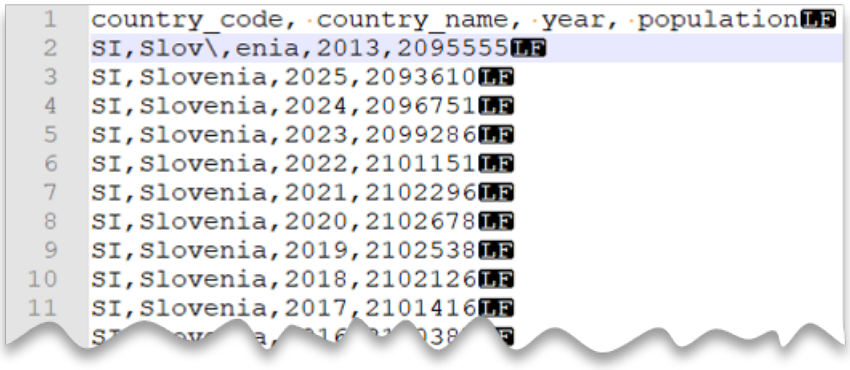
NOTE

This query would return the same results if you omitted the FIELDQUOTE parameter since the default value for FIELDQUOTE is a double-quote.

Escape characters

The following query shows how to read a file with a header row, with a Unix-style new line, comma-delimited columns, and an escape char used for the field delimiter (comma) within values. Note the different location of the file as compared to the other examples.

File preview:



```
SELECT *
FROM OPENROWSET(
        BULK 'csv/population-unix-hdr-escape/population.csv',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '0x0a',
        FIRSTROW = 2,
        ESCAPECHAR = '\\'
    )
    WITH (
        [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
        [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
        [year] smallint,
        [population] bigint
    ) AS [r]
WHERE
    country_name = 'Slovenia';
```

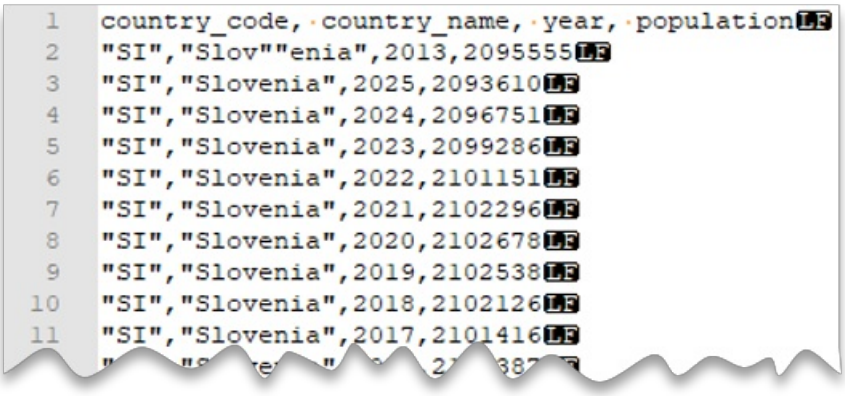
NOTE

This query would fail if ESCAPECHAR is not specified since the comma in "Slov,enia" would be treated as field delimiter instead of part of the country/region name. "Slov,enia" would be treated as two columns. Therefore, the particular row would have one column more than the other rows, and one column more than you defined in the WITH clause.

Escape quoting characters

The following query shows how to read a file with a header row, with a Unix-style new line, comma-delimited columns, and an escaped double quote char within values. Note the different location of the file as compared to the other examples.

File preview:



```
SELECT *
FROM OPENROWSET(
        BULK 'csv/population-unix-hdr-escape-quoted/population.csv',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '0x0a',
        FIRSTROW = 2
    )
    WITH (
        [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
        [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
        [year] smallint,
        [population] bigint
    ) AS [r]
WHERE
    country_name = 'Slovenia';
```

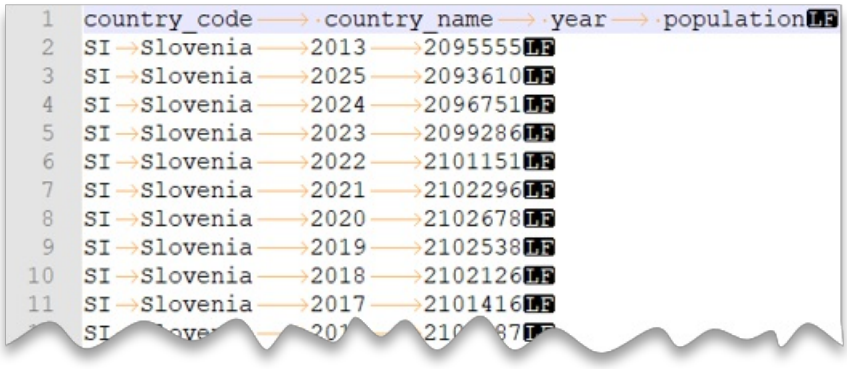
NOTE

The quoting character must be escaped with another quoting character. Quoting character can appear within column value only if value is encapsulated with quoting characters.

Tab-delimited files

The following query shows how to read a file with a header row, with a Unix-style new line, and tab-delimited columns. Note the different location of the file as compared to the other examples.

File preview:




```
SELECT *
FROM OPENROWSET(
        BULK 'csv/population-unix-hdr-tsv/population.csv',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIELDTERMINATOR ='\t',
        ROWTERMINATOR = '0x0a',
        FIRSTROW = 2
    )
    WITH (
        [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
        [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
        [year] smallint,
        [population] bigint
    ) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017
```

Return a subset of columns

So far, you've specified the CSV file schema using WITH and listing all columns. You can only specify columns you actually need in your query by using an ordinal number for each column needed. You'll also omit columns of no interest.

The following query returns the number of distinct country/region names in a file, specifying only the columns that are needed:

> **NOTE**
>
> Take a look at the WITH clause in the query below and note that there is "2" (without quotes) at the end of row where you define the *[country_name]* column. It means that the *[country_name]* column is the second column in the file. The query will ignore all columns in the file except the second one.

```
SELECT
    COUNT(DISTINCT country_name) AS countries
FROM OPENROWSET(
        BULK 'csv/population/population.csv',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '\n'
    )
WITH (
    --[country_code] VARCHAR (5),
    [country_name] VARCHAR (100) 2
    --[year] smallint,
    --[population] bigint
) AS [r]
```

Querying appendable files

The CSV files that are used in the query should not be changed while the query is running. In the long-running query, SQL pool may retry reads, read parts of the files, or even read the file multiple times. Changes of the file content would cause wrong results. Therefore, SQL pool fails the query if detects that modification time of any file is changed during the query execution.

In some scenarios you might want to read the files that are constantly appended. To avoid the query failures due to constantly appended files, you can allow the `OPENROWSET` function to ignore potentially inconsistent reads using the `ROWSET_OPTIONS` setting.

```
select top 10 *
from openrowset(
    bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-
19/ecdc_cases/latest/ecdc_cases.csv',
    format = 'csv',
    parser_version = '2.0',
    firstrow = 2,
    ROWSET_OPTIONS = '{"READ_OPTIONS":["ALLOW_INCONSISTENT_READS"]}') as rows
```

The `ALLOW_INCONSISTENT_READS` read option will disable the file modification time check during the query lifecycle and read whatever is available in the file. In the appendable files, the existing content is not updated, and only new rows are added. Therefore, the probability of wrong results is minimized compared to the updateable files. This option might enable you to read the frequently appended files without handling the errors. Im most of the scenarios, SQL pool will just ignore some rows that are appended to the files during the query execution.

Next steps

The next articles will show you how to:

- Querying Parquet files
- Querying folders and multiple files

Query Parquet files using serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

In this article, you'll learn how to write a query using serverless SQL pool that will read Parquet files.

Quickstart example

`OPENROWSET` function enables you to read the content of parquet file by providing the URL to your file.

Read parquet file

The easiest way to see to the content of your `PARQUET` file is to provide file URL to `OPENROWSET` function and specify parquet `FORMAT`. If the file is publicly available or if your Azure AD identity can access this file, you should be able to see the content of the file using the query like the one shown in the following example:

```
select top 10 *
from openrowset(
    bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-
19/ecdc_cases/latest/ecdc_cases.parquet',
    format = 'parquet') as rows
```

Make sure that you can access this file. If your file is protected with SAS key or custom Azure identity, you would need to setup server level credential for sql login.

> **IMPORTANT**
>
> Ensure you are using a UTF-8 database collation (for example `Latin1_General_100_BIN2_UTF8`) because string values in PARQUET files are encoded using UTF-8 encoding. A mismatch between the text encoding in the PARQUET file and the collation may cause unexpected conversion errors. You can easily change the default collation of the current database using the following T-SQL statement: `alter database current collate Latin1_General_100_BIN2_UTF8`

If you use the `Latin1_General_100_BIN2_UTF8` collation you will get an additional performance boost compared to the other collations. The `Latin1_General_100_BIN2_UTF8` collation is compatible with parquet string sorting rules. The SQL pool is able to eliminate some parts of the parquet files that will not contain data needed in the queries (file/column-segment pruning). If you use other collations, all data from the parquet files will be loaded into Synapse SQL and the filtering is happening within the SQL process. The `Latin1_General_100_BIN2_UTF8` collation has additional performance optimization that works only for parquet and CosmosDB. The downside is that you lose fine-grained comparison rules like case insensitivity.

Data source usage

Previous example uses full path to the file. As an alternative, you can create an external data source with the location that points to the root folder of the storage, and use that data source and the relative path to the file in `OPENROWSET` function:

```
create external data source covid
with ( location = 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases' );
go
select top 10 *
from openrowset(
        bulk 'latest/ecdc_cases.parquet',
        data_source = 'covid',
        format = 'parquet'
    ) as rows
```

If a data source is protected with SAS key or custom identity you can configure data source with database scoped credential.

Explicitly specify schema

`OPENROWSET` enables you to explicitly specify what columns you want to read from the file using `WITH` clause:

```
select top 10 *
from openrowset(
        bulk 'latest/ecdc_cases.parquet',
        data_source = 'covid',
        format = 'parquet'
    ) with ( date_rep date, cases int, geo_id varchar(6) ) as rows
```

IMPORTANT

Make sure that you are explicilty specifying some UTF-8 collation (for example `Latin1_General_100_BIN2_UTF8`) for all string columns in `WITH` clause or set some UTF-8 collation at database level. Mismatch between text encoding in the file and string column collation might cause unexpected conversion errors. You can easily change default collation of the current database using the following T-SQL statement:
`alter database current collate Latin1_General_100_BIN2_UTF8` You can easily set collation on the colum types using the following definition: `geo_id varchar(6) collate Latin1_General_100_BIN2_UTF8`

In the following sections you can see how to query various types of PARQUET files.

Prerequisites

Your first step is to **create a database** with a datasource that references NYC Yellow Taxi storage account. Then initialize the objects by executing setup script on that database. This setup script will create the data sources, database scoped credentials, and external file formats that are used in these samples.

Dataset

NYC Yellow Taxi dataset is used in this sample. You can query Parquet files the same way you read CSV files. The only difference is that the `FILEFORMAT` parameter should be set to `PARQUET`. Examples in this article show the specifics of reading Parquet files.

Query set of parquet files

You can specify only the columns of interest when you query Parquet files.

```
SELECT
        YEAR(tpepPickupDateTime),
        passengerCount,
        COUNT(*) AS cnt
FROM
    OPENROWSET(
        BULK 'puYear=2018/puMonth=*/*.snappy.parquet',
        DATA_SOURCE = 'YellowTaxi',
        FORMAT='PARQUET'
    ) WITH (
        tpepPickupDateTime DATETIME2,
        passengerCount INT
    ) AS nyc
GROUP BY
    passengerCount,
    YEAR(tpepPickupDateTime)
ORDER BY
    YEAR(tpepPickupDateTime),
    passengerCount;
```

Automatic schema inference

You don't need to use the OPENROWSET WITH clause when reading Parquet files. Column names and data types are automatically read from Parquet files.

The sample below shows the automatic schema inference capabilities for Parquet files. It returns the number of rows in September 2018 without specifying a schema.

> **NOTE**
> You don't have to specify columns in the OPENROWSET WITH clause when reading Parquet files. In that case, serverless SQL pool query service will utilize metadata in the Parquet file and bind columns by name.

```
SELECT TOP 10 *
FROM
    OPENROWSET(
        BULK 'puYear=2018/puMonth=9/*.snappy.parquet',
        DATA_SOURCE = 'YellowTaxi',
        FORMAT='PARQUET'
    ) AS nyc
```

Query partitioned data

The data set provided in this sample is divided (partitioned) into separate subfolders. You can target specific partitions using the filepath function. This example shows fare amounts by year, month, and payment_type for the first three months of 2017.

> **NOTE**
> The serverless SQL pool query is compatible with Hive/Hadoop partitioning scheme.

```
SELECT
        YEAR(tpepPickupDateTime),
        passengerCount,
        COUNT(*) AS cnt
FROM
    OPENROWSET(
        BULK 'puYear=*/puMonth=*/*.snappy.parquet',
        DATA_SOURCE = 'YellowTaxi',
        FORMAT='PARQUET'
    ) nyc
WHERE
    nyc.filepath(1) = 2017
    AND nyc.filepath(2) IN (1, 2, 3)
    AND tpepPickupDateTime BETWEEN CAST('1/1/2017' AS datetime) AND CAST('3/31/2017' AS datetime)
GROUP BY
    passengerCount,
    YEAR(tpepPickupDateTime)
ORDER BY
    YEAR(tpepPickupDateTime),
    passengerCount;
```

Type mapping

For Parquet type mapping to SQL native type check type mapping for Parquet.

Next steps

Advance to the next article to learn how to Query Parquet nested types.

Query JSON files using serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

In this article, you'll learn how to write a query using serverless SQL pool in Azure Synapse Analytics. The query's objective is to read JSON files using OPENROWSET.

- Standard JSON files where multiple JSON documents are stored as a JSON array.
- Line-delimited JSON files, where JSON documents are separated with new-line character. Common extensions for these types of files are `jsonl`, `ldjson`, and `ndjson`.

Read JSON documents

The easiest way to see to the content of your JSON file is to provide the file URL to the `OPENROWSET` function, specify csv `FORMAT`, and set values `0x0b` for `fieldterminator` and `fieldquote`. If you need to read line-delimited JSON files, then this is enough. If you have classic JSON file, you would need to set values `0x0b` for `rowterminator`. `OPENROWSET` function will parse JSON and return every document in the following format:

doc
{"date_rep":"2020-07-24","day":24,"month":7,"year":2020,"cases":3,"deaths":0,"geo_id":"AF"}
{"date_rep":"2020-07-25","day":25,"month":7,"year":2020,"cases":7,"deaths":0,"geo_id":"AF"}
{"date_rep":"2020-07-26","day":26,"month":7,"year":2020,"cases":4,"deaths":0,"geo_id":"AF"}
{"date_rep":"2020-07-27","day":27,"month":7,"year":2020,"cases":8,"deaths":0,"geo_id":"AF"}

If the file is publicly available, or if your Azure AD identity can access this file, you should see the content of the file using the query like the one shown in the following examples.

Read JSON files

The following sample query reads JSON and line-delimited JSON files, and returns every document as a separate row.

```
select top 10 *
from openrowset(
        bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-
19/ecdc_cases/latest/ecdc_cases.jsonl',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
go
select top 10 *
from openrowset(
        bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-
19/ecdc_cases/latest/ecdc_cases.json',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b',
        rowterminator = '0x0b' --> You need to override rowterminator to read classic JSON
    ) with (doc nvarchar(max)) as rows
```

The JSON document in the preceding sample query includes an array of objects. The query returns each object as a separate row in the result set. Make sure that you can access this file. If your file is protected with SAS key or custom identity, you would need to set up server level credential for sql login.

Data source usage

The previous example uses full path to the file. As an alternative, you can create an external data source with the location that points to the root folder of the storage, and use that data source and the relative path to the file in the `OPENROWSET` function:

```
create external data source covid
with ( location = 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases' );
go
select top 10 *
from openrowset(
        bulk 'latest/ecdc_cases.jsonl',
        data_source = 'covid',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
go
select top 10 *
from openrowset(
        bulk 'latest/ecdc_cases.json',
        data_source = 'covid',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b',
        rowterminator = '0x0b' --> You need to override rowterminator to read classic JSON
    ) with (doc nvarchar(max)) as rows
```

If a data source is protected with SAS key or custom identity, you can configure data source with database scoped credential.

In the following sections, you can see how to query various types of JSON files.

Parse JSON documents

The queries in the previous examples return every JSON document as a single string in a separate row of the result set. You can use functions `JSON_VALUE` and `OPENJSON` to parse the values in JSON documents and return them as relational values, as it's shown in the following example:

DATE_REP	CASES	GEO_ID
2020-07-24	3	AF
2020-07-25	7	AF
2020-07-26	4	AF
2020-07-27	8	AF

Sample JSON document

The query examples read *json* files containing documents with following structure:

```
{
    "date_rep":"2020-07-24",
    "day":24,"month":7,"year":2020,
    "cases":13,"deaths":0,
    "countries_and_territories":"Afghanistan",
    "geo_id":"AF",
    "country_territory_code":"AFG",
    "continent_exp":"Asia",
    "load_date":"2020-07-25 00:05:14",
    "iso_country":"AF"
}
```

> **NOTE**
>
> If these documents are stored as line-delimited JSON, you need to set `FIELDTERMINATOR` and `FIELDQUOTE` to 0x0b. If you have standard JSON format you need to set `ROWTERMINATOR` to 0x0b.

Query JSON files using JSON_VALUE

The query below shows you how to use JSON_VALUE to retrieve scalar values (`date_rep` , `countries_and_territories` , `cases`) from a JSON documents:

```
select
    JSON_VALUE(doc, '$.date_rep') AS date_reported,
    JSON_VALUE(doc, '$.countries_and_territories') AS country,
    CAST(JSON_VALUE(doc, '$.deaths') AS INT) as fatal,
    JSON_VALUE(doc, '$.cases') as cases,
    doc
from openrowset(
        bulk 'latest/ecdc_cases.jsonl',
        data_source = 'covid',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
order by JSON_VALUE(doc, '$.geo_id') desc
```

Once you extract JSON properties from a JSON document, you can define column aliases and optionally cast the textual value to some type.

Query JSON files using OPENJSON

The following query uses OPENJSON. It will retrieve COVID statistics reported in Serbia:

```
select
    *
from openrowset(
        bulk 'latest/ecdc_cases.jsonl',
        data_source = 'covid',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
    cross apply openjson (doc)
        with (  date_rep datetime2,
                cases int,
                fatal int '$.deaths',
                country varchar(100) '$.countries_and_territories')
where country = 'Serbia'
order by country, date_rep desc;
```

The results are functionally same as the results returned using the `JSON_VALUE` function. In some cases, `OPENJSON` might have advantage over `JSON_VALUE`:

- In the `WITH` clause you can explicitly set the column aliases and the types for every property. You don't need to put the `CAST` function in every column in `SELECT` list.
- `OPENJSON` might be faster if you are returning a large number of properties. If you are returning just 1-2 properties, the `OPENJSON` function might be overhead.
- You must use the `OPENJSON` function if you need to parse the array from each document, and join it with the parent row.

Next steps

The next articles in this series will demonstrate how to:

- Querying folders and multiple files
- Create and use views

Query nested types in Parquet and JSON files by using serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

In this article, you'll learn how to write a query by using serverless SQL pool in Azure Synapse Analytics. The query will read Parquet nested types. Nested types are complex structures that represent objects or arrays. Nested types can be stored in:

- Parquet, where you can have multiple complex columns that contain arrays and objects.
- Hierarchical JSON files, where you can read a complex JSON document as a single column.
- Azure Cosmos DB collections (currently under gated public preview), where every document can contain complex nested properties.

Serverless SQL pool formats all nested types as JSON objects and arrays. So you can extract or modify complex objects by using JSON functions or parse JSON data by using the OPENJSON function.

Here's an example of a query that extracts scalar and object values from the COVID-19 Open Research Dataset JSON file, which contains nested objects:

```
SELECT
    title = JSON_VALUE(doc, '$.metadata.title'),
    first_author = JSON_QUERY(doc, '$.metadata.authors[0]'),
    first_author_name = JSON_VALUE(doc, '$.metadata.authors[0].first'),
    complex_object = doc
FROM
    OPENROWSET(
        BULK
'https://azureopendatastorage.blob.core.windows.net/covid19temp/comm_use_subset/pdf_json/000b7d1517ceebb34e1
e3e817695b6de03e2fa78.json',
        FORMAT='CSV', FIELDTERMINATOR ='0x0b', FIELDQUOTE = '0x0b', ROWTERMINATOR = '0x0b'
    )
    WITH ( doc varchar(MAX) ) AS docs;
```

The `JSON_VALUE` function returns a scalar value from the field at the specified path. The `JSON_QUERY` function returns an object formatted as JSON from the field at the specified path.

> **IMPORTANT**
>
> This example uses a file from the COVID-19 Open Research Dataset. See the licence and the structure of the data here.

Prerequisites

The first step is to create a database where the datasource will be created. You'll then initialize the objects by running a setup script on the database. The setup script will create the data sources, database-scoped credentials, and external file formats that are used in the samples.

Project nested or repeated data

A Parquet file can have multiple columns with complex types. The values from these columns are formatted as JSON text and returned as VARCHAR columns. The following query reads the structExample.parquet file and shows how to read the values of the nested columns:

```
SELECT
    DateStruct, TimeStruct, TimestampStruct, DecimalStruct, FloatStruct
FROM
    OPENROWSET(
        BULK 'parquet/nested/structExample.parquet',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT='PARQUET'
    )
    WITH (
        DateStruct VARCHAR(8000),
        TimeStruct VARCHAR(8000),
        TimestampStruct VARCHAR(8000),
        DecimalStruct VARCHAR(8000),
        FloatStruct VARCHAR(8000)
    ) AS [r];
```

This query returns the following result. The content of every nested object is returned as JSON text.

DATESTRUCT	TIMESTRUCT	TIMESTAMPSTRUCT	DECIMALSTRUCT	FLOATSTRUCT
{"Date":"2009-04-25"}	{"Time":"20:51:54.3598000"}	{"Timestamp":"5501-04-08 12:13:57.4821000"}	{"Decimal":11143412.25350}	{"Float":0.5}
{"Date":"1916-04-29"}	{"Time":"00:16:04.6778000"}	{"Timestamp":"1990-06-30 20:50:52.6828000"}	{"Decimal":1963545.62800}	{"Float":-2.125}

The following query reads the justSimpleArray.parquet file. It projects all columns from the Parquet file, including nested and repeated data.

```
SELECT
    SimpleArray
FROM
    OPENROWSET(
        BULK 'parquet/nested/justSimpleArray.parquet',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT='PARQUET'
    ) AS [r];
```

This query will return the following result:

SIMPLEARRAY
[11,12,13]
[21,22,23]

Read properties from nested object columns

The `JSON_VALUE` function enables you to return values from columns formatted as JSON text:

```
SELECT
    title = JSON_VALUE(complex_column, '$.metadata.title'),
    first_author_name = JSON_VALUE(complex_column, '$.metadata.authors[0].first'),
    body_text = JSON_VALUE(complex_column, '$.body_text.text'),
    complex_column
FROM
    OPENROWSET( BULK
'https://azureopendatastorage.blob.core.windows.net/covid19temp/comm_use_subset/pdf_json/000b7d1517ceebb34e1e3e817695b6de03e2fa78.json',
                FORMAT='CSV', FIELDTERMINATOR ='0x0b', FIELDQUOTE = '0x0b', ROWTERMINATOR = '0x0b' ) WITH (
complex_column varchar(MAX) ) AS docs;
```

The result is shown in the following table:

TITLE	FIRST_AUTHOR_NAME	BODY_TEXT	COMPLEX_COLUMN
Supplementary Information An eco-epidemiolo...	Julien	- Figure S1 : Phylogeny of...	{ "paper_id": "000b7d1517ceebb34e1e3e817695b6de03e2fa "metadata": { "title": "Supplementary Information An eco-epidemiological stu Morbilli-related paramyxovirus infecti Madagascar bats reveals host-switching the dominant macro-evolutionary mechan "authors": [{ "first": "Julien"

Unlike JSON files, which in most cases return a single column that contains a complex JSON object, Parquet files can have multiple complex columns. You can read the properties of nested columns by using the `JSON_VALUE` function on each column. `OPENROWSET` enables you to directly specify the paths of the nested properties in a `WITH` clause. You can set the paths as the name of a column, or you can add a JSON path expression after the column type.

The following query reads the structExample.parquet file and shows how to surface elements of a nested column. There are two ways to reference a nested value:

- By specifying the nested value path expression after the type specification.
- By formatting the column name as a nested path by using do "." to reference the fields.

```
SELECT
    *
FROM
    OPENROWSET(
        BULK 'parquet/nested/structExample.parquet',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT='PARQUET'
    )
    WITH (
        [DateValue] DATE '$.DateStruct.Date',
        [TimeStruct.Time] TIME,
        [TimestampStruct.Timestamp] DATETIME2,
        DecimalValue DECIMAL(18, 5) '$.DecimalStruct.Decimal',
        [FloatStruct.Float] FLOAT
    ) AS [r];
```

Access elements from repeated columns

The following query reads the justSimpleArray.parquet file and uses JSON_VALUE to retrieve a scalar element from within a repeated column, like an array or map:

```
SELECT
    *,
    JSON_VALUE(SimpleArray, '$[0]') AS FirstElement,
    JSON_VALUE(SimpleArray, '$[1]') AS SecondElement,
    JSON_VALUE(SimpleArray, '$[2]') AS ThirdElement
FROM
    OPENROWSET(
        BULK 'parquet/nested/justSimpleArray.parquet',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT='PARQUET'
    ) AS [r];
```

Here's the result:

SIMPLEARRAY	FIRSTELEMENT	SECONDELEMENT	THIRDELEMENT
[11,12,13]	11	12	13

SIMPLEARRAY	FIRSTELEMENT	SECONDELEMENT	THIRDELEMENT
[21,22,23]	21	22	23

Access sub-objects from complex columns

The following query reads the mapExample.parquet file and uses JSON_QUERY to retrieve a non-scalar element from within a repeated column, like an array or map:

```
SELECT
    MapOfPersons,
    JSON_QUERY(MapOfPersons, '$."John Doe"') AS [John]
FROM
    OPENROWSET(
        BULK 'parquet/nested/mapExample.parquet',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT='PARQUET'
    ) AS [r];
```

You can also explicitly reference the columns that you want to return in a `WITH` clause:

```
SELECT DocId,
    MapOfPersons,
    JSON_QUERY(MapOfPersons, '$."John Doe"') AS [John]
FROM
    OPENROWSET(
        BULK 'parquet/nested/mapExample.parquet',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT='PARQUET'
    )
    WITH (DocId bigint, MapOfPersons VARCHAR(max)) AS [r];
```

The structure `MapOfPersons` is returned as a VARCHAR column and formatted as a JSON string.

Project values from repeated columns

If you have an array of scalar values (for example `[1,2,3]`) in some columns, you can easily expand them and join them with the main row by using this script:

```
SELECT
    SimpleArray, Element
FROM
    OPENROWSET(
        BULK 'parquet/nested/justSimpleArray.parquet',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT='PARQUET'
    ) AS arrays
    CROSS APPLY OPENJSON (SimpleArray) WITH (Element int '$') as array_values
```

Next steps

The next article will show you how to Query JSON files.

Query Delta Lake files using serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

In this article, you'll learn how to write a query using serverless Synapse SQL pool to read Apache Delta Lake files. Delta Lake is an open-source storage layer that brings ACID (atomicity, consistency, isolation, and durability) transactions to Apache Spark and big data workloads.

The serverless SQL pool in Synapse workspace enables you to read the data stored in Delta Lake format, and serve it to reporting tools. A serverless SQL pool can read Delta Lake files that are created using Apache Spark, Azure Databricks, or any other producer of the Delta Lake format.

Apache Spark pools in Azure Synapse enable data engineers to modify Delta Lake files using Scala, PySpark, and .NET. Serverless SQL pools help data analysts to create reports on Delta Lake files created by data engineers.

Quickstart example

The OPENROWSET function enables you to read the content of Delta Lake files by providing the URL to your root folder.

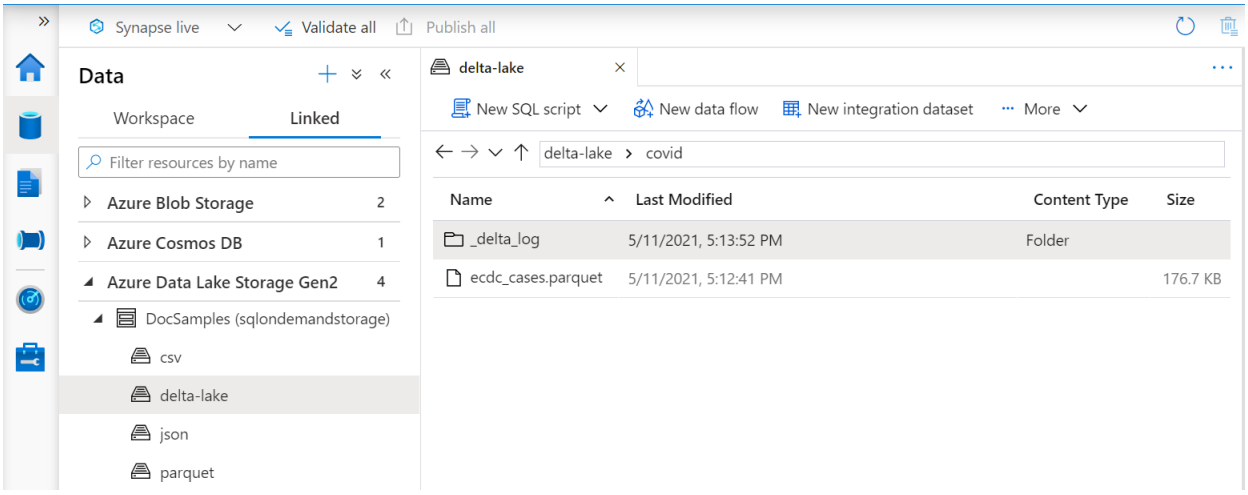
Read Delta Lake folder

The easiest way to see to the content of your `DELTA` file is to provide the file URL to the OPENROWSET function and specify `DELTA` format. If the file is publicly available or if your Azure AD identity can access this file, you should be able to see the content of the file using a query like the one shown in the following example:

```
SELECT TOP 10 *
FROM OPENROWSET(
    BULK 'https://sqlondemandstorage.blob.core.windows.net/delta-lake/covid/',
    FORMAT = 'delta') as rows;
```

Column names and data types are automatically read from Delta Lake files. The `OPENROWSET` function uses best guess types like VARCHAR(1000) for the string columns.

The URI in the `OPENROWSET` function must reference the root Delta Lake folder that contains a subfolder called `_delta_log`.



If you don't have this subfolder, you are not using Delta Lake format. You can convert your plain Parquet files in

the folder to Delta Lake format using the following Apache Spark Python script:

```
%%pyspark
from delta.tables import *
deltaTable = DeltaTable.convertToDelta(spark, "parquet.`abfss://delta-
lake@sqlondemandstorage.dfs.core.windows.net/covid`")
```

To improve the performance of your queries, consider specifying explicit types in the `WITH` clause.

> **NOTE**
> The serverless Synapse SQL pool uses schema inference to automatically determine columns and their types. The rules for schema inference are the same used for Parquet files. For Delta Lake type mapping to SQL native type check type mapping for Parquet.

Make sure you can access your file. If your file is protected with SAS key or custom Azure identity, you will need to set up a server level credential for sql login.

> **IMPORTANT**
> Ensure you are using a UTF-8 database collation (for example `Latin1_General_100_BIN2_UTF8`) because string values in Delta Lake files are encoded using UTF-8 encoding. A mismatch between the text encoding in the Delta Lake file and the collation may cause unexpected conversion errors. You can easily change the default collation of the current database using the following T-SQL statement: `alter database current collate Latin1_General_100_BIN2_UTF8`

Data source usage

The previous examples used the full path to the file. As an alternative, you can create an external data source with the location that points to the root folder of the storage. Once you've created the external data source, use the data source and the relative path to the file in the `OPENROWSET` function. This way you don't need to use the full absolute URI to your files. You can also then define custom credentials to access the storage location.

> **IMPORTANT**
> Data sources can be created only in custom databases (not in the master database or the databases replicated from Apache Spark pools).

To use the samples below, you will need to complete the following step:

1. **Create a database** with a datasource that references NYC Yellow Taxi storage account.
2. Initialize the objects by executing setup script on the database you created in step 1. This setup script will create the data sources, database scoped credentials, and external file formats that are used in these samples.

If you created your database, and switched the context to your database (using `USE database_name` statement or dropdown for selecting database in some query editor), you can create your external data source containing the root URI to your data set and use it to query Delta Lake files:

```
CREATE EXTERNAL DATA SOURCE DeltaLakeStorage
WITH ( LOCATION = 'https://sqlondemandstorage.blob.core.windows.net/delta-lake/' );
GO

SELECT TOP 10 *
FROM OPENROWSET(
        BULK 'covid',
        DATA_SOURCE = 'DeltaLakeStorage',
        FORMAT = 'delta'
    ) as rows;
```

If a data source is protected with SAS key or custom identity, you can configure data source with database scoped credential.

Explicitly specify schema

`OPENROWSET` enables you to explicitly specify what columns you want to read from the file using `WITH` clause:

```
SELECT TOP 10 *
FROM OPENROWSET(
        BULK 'covid',
        DATA_SOURCE = 'DeltaLakeStorage',
        FORMAT = 'delta'
    )
    WITH ( date_rep date,
           cases int,
           geo_id varchar(6)
           ) as rows;
```

With the explicit specification of the result set schema, you can minimize the type sizes and use the more precise types VARCHAR(6) for string columns instead of pessimistic VARCHAR(1000). Minimization of types might significantly improve performance of your queries.

> **IMPORTANT**
> Make sure that you are explicitly specifying a UTF-8 collation (for example `Latin1_General_100_BIN2_UTF8`) for all string columns in `WITH` clause or set a UTF-8 collation at the database level. Mismatch between text encoding in the file and string column collation might cause unexpected conversion errors. You can easily change default collation of the current database using the following T-SQL statement: `alter database current collate Latin1_General_100_BIN2_UTF8` You can easily set collation on the colum types using the following definition:
> `geo_id varchar(6) collate Latin1_General_100_BIN2_UTF8`

Dataset

NYC Yellow Taxi dataset is used in this sample. You can query Parquet files the same way you read CSV files. The only difference is that the `FILEFORMAT` parameter should be set to `PARQUET`. Examples in this article show the specifics of reading Parquet files.

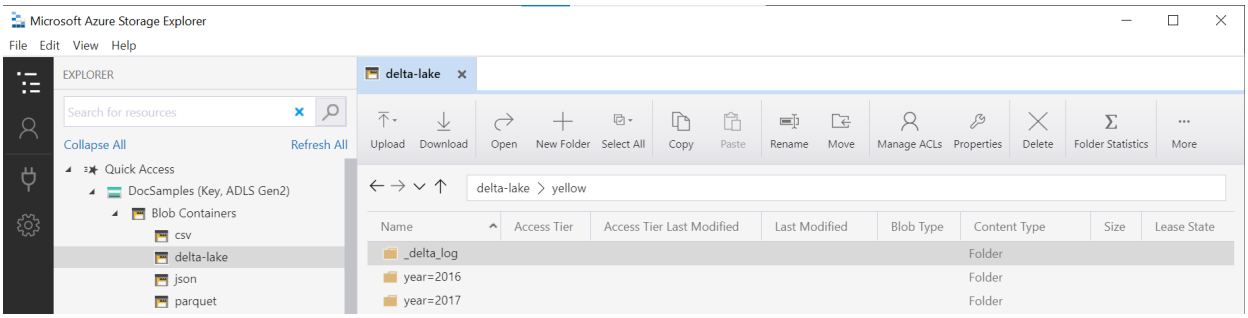
Query partitioned data

The data set provided in this sample is divided (partitioned) into separate subfolders. Unlike Parquet, you don't need to target specific partitions using the `FILEPATH` function. The `OPENROWSET` will identify partitioning columns in your Delta Lake folder structure and enable you to directly query data using these columns. This example shows fare amounts by year, month, and payment_type for the first three months of 2017.

```
SELECT
        YEAR(pickup_datetime) AS year,
        passenger_count,
        COUNT(*) AS cnt
FROM
    OPENROWSET(
        BULK 'yellow',
        DATA_SOURCE = 'DeltaLakeStorage',
        FORMAT='DELTA'
    ) nyc
WHERE
    nyc.year = 2017
    AND nyc.month IN (1, 2, 3)
    AND pickup_datetime BETWEEN CAST('1/1/2017' AS datetime) AND CAST('3/31/2017' AS datetime)
GROUP BY
    passenger_count,
    YEAR(pickup_datetime)
ORDER BY
    YEAR(pickup_datetime),
    passenger_count;
```

The `OPENROWSET` function will eliminate partitions that don't match the `year` and `month` in the where clause. This file/partition pruning technique will significantly reduce your data set, improve performance, and reduce the cost of the query.

The folder name in the `OPENROWSET` function (`yellow` in this example) is concatenated using the `LOCATION` in `DeltaLakeStorage` data source, and must reference the root Delta Lake folder that contains a subfolder called `_delta_log`.



If you don't have this subfolder, you are not using Delta Lake format. You can convert your plain Parquet files in the folder to Delta Lake format using the following Apache Spark Python script:

```
%%pyspark
from delta.tables import *
deltaTable = DeltaTable.convertToDelta(spark, "parquet.`abfss://delta-
lake@sqlondemandstorage.dfs.core.windows.net/yellow`", "year INT, month INT")
```

The second argument of `DeltaTable.convertToDeltaLake` function represents the partitioning columns (year and month) that are a part of folder pattern (`year=*/month=*` in this example) and their types.

Limitations

- Review the limitations and the known issues on Synapse serverless SQL pool self-help page.
- Currently, both the Spark pool and serverless SQL pool in Azure Synapse Analytics support Delta Lake format. Serverless SQL pools do not support updating Delta Lake files. Only tables in Parquet format are shared from Spark pools to a serverless SQL pool. For more information, see Shared Spark tables.

Next steps

Advance to the next article to learn how to Query Parquet nested types. If you want to continue building Delta Lake solution, learn how to create views or external tables on the Delta Lake folder.

See also

- What is Delta Lake
- Learn how to use Delta Lake in Apache Spark pools for Azure Synapse Analytics
- Azure Databricks Delta Lake best practices
- Delta Lake Documentation Page
- Know issues and limitations

Query folders and multiple files

8/19/2022 • 6 minutes to read • Edit Online

In this article, you'll learn how to write a query using serverless SQL pool in Azure Synapse Analytics.

Serverless SQL pool supports reading multiple files/folders by using wildcards, which are similar to the wildcards used in Windows OS. However, greater flexibility is present since multiple wildcards are allowed.

Prerequisites

Your first step is to **create a database** where you'll execute the queries. Then initialize the objects by executing setup script on that database. This setup script will create the data sources, database scoped credentials, and external file formats that are used in these samples.

You'll use the folder *csv/taxi* to follow the sample queries. It contains NYC Taxi - Yellow Taxi Trip Records data from July 2016 to June 2018. Files in *csv/taxi* are named after year and month using the following pattern: yellow_tripdata_<year>-<month>.csv

Read all files in folder

The example below reads all NYC Yellow Taxi data files from the *csv/taxi* folder and returns the total number of passengers and rides per year. It also shows usage of aggregate functions.

```
SELECT
    YEAR(pickup_datetime) as [year],
    SUM(passenger_count) AS passengers_total,
    COUNT(*) AS [rides_total]
FROM OPENROWSET(
        BULK 'csv/taxi/*.csv',
        DATA_SOURCE = 'sqlondemanddemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIRSTROW = 2
    )
    WITH (
        pickup_datetime DATETIME2 2,
        passenger_count INT 4
    ) AS nyc
GROUP BY
    YEAR(pickup_datetime)
ORDER BY
    YEAR(pickup_datetime);
```

NOTE

All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Read subset of files in folder

The example below reads the 2017 NYC Yellow Taxi data files from the *csv/taxi* folder using a wildcard and returns the total fare amount per payment type.

```
SELECT
    payment_type,
    SUM(fare_amount) AS fare_total
FROM OPENROWSET(
        BULK 'csv/taxi/yellow_tripdata_2017-*.csv',
        DATA_SOURCE = 'sqlondemanddemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIRSTROW = 2
    )
    WITH (
        payment_type INT 10,
        fare_amount FLOAT 11
    ) AS nyc
GROUP BY payment_type
ORDER BY payment_type;
```

NOTE

All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Read subset of files in folder using multiple file paths

The example below reads the 2017 NYC Yellow Taxi data files from the *csv/taxi* folder using 2 file paths first one with full path to the file containing data from month January and second with a wildcard reading months November and December which returns the total fare amount per payment type.

```
SELECT
    payment_type,
    SUM(fare_amount) AS fare_total
FROM OPENROWSET(
        BULK (
            'csv/taxi/yellow_tripdata_2017-01.csv',
            'csv/taxi/yellow_tripdata_2017-1*.csv'
        ),
        DATA_SOURCE = 'sqlondemanddemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIRSTROW = 2
    )
    WITH (
        payment_type INT 10,
        fare_amount FLOAT 11
    ) AS nyc
GROUP BY payment_type
ORDER BY payment_type;
```

NOTE

All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Read folders

The path that you provide to OPENROWSET can also be a path to a folder. The following sections include these query types.

Read all files from specific folder

You can read all the files in a folder using the file level wildcard as shown in Read all files in folder. But, there's a

way to query a folder and consume all files within that folder.

If the path provided in OPENROWSET points to a folder, all files in that folder will be used as a source for your query. The following query will read all files in the *csv/taxi* folder.

> **NOTE**
> Note the existence of the / at the end of the path in the query below. It denotes a folder. If the / is omitted, the query will target a file named *taxi* instead.

```
SELECT
    YEAR(pickup_datetime) as [year],
    SUM(passenger_count) AS passengers_total,
    COUNT(*) AS [rides_total]
FROM OPENROWSET(
        BULK 'csv/taxi/',
        DATA_SOURCE = 'sqlondemanddemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIRSTROW = 2
    )
    WITH (
        vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count INT,
        trip_distance FLOAT,
        rate_code INT,
        store_and_fwd_flag VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_location_id INT,
        dropoff_location_id INT,
        payment_type INT,
        fare_amount FLOAT,
        extra FLOAT,
        mta_tax FLOAT,
        tip_amount FLOAT,
        tolls_amount FLOAT,
        improvement_surcharge FLOAT,
        total_amount FLOAT
    ) AS nyc
GROUP BY
    YEAR(pickup_datetime)
ORDER BY
    YEAR(pickup_datetime);
```

> **NOTE**
> All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Read all files from multiple folders

It's possible to read files from multiple folders by using a wildcard. The following query will read all files from all folders located in the *csv* folder that have names starting with *t* and ending with *i*.

> **NOTE**
> Note the existence of the / at the end of the path in the query below. It denotes a folder. If the / is omitted, the query will target files named *t*i* instead.

```
SELECT
    YEAR(pickup_datetime) as [year],
    SUM(passenger_count) AS passengers_total,
    COUNT(*) AS [rides_total]
FROM OPENROWSET(
        BULK 'csv/t*i/',
        DATA_SOURCE = 'sqlondemanddemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIRSTROW = 2
    )
    WITH (
        vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count INT,
        trip_distance FLOAT,
        rate_code INT,
        store_and_fwd_flag VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_location_id INT,
        dropoff_location_id INT,
        payment_type INT,
        fare_amount FLOAT,
        extra FLOAT,
        mta_tax FLOAT,
        tip_amount FLOAT,
        tolls_amount FLOAT,
        improvement_surcharge FLOAT,
        total_amount FLOAT
    ) AS nyc
GROUP BY
    YEAR(pickup_datetime)
ORDER BY
    YEAR(pickup_datetime);
```

NOTE

All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Since you have only one folder that matches the criteria, the query result is the same as Read all files in folder.

Traverse folders recursively

Serverless SQL pool can recursively traverse folders if you specify /** at the end of path. The following query will read all files from all folders and subfolders located in the *csv/taxi* folder.

```
SELECT
    YEAR(pickup_datetime) as [year],
    SUM(passenger_count) AS passengers_total,
    COUNT(*) AS [rides_total]
FROM OPENROWSET(
        BULK 'csv/taxi/**',
        DATA_SOURCE = 'sqlondemanddemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIRSTROW = 2
    )
    WITH (
        vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count INT,
        trip_distance FLOAT,
        rate_code INT,
        store_and_fwd_flag VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_location_id INT,
        dropoff_location_id INT,
        payment_type INT,
        fare_amount FLOAT,
        extra FLOAT,
        mta_tax FLOAT,
        tip_amount FLOAT,
        tolls_amount FLOAT,
        improvement_surcharge FLOAT,
        total_amount FLOAT
    ) AS nyc
GROUP BY
    YEAR(pickup_datetime)
ORDER BY
    YEAR(pickup_datetime);
```

NOTE

All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Multiple wildcards

You can use multiple wildcards on different path levels. For example, you can enrich previous query to read files with 2017 data only, from all folders which names start with *t* and end with *i*.

NOTE

Note the existence of the / at the end of the path in the query below. It denotes a folder. If the / is omitted, the query will target files named *t***i* instead. There is a maximum limit of 10 wildcards per query.

```
SELECT
    YEAR(pickup_datetime) as [year],
    SUM(passenger_count) AS passengers_total,
    COUNT(*) AS [rides_total]
FROM OPENROWSET(
        BULK 'csv/t*i/yellow_tripdata_2017-*.csv',
        DATA_SOURCE = 'sqlondemanddemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIRSTROW = 2
    )
    WITH (
        vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count INT,
        trip_distance FLOAT,
        rate_code INT,
        store_and_fwd_flag VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_location_id INT,
        dropoff_location_id INT,
        payment_type INT,
        fare_amount FLOAT,
        extra FLOAT,
        mta_tax FLOAT,
        tip_amount FLOAT,
        tolls_amount FLOAT,
        improvement_surcharge FLOAT,
        total_amount FLOAT
    ) AS nyc
GROUP BY
    YEAR(pickup_datetime)
ORDER BY
    YEAR(pickup_datetime);
```

NOTE

All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Since you have only one folder that matches the criteria, the query result is the same as Read subset of files in folder and Read all files from specific folder. More complex wildcard usage scenarios are covered in Query Parquet files.

Next steps

More information can be found in the Query specific files article.

Use file metadata in serverless SQL pool queries

8/19/2022 • 2 minutes to read • Edit Online

Serverless SQL pool can address multiple files and folders as described in the Query folders and multiple files article. In this article, you learn how to use metadata information about file and folder names in the queries.

Sometimes, you may need to know which file or folder source correlates to a specific row in the result set.

You can use function `filepath` and `filename` to return file names and/or the path in the result set. Or you can use them to filter data based on the file name and/or folder path. These functions are described in the syntax section filename function and filepath function. In the following sections, you'll find short descriptions along samples.

Prerequisites

Your first step is to **create a database** with a datasource that references storage account. Then initialize the objects by executing setup script on that database. This setup script will create the data sources, database scoped credentials, and external file formats that are used in these samples.

Functions

Filename

This function returns the file name that row originates from.

The following sample reads the NYC Yellow Taxi data files for the last three months of 2017 and returns the number of rides per file. The OPENROWSET part of the query specifies which files will be read.

```
SELECT
    nyc.filename() AS [filename]
    ,COUNT_BIG(*) AS [rows]
FROM
    OPENROWSET(
        BULK 'parquet/taxi/year=2017/month=9/*.parquet',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT='PARQUET'
    ) nyc
GROUP BY nyc.filename();
```

The following example shows how *filename()* can be used in the WHERE clause to filter the files to be read. It accesses the entire folder in the OPENROWSET part of the query and filters files in the WHERE clause.

Your results will be the same as the prior example.

```
SELECT
    r.filename() AS [filename]
    ,COUNT_BIG(*) AS [rows]
FROM OPENROWSET(
    BULK 'csv/taxi/yellow_tripdata_2017-*.csv',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV',
        PARSER_VERSION = '2.0',
        FIRSTROW = 2)
        WITH (C1 varchar(200) ) AS [r]
WHERE
    r.filename() IN ('yellow_tripdata_2017-10.csv', 'yellow_tripdata_2017-11.csv', 'yellow_tripdata_2017-
12.csv')
GROUP BY
    r.filename()
ORDER BY
    [filename];
```

Filepath

The filepath function returns a full or partial path:

- When called without a parameter, it returns the full file path that the row originates from. When DATA_SOURCE is used in OPENROWSET, it returns path relative to DATA_SOURCE.
- When called with a parameter, it returns part of the path that matches the wildcard on the position specified in the parameter. For example, parameter value 1 would return part of the path that matches the first wildcard.

The following sample reads NYC Yellow Taxi data files for the last three months of 2017. It returns the number of rides per file path. The OPENROWSET part of the query specifies which files will be read.

```
SELECT
    r.filepath() AS filepath
    ,COUNT_BIG(*) AS [rows]
FROM OPENROWSET(
        BULK 'csv/taxi/yellow_tripdata_2017-1*.csv',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV',
        PARSER_VERSION = '2.0',
        FIRSTROW = 2
    )
    WITH (
        vendor_id INT
    ) AS [r]
GROUP BY
    r.filepath()
ORDER BY
    filepath;
```

The following example shows how *filepath()* can be used in the WHERE clause to filter the files to be read.

You can use the wildcards in the OPENROWSET part of the query and filter the files in the WHERE clause. Your results will be the same as the prior example.

```
SELECT
    r.filepath() AS filepath
    ,r.filepath(1) AS [year]
    ,r.filepath(2) AS [month]
    ,COUNT_BIG(*) AS [rows]
FROM OPENROWSET(
        BULK 'csv/taxi/yellow_tripdata_*-*.csv',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV',
        PARSER_VERSION = '2.0',
        FIRSTROW = 2
    )
WITH (
    vendor_id INT
) AS [r]
WHERE
    r.filepath(1) IN ('2017')
    AND r.filepath(2) IN ('10', '11', '12')
GROUP BY
    r.filepath()
    ,r.filepath(1)
    ,r.filepath(2)
ORDER BY
    filepath;
```

Next steps

In the next article, you'll learn how to query Parquet files.

Create and use views using serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

In this section, you'll learn how to create and use views to wrap serverless SQL pool queries. Views will allow you to reuse those queries. Views are also needed if you want to use tools, such as Power BI, in conjunction with serverless SQL pool.

Prerequisites

Your first step is to create a database where the view will be created and initialize the objects needed to authenticate on Azure storage by executing setup script on that database. All queries in this article will be executed on your sample database.

Views over external data

You can create views the same way you create regular SQL Server views. The following query creates view that reads *population.csv* file.

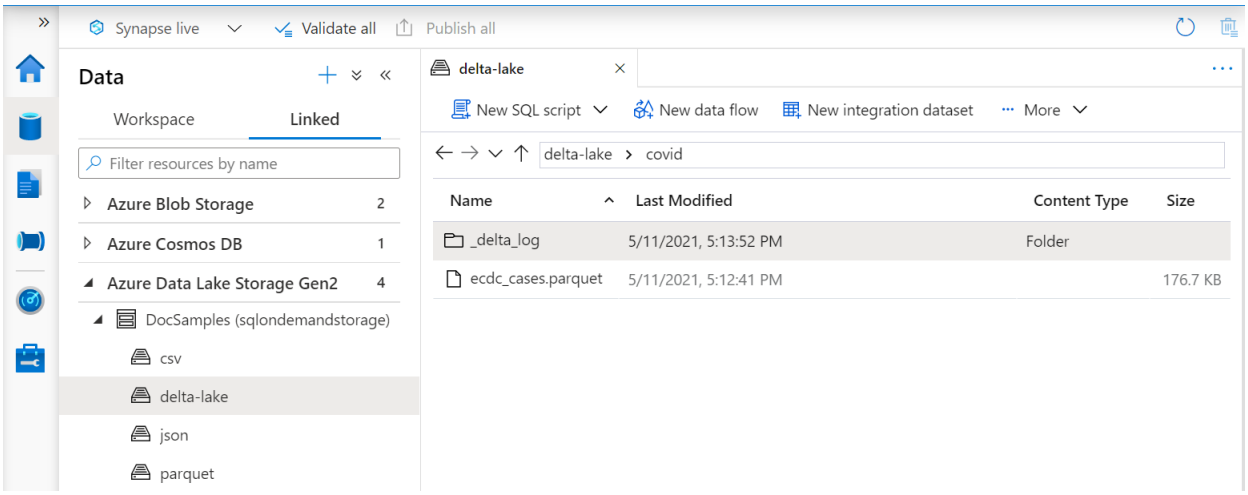
> **NOTE**
> Change the first line in the query, i.e., [mydbname], so you're using the database you created.

```
USE [mydbname];
GO

DROP VIEW IF EXISTS populationView;
GO

CREATE VIEW populationView AS
SELECT *
FROM OPENROWSET(
        BULK 'csv/population/population.csv',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '\n'
    )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
) AS [r];
```

The view uses an `EXTERNAL DATA SOURCE` with a root URL of your storage, as a `DATA_SOURCE` and adds a relative file path to the files.

Delta Lake views

If you are creating the views on top of Delta Lake folder, you need to specify the location to the root folder after the `BULK` option instead of specifying the file path.



The `OPENROWSET` function that reads data from the Delta Lake folder will examine the folder structure and automatically identify the file locations.

```
create or alter view CovidDeltaLake
as
select *
from openrowset(
           bulk 'covid',
           data_source = 'DeltaLakeStorage',
           format = 'delta'
    ) with (
           date_rep date,
           cases int,
           geo_id varchar(6)
           ) as rows
```

Review the known issues on Synapse serverless SQL pool self-help page.

Partitioned views

If you have a set of files that is partitioned in the hierarchical folder structure, you can describe the partition pattern using the wildcards in the file path. Use the `FILEPATH` function to expose parts of the folder path as partitioning columns.

```
CREATE VIEW TaxiView
AS SELECT *, nyc.filepath(1) AS [year], nyc.filepath(2) AS [month]
FROM
    OPENROWSET(
        BULK 'parquet/taxi/year=*/month=*/*.parquet',
        DATA_SOURCE = 'sqlondemanddemo',
        FORMAT='PARQUET'
    ) AS nyc
```

The partitioned views will perform folder partition elimination if you query this view with the filters on the partitioning columns. This might improve performance of your queries.

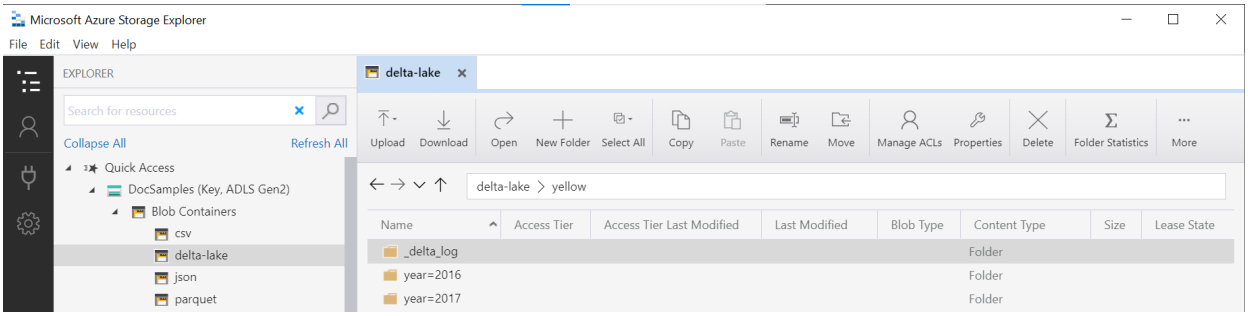
Delta Lake partitioned views

If you are creating the partitioned views on top of Delta Lake storage, you can specify just a root Delta Lake folder and don't need to explicitly expose the partitioning columns using the `FILEPATH` function:

```
CREATE OR ALTER VIEW YellowTaxiView
AS SELECT *
FROM
    OPENROWSET(
        BULK 'yellow',
        DATA_SOURCE = 'DeltaLakeStorage',
        FORMAT='DELTA'
    ) nyc
```

The `OPENROWSET` function will examine the structure of the underlying Delta Lake folder and automatically identify and expose the partitioning columns. The partition elimination will be done automatically if you put the partitioning column in the `WHERE` clause of a query.

The folder name in the `OPENROWSET` function (`yellow` in this example) that is concatenated with the `LOCATION` URI defined in `DeltaLakeStorage` data source must reference the root Delta Lake folder that contains a subfolder called `_delta_log`.



Review the known issues on Synapse serverless SQL pool self-help page.

JSON views

The views are the good choice if you need to do some additional processing on top of the result set that is fetched from the files. One example might be parsing JSON files where we need to apply the JSON functions to extract the values from the JSON documents:

```
CREATE OR ALTER VIEW CovidCases
AS
select
    *
from openrowset(
        bulk 'latest/ecdc_cases.jsonl',
        data_source = 'covid',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
    cross apply openjson (doc)
        with (  date_rep datetime2,
                cases int,
                fatal int '$.deaths',
                country varchar(100) '$.countries_and_territories')
```

The `OPENJSON` function parses each line from the JSONL file containing one JSON document per line in textual format.

Cosmos DB views on containers

The views can be created on top of the Azure Cosmos DB containers if the Cosmos DB analytical storage is enabled on the container. Cosmos DB account name, database name, and container name should be added as a

part of the view, and the read-only access key should be placed in the database scoped credential that the view references.

```
CREATE DATABASE SCOPED CREDENTIAL MyCosmosDbAccountCredential
WITH IDENTITY = 'SHARED ACCESS SIGNATURE', SECRET =
's5zarR2pT0JWH9k8roipnWxUYBegOuFGjJpSjGlR36y86cW0GQ6RaaG8kGjsRAQoWMw1QKTkkX8HQtFpJjC8Hg==';
GO
CREATE OR ALTER VIEW Ecdc
AS SELECT *
FROM OPENROWSET(
      PROVIDER = 'CosmosDB',
      CONNECTION = 'Account=synapselink-cosmosdb-sqlsample;Database=covid',
      OBJECT = 'Ecdc',
      CREDENTIAL = 'MyCosmosDbAccountCredential'
    ) with ( date_rep varchar(20), cases bigint, geo_id varchar(6) ) as rows
```

For more information, see Query Azure Cosmos DB data with a serverless SQL pool in Azure Synapse Link.

Use a view

You can use views in your queries the same way you use views in SQL Server queries.

The following query demonstrates using the *population_csv* view we created in Create a view. It returns country/region names with their population in 2019 in descending order.

> **NOTE**
> Change the first line in the query, i.e., [mydbname], so you're using the database you created.

```
USE [mydbname];
GO

SELECT
    country_name, population
FROM populationView
WHERE
    [year] = 2019
ORDER BY
    [population] DESC;
```

Next steps

For information on how to query different file types, refer to the Query single CSV file, Query Parquet files, and Query JSON files articles.

Create and use native external tables using SQL pools in Azure Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

In this section, you'll learn how to create and use native external tables in Synapse SQL pools. Native external tables have better performance when compared to external tables with `TYPE=HADOOP` in their external data source definition. This is because native external tables use native code to access external data.

External tables are useful when you want to control access to external data in Synapse SQL pool. External tables are also useful if you want to use tools, such as Power BI, in conjunction with Synapse SQL pool. External tables can access two types of storage:

- Public storage where users access public storage files.
- Protected storage where users access storage files using SAS credential, Azure AD identity, or Managed Identity of Synapse workspace.

> **NOTE**
> In dedicated SQL pools you can only use native external tables with a Parquet file type, and this feature is in **public preview**. If you want to use generally available Parquet reader functionality in dedicated SQL pools, or you need to access CSV or ORC files, use Hadoop external tables. Native external tables are generally available in serverless SQL pools. Learn more about the differences between native and Hadoop external tables in Use external tables with Synapse SQL.

The following table lists the data formats supported:

DATA FORMAT (NATIVE EXTERNAL TABLES)	SERVERLESS SQL POOL	DEDICATED SQL POOL
Parquet	Yes (GA)	Yes (public preview)
CSV	Yes	No (Alternatively, use Hadoop external tables)
delta	Yes	No
Spark	Yes	No
Dataverse	Yes	No
Azure Cosmos DB data formats (JSON, BSON etc.)	No (Alternatively, create views)	No

Prerequisites

Your first step is to create a database where the tables will be created. Then create the following objects that are used in this sample:

- DATABASE SCOPED CREDENTIAL `sqlondemand` that enables access to SAS-protected `https://sqlondemandstorage.blob.core.windows.net` Azure storage account.

```
CREATE DATABASE SCOPED CREDENTIAL [sqlondemand]
WITH IDENTITY='SHARED ACCESS SIGNATURE',
SECRET = 'sv=2018-03-28&ss=bf&srt=sco&sp=rl&st=2019-10-14T12%3A10%3A25Z&se=2061-12-
31T12%3A10%3A00Z&sig=KlSU2ullCscyTS0An0nozEpo4tO5JAgGBvw%2FJX2lguw%3D'
```

- EXTERNAL DATA SOURCE `sqlondemanddemo` that references demo storage account protected with SAS key, and EXTERNAL DATA SOURCE `nyctlc` that references publicly available Azure storage account on location `https://azureopendatastorage.blob.core.windows.net/nyctlc/`.

```
CREATE EXTERNAL DATA SOURCE SqlOnDemandDemo WITH (
    LOCATION = 'https://sqlondemandstorage.blob.core.windows.net',
    CREDENTIAL = sqlondemand
);
GO
CREATE EXTERNAL DATA SOURCE nyctlc
WITH ( LOCATION = 'https://azureopendatastorage.blob.core.windows.net/nyctlc/')
GO
CREATE EXTERNAL DATA SOURCE DeltaLakeStorage
WITH ( location = 'https://sqlondemandstorage.blob.core.windows.net/delta-lake/' );
```

- File formats `QuotedCSVWithHeaderFormat` and `ParquetFormat` that describe CSV and parquet file types.

```
CREATE EXTERNAL FILE FORMAT QuotedCsvWithHeaderFormat
WITH (
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS ( FIELD_TERMINATOR = ',', STRING_DELIMITER = '"', FIRST_ROW = 2   )
);
GO
CREATE EXTERNAL FILE FORMAT ParquetFormat WITH (  FORMAT_TYPE = PARQUET );
GO
CREATE EXTERNAL FILE FORMAT DeltaLakeFormat WITH (  FORMAT_TYPE = DELTA );
GO
```

The queries in this article will be executed on your sample database and use these objects.

External table on a file

You can create external tables that access data on an Azure storage account that allows access to users with some Azure AD identity or SAS key. You can create external tables the same way you create regular SQL Server external tables.

The following query creates an external table that reads *population.csv* file from SynapseSQL demo Azure storage account that is referenced using `sqlondemanddemo` data source and protected with database scoped credential called `sqlondemand`.

Data source and database scoped credential are created in setup script.

> **NOTE**
>
> Change the first line in the query, i.e., [mydbname], so you're using the database you created.

```
USE [mydbname];
GO
CREATE EXTERNAL TABLE populationExternalTable
(
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
)
WITH (
    LOCATION = 'csv/population/population.csv',
    DATA_SOURCE = sqlondemanddemo,
    FILE_FORMAT = QuotedCSVWithHeaderFormat
);
```

Native CSV tables are currently available only in the serverless SQL pools.

External table on a set of files

You can create external tables that read data from a set of files placed on Azure storage:

```
CREATE EXTERNAL TABLE Taxi (
     vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
     pickup_datetime DATETIME2,
     dropoff_datetime DATETIME2,
     passenger_count INT,
     trip_distance FLOAT,
     fare_amount FLOAT,
     tip_amount FLOAT,
     tolls_amount FLOAT,
     total_amount FLOAT
) WITH (
         LOCATION = 'yellow/puYear=*/puMonth=*/*.parquet',
         DATA_SOURCE = nyctlc,
         FILE_FORMAT = ParquetFormat
);
```

You can specify the pattern that the files must satisfy in order to be referenced by the external table. The pattern is required only for Parquet and CSV tables. If you are using Delta Lake format, you need to specify just a root folder, and the external table will automatically find the pattern.

> **NOTE**
>
> The table is created on partitioned folder structure, but you cannot leverage some partition elimination. If you want to get better performance by skipping the files that do not satisfy some criterion (like specific year or month in this case), use views on external data.

External table on appendable files

The files that are referenced by an external table should not be changed while the query is running. In the long-running query, SQL pool may retry reads, read parts of the files, or even read the file multiple times. Changes of the file content would cause wrong results. Therefore, the SQL pool fails the query if detects that the modification time of any file is changed during the query execution. In some scenarios you might want to create a table on the files that are constantly appended. To avoid the query failures due to constantly appended files, you can specify that the external table should ignore potentially inconsistent reads using the `TABLE_OPTIONS` setting.

```
CREATE EXTERNAL TABLE populationExternalTable
(
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
)
WITH (
    LOCATION = 'csv/population/population.csv',
    DATA_SOURCE = sqlondemanddemo,
    FILE_FORMAT = QuotedCSVWithHeaderFormat,
    TABLE_OPTIONS = N'{"READ_OPTIONS":["ALLOW_INCONSISTENT_READS"]}'
);
```

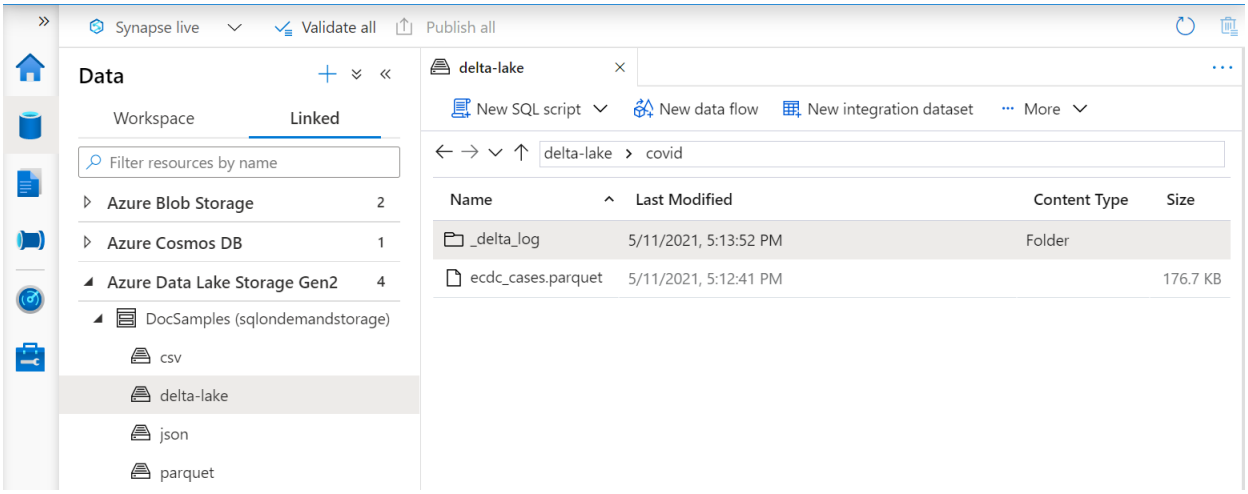
The `ALLOW_INCONSISTENT_READS` read option will disable file modification time check during the query lifecycle and read whatever is available in the files that are referenced by the external table. In appendable files, the existing content is not updated, and only new rows are added. Therefore, the probability of wrong results is minimized compared to the updateable files. This option might enable you to read the frequently appended files without handling the errors.

This option is available only in the external tables created on CSV file format.

> **NOTE**
>
> As the option name implies, the creator of the table accepts a risk that the results might not be consistent. In the appendable files, you might get incorrect results if you force multiple read of the underlying files by self-joining the table. In most of the "classic" queries, the external table will just ignore some rows that are appended while the query was running.

Delta Lake external table

External tables can be created on top of a Delta Lake folder. The only difference between the external tables created on a single file or a file set and the external tables created on a Delta Lake format is that in Delta Lake external table you need to reference a folder containing the Delta Lake structure.



An example of a table definition created on a Delta Lake folder is:

```
CREATE EXTERNAL TABLE Covid (
     date_rep date,
     cases int,
     geo_id varchar(6)
) WITH (
        LOCATION = 'covid', --> the root folder containing the Delta Lake files
        data_source = DeltaLakeStorage,
        FILE_FORMAT = DeltaLakeFormat
);
```

External tables cannot be created on a partitioned folder. Review the other known issues on Synapse serverless SQL pool self-help page.

Delta tables on partitioned folders

External tables in serverless SQL pools do not support partitioning on Delta Lake format. Use Delta partitioned views instead of tables if you have partitioned Delta Lake data sets.

> **IMPORTANT**
> Do not create external tables on partitioned Delta Lake folders even if you see that they might work in some cases. Using unsupported features like external tables on partitioned delta folders might cause issues or instability of the serverless pool. Azure support will not be able to resolve any issue if it is using tables on partitioned folders. You would be asked to transition to Delta partitioned views and rewrite your code to use only the supported feature before proceeding with issue resolution.

Use an external table

You can use external tables in your queries the same way you use them in SQL Server queries.

The following query demonstrates this using the *population* external table we created in previous section. It returns country/region names with their population in 2019 in descending order.

> **NOTE**
> Change the first line in the query, i.e., [mydbname], so you're using the database you created.

```
USE [mydbname];
GO

SELECT
    country_name, population
FROM populationExternalTable
WHERE
    [year] = 2019
ORDER BY
    [population] DESC;
```

Performance of this query might vary depending on region. Your workspace might not be placed in the same region as the Azure storage accounts used in these samples. For production workloads, place your Synapse workspace and Azure storage in the same region.

Next steps

For information on how to store results of a query to storage, refer to Store query results to the storage article.

Store query results to storage using serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

In this article, you'll learn how to store query results to storage using serverless SQL pool.

Prerequisites

Your first step is to **create a database** where you will execute the queries. Then initialize the objects by executing setup script on that database. This setup script will create the data sources, database scoped credentials, and external file formats that are used to read data in these samples.

Follow the instructions in this article to create data sources, database scoped credentials, and external file formats that are used to write data into the output storage.

Create external table as select

You can use the CREATE EXTERNAL TABLE AS SELECT (CETAS) statement to store the query results to storage.

> **NOTE**
> Change the first line in the query, i.e., [mydbname], so you're using the database you created.

```
USE [mydbname];
GO

CREATE DATABASE SCOPED CREDENTIAL [SasTokenWrite]
WITH IDENTITY = 'SHARED ACCESS SIGNATURE',
     SECRET = 'sv=2018-03-28&ss=bfqt&srt=sco&sp=rwdlacup&se=2019-04-18T20:42:12Z&st=2019-04-
18T12:42:12Z&spr=https&sig=lQHczNvrk1KoYLCpFdSsMANd0ef9BrIPBNJ3VYEIq78%3D';
GO

CREATE EXTERNAL DATA SOURCE [MyDataSource] WITH (
    LOCATION = 'https://<storage account name>.blob.core.windows.net/csv', CREDENTIAL = [SasTokenWrite]
);
GO

CREATE EXTERNAL FILE FORMAT [ParquetFF] WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
);
GO

CREATE EXTERNAL TABLE [dbo].[PopulationCETAS] WITH (
        LOCATION = 'populationParquet/',
        DATA_SOURCE = [MyDataSource],
        FILE_FORMAT = [ParquetFF]
) AS
SELECT
    *
FROM
    OPENROWSET(
        BULK 'csv/population-unix/population.csv',
        DATA_SOURCE = 'sqlondemanddemo',
        FORMAT = 'CSV', PARSER_VERSION = '2.0'
    ) WITH (
        CountryCode varchar(4),
        CountryName varchar(64),
        Year int,
        PopulationCount int
    ) AS r;
```

NOTE

You must modify this script and change the target location to execute it again. External tables cannot be created on the location where you already have some data.

Use the external table

You can use the external table created through CETAS like a regular external table.

NOTE

Change the first line in the query, i.e., [mydbname], so you're using the database you created.

```
USE [mydbname];
GO

SELECT
    country_name, population
FROM PopulationCETAS
WHERE
    [year] = 2019
ORDER BY
    [population] DESC;
```

Remarks

Once you store your results, the data in the external table cannot be modified. You cannot repeat this script because CETAS will not overwrite the underlying data created in the previous execution. Vote for the following feedback items if some of these are required in your scenarios, or propose the new ones on Azure feedback site:

- Enable inserting new data into external table
- Enable deleting data from external table
- Specify partitions in CETAS
- Specify file sizes and counts

The only supported output types are Parquet and CSV. You can vote for the other types on Azure feedback site.

Next steps

For more information on how to query different file types, see the Query single CSV file, Query Parquet files, and Query JSON files articles.

Analyze complex data types in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

This article is relevant for Parquet files and containers in Azure Synapse Link for Azure Cosmos DB. You can use Spark or SQL to read or transform data with complex schemas such as arrays or nested structures. The following example is completed with a single document, but it can easily scale to billions of documents with Spark or SQL. The code included in this article uses PySpark (Python).

Use case

Complex data types are increasingly common and represent a challenge for data engineers. Analyzing nested schema and arrays can involve time-consuming and complex SQL queries. Additionally, it can be difficult to rename or cast the nested columns data type. Also, when you're working with deeply nested objects, you can encounter performance problems.

Data engineers need to understand how to efficiently process complex data types and make them easily accessible to everyone. In the following example, you use Spark in Azure Synapse Analytics to read and transform objects into a flat structure through data frames. You use the serverless model of SQL in Azure Synapse Analytics to query such objects directly, and return those results as a regular table.

What are arrays and nested structures?

The following object comes from Application Insights. In this object, there are nested structures and arrays that contain nested structures.

```
{
    "id": "66532691-ab20-11ea-8b1d-936b3ec64e54",
    "context": {
        "data": {
            "eventTime": "2020-06-10T13:43:34.553Z",
            "samplingRate": "100.0",
            "isSynthetic": "false"
        },
        "session": {
            "isFirst": "false",
            "id": "38619c14-7a23-4687-8268-95862c5326b1"
        },
        "custom": {
            "dimensions": [
                {
                    "customerInfo": {
                        "ProfileType": "ExpertUser",
                        "RoomName": "",
                        "CustomerName": "diamond",
                        "UserName": "XXXX@yahoo.com"
                    }
                },
                {
                    "customerInfo": {
                        "ProfileType": "Novice",
                        "RoomName": "",
                        "CustomerName": "topaz",
                        "UserName": "XXXX@outlook.com"
                    }
                }
            ]
        }
    }
}
```

Schema example of arrays and nested structures

When you're printing the schema of the object's data frame (called **df**) with the command `df.printschema`, you see the following representation:

- Yellow represents nested structures.
- Green represents an array with two elements.

```
root
 |-- _rid: string (nullable = true)
 |-- _ts: long (nullable = true)
 |-- id: string (nullable = true)
 |-- context: struct (nullable = true)
 |    |-- data: struct (nullable = true)
 |    |    |-- eventTime: string (nullable = true)
 |    |    |-- samplingRate: string (nullable = true)
 |    |    |-- isSynthetic: string (nullable = true)
 |    |-- session: struct (nullable = true)
 |    |    |-- isFirst: string (nullable = true)
 |    |    |-- id: string (nullable = true)
 |    |-- custom: struct (nullable = true)
 |    |    |-- dimensions: array (nullable = true)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- customerInfo: struct (nullable = true)
 |    |    |    |    |    |-- ProfileType: string (nullable = true)
 |    |    |    |    |    |-- RoomName: string (nullable = true)
 |    |    |    |    |    |-- CustomerName: string (nullable = true)
 |    |    |    |    |    |-- UserName: string (nullable = true)
 |-- _etag: string (nullable = true)
```

`_rid`, `_ts`, and `_etag` have been added to the system as the document was ingested into the Azure Cosmos DB transactional store.

The preceding data frame counts for 5 columns and 1 row only. After transformation, the curated data frame will have 13 columns and 2 rows, in a tabular format.

Flatten nested structures and explode arrays

With Spark in Azure Synapse Analytics, it's easy to transform nested structures into columns and array elements into multiple rows. Use the following steps for implementation.

Define a function to flatten the nested schema

You can use this function without change. Create a cell in a PySpark notebook with the following function:

```
from pyspark.sql.functions import col

def flatten_df(nested_df):
    stack = [((), nested_df)]
    columns = []

    while len(stack) > 0:
        parents, df = stack.pop()

        flat_cols = [
            col(".".join(parents + (c[0],))).alias("_".join(parents + (c[0],)))
            for c in df.dtypes
            if c[1][:6] != "struct"
        ]

        nested_cols = [
            c[0]
            for c in df.dtypes
            if c[1][:6] == "struct"
        ]

        columns.extend(flat_cols)

        for nested_col in nested_cols:
            projected_df = df.select(nested_col + ".*")
            stack.append((parents + (nested_col,), projected_df))

    return nested_df.select(columns)
```

Use the function to flatten the nested schema

In this step, you flatten the nested schema of the data frame (**df**) into a new data frame (`df_flat`):

```
from pyspark.sql.types import StringType, StructField, StructType
df_flat = flatten_df(df)
display(df_flat.limit(10))
```

The display function should return 10 columns and 1 row. The array and its nested elements are still there.

Transform the array

Here, you transform the array, `context_custom_dimensions`, in the data frame `df_flat`, into a new data frame `df_flat_explode`. In the following code, you also define which column to select:

```
from pyspark.sql.functions import explode
from pyspark.sql.functions import flatten
from pyspark.sql.functions import arrays_zip
df_flat_explode =
df_flat.select("_rid","_ts","id","_etag",explode(df_flat.context_custom_dimensions),"context_session_isFirst
","context_session_id","context_data_eventTime","context_data_samplingRate","context_data_isSynthetic")\
.select("_rid","_ts","id","_etag","col.*","context_session_isFirst","context_session_id","context_data_event
Time","context_data_samplingRate","context_data_isSynthetic")
display(df_flat_explode.limit(10))
```

The display function should return 10 columns and 2 rows. The next step is to flatten nested schemas with the function defined in step 1.

Use the function to flatten the nested schema

Finally, you use the function to flatten the nested schema of the data frame `df_flat_explode`, into a new data frame, `df_flat_explode_flat`:

```
df_flat_explode_flat = flatten_df(df_flat_explode)
display(df_flat_explode_flat.limit(10))
```

The display function should show 13 columns and 2 rows.

The function `printSchema` of the data frame `df_flat_explode_flat` returns the following result:

```
root
 |-- _rid: string (nullable = true)
 |-- _ts: long (nullable = true)
 |-- id: string (nullable = true)
 |-- _etag: string (nullable = true)
 |-- context_session_isFirst: string (nullable = true)
 |-- context_session_id: string (nullable = true)
 |-- context_data_eventTime: string (nullable = true)
 |-- context_data_samplingRate: string (nullable = true)
 |-- context_data_isSynthetic: string (nullable = true)
 |-- customerInfo_ProfileType: string (nullable = true)
 |-- customerInfo_RoomName: string (nullable = true)
 |-- customerInfo_CustomerName: string (nullable = true)
 |-- customerInfo_UserName: string (nullable = true)
```

Read arrays and nested structures directly

With the serverless model of SQL, you can query and create views and tables over such objects.

First, depending how the data has been stored, users should use the following taxonomy. Everything shown in uppercase is specific to your use case:

BULK	FORMAT
'https://ACCOUNTNAME.dfs.core.windows.net/FILESYSTEM/PATH/FINENAME.parquet'	'Parquet' (ADLSg2)

BULK	FORMAT
N'endpoint=https://ACCOUNTNAME.documents-staging.windows-ppe.net:443/;account=ACCOUNTNAME;database=DATABASENAME;collection=COLLECTIONNAME;region=REGIONTOQUERY', SECRET='YOURSECRET'	'CosmosDB' (Azure Synapse Link)

Replace each field as follows:

- 'YOUR BULK ABOVE' is the connection string of the data source you connect to.
- 'YOUR TYPE ABOVE' is the format you use to connect to the source.

```
select *
FROM
openrowset(
    BULK 'YOUR BULK ABOVE',
    FORMAT='YOUR TYPE ABOVE'
)
with (id varchar(50),
        contextdataeventTime varchar(50) '$.context.data.eventTime',
        contextdatasamplingRate varchar(50) '$.context.data.samplingRate',
        contextdataisSynthetic varchar(50) '$.context.data.isSynthetic',
        contextsessionisFirst varchar(50) '$.context.session.isFirst',
        contextsessionid varchar(50) '$.context.session.id',
        contextcustomdimensions varchar(max) '$.context.custom.dimensions'
) as q
cross apply openjson (contextcustomdimensions)
with ( ProfileType varchar(50) '$.customerInfo.ProfileType',
            RoomName varchar(50) '$.customerInfo.RoomName',
            CustomerName varchar(50) '$.customerInfo.CustomerName',
            UserName varchar(50) '$.customerInfo.UserName'
    )
```

There are two different types of operations:

- The first operation type is indicated in the following line of code, which defines the column called `contextdataeventTime` that refers to the nested element, `Context.Data.eventTime`.

  ```
  contextdataeventTime varchar(50) '$.context.data.eventTime'
  ```

  This line defines the column called `contextdataeventTime` that refers to the nested element, `Context>Data>eventTime`.

- The second operation type uses `cross apply` to create new rows for each element under the array. Then it defines each nested object.

  ```
  cross apply openjson (contextcustomdimensions)
  with ( ProfileType varchar(50) '$.customerInfo.ProfileType',
  ```

  If the array had 5 elements with 4 nested structures, the serverless model of SQL returns 5 rows and 4 columns. The serverless model of SQL can query in place, map the array in 2 rows, and display all nested structures into columns.

Next steps

- Learn how to query Synapse Link for Azure Cosmos DB with Spark 3

- Learn how to query Synapse Link for Azure Cosmos DB with Spark 2
- Query Parquet nested types

Use Azure Stream Analytics with dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

Azure Stream Analytics is a fully managed service providing low-latency, highly available, scalable complex event processing over streaming data in the cloud. You can learn the basics by reading Introduction to Azure Stream Analytics. You can then learn how to create an end-to-end solution with Stream Analytics by following the Get started using Azure Stream Analytics tutorial.

In this article, you will learn how to use your dedicated SQL pool as an output sink for high throughput data ingestion with Azure Stream Analytics jobs.

Prerequisites

- Azure Stream Analytics Job - To create an Azure Stream Analytics job, follow the steps in the Get started using Azure Stream Analytics tutorial to:
  1. Create an Event Hub input
  2. Configure and start event generator application
  3. Provision a Stream Analytics job
  4. Specify job input and query
- Dedicated SQL pool - To create a new dedicated SQL pool, follow the steps in the Quickstart: Create a dedicated SQL pool.

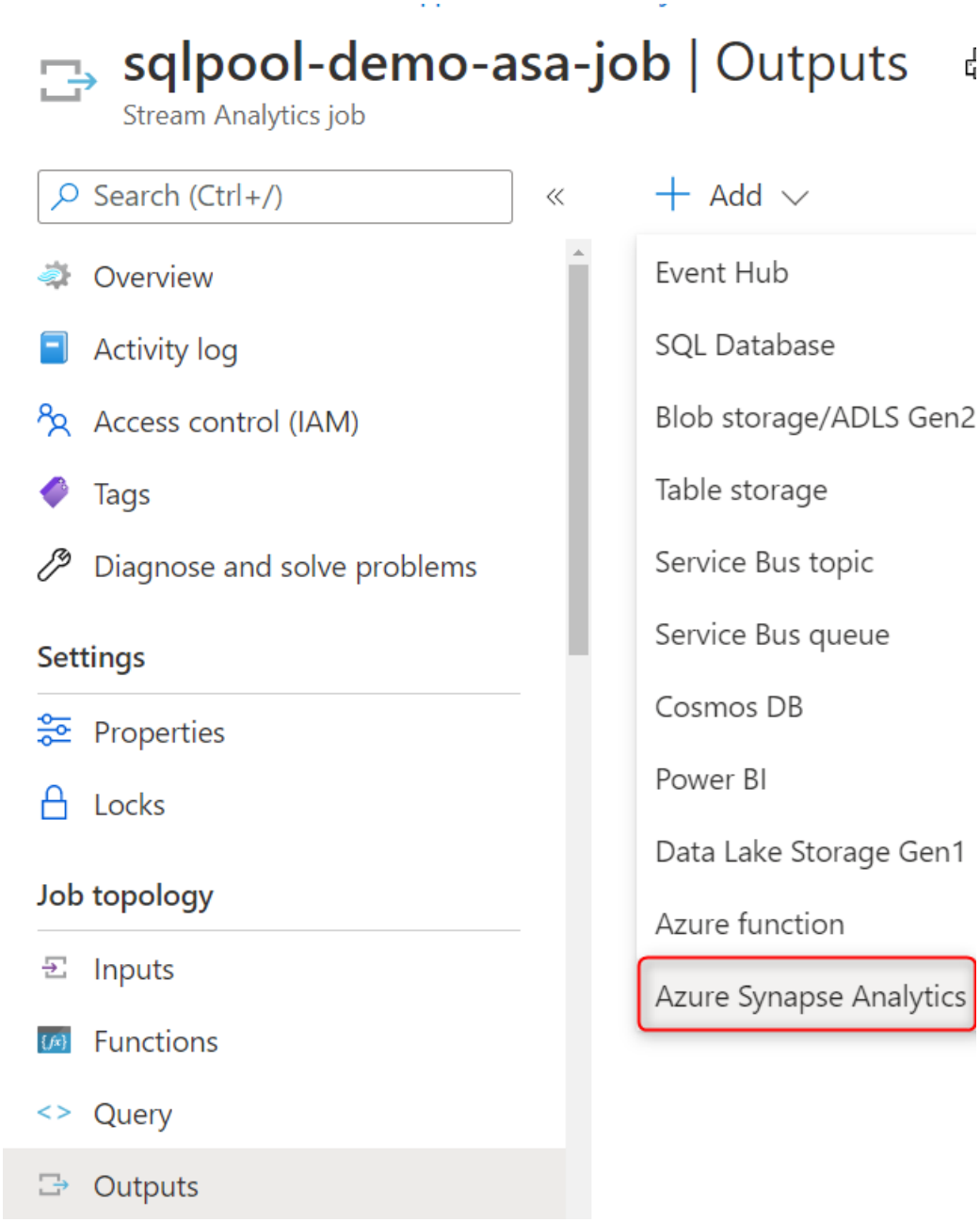
Specify streaming output to point to your dedicated SQL pool

Step 1

From the Azure portal, go to your Stream Analytics job and click on **Outputs** under the **Job topology** menu.

Step 2

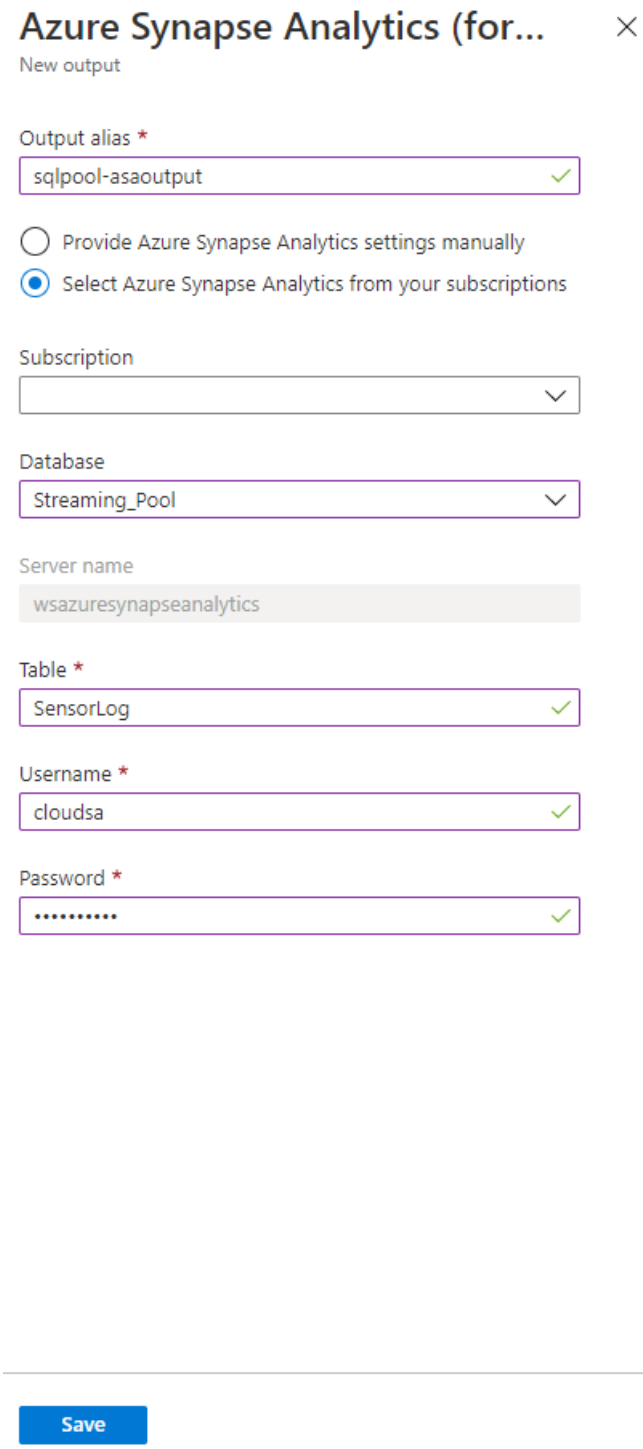
Click on the **Add** button and choose **Azure Synapse Analytics** from the drop down menu.



Step 3

Enter the following values:

- *Output Alias*: Enter a friendly name for this job output.
- *Subscription*:
  - If your dedicated SQL pool is in the same subscription as the Stream Analytics job, click on ***Select Azure Synapse Analytics from your subscriptions***.
  - If your dedicated SQL pool is in a different subscription, click on Provide Azure Synapse Analytics settings manually.
- *Database*: Select the destination database from the drop down list.
- *User Name*: Specify the user name of an account that has write permissions for the database.
- *Password*: Provide the password for the specified user account.
- *Table*: Specify the name of the target table in the database.
- click on the **Save** button



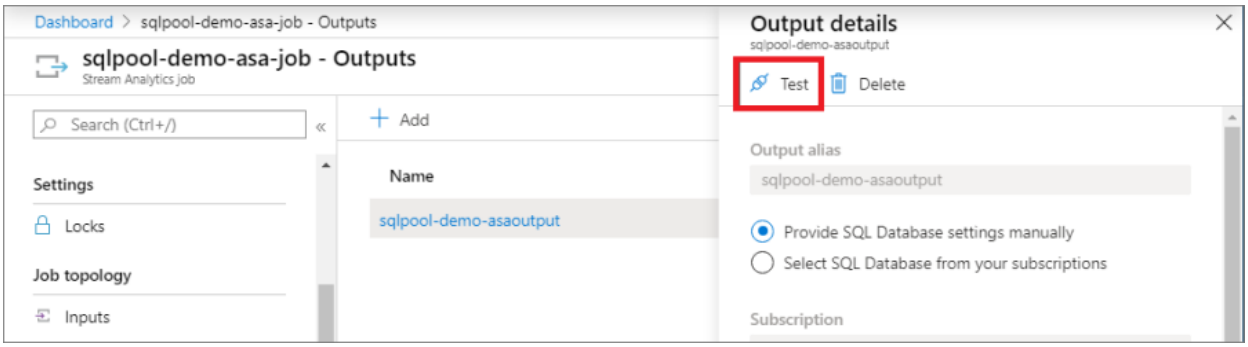
Step 4

Before you can run a test, you will need to create the table in your dedicated SQL pool. Run the following table creation script using SQL Server Management Studio (SSMS) or your choice of query tool.

```
CREATE TABLE SensorLog
(
    RecordType VARCHAR(2)
    , SystemIdentity VARCHAR(2)
    , FileNum INT
    , SwitchNum VARCHAR(50)
    , CallingNum VARCHAR(25)
    , CallingIMSI VARCHAR(25)
    , CalledNum VARCHAR(25)
    , CalledIMSI VARCHAR(25)
    , DateS VARCHAR(25)
    , TimeS VARCHAR(25)
    , TimeType INT
    , CallPeriod INT
    , CallingCellID VARCHAR(25)
    , CalledCellID VARCHAR(25)
    , ServiceType VARCHAR(25)
    , [Transfer] INT
    , IncomingTrunk VARCHAR(25)
    , OutgoingTrunk VARCHAR(25)
    , MSRN VARCHAR(25)
    , CalledNum2 VARCHAR(25)
    , FCIFlag VARCHAR(25)
    , callrecTime VARCHAR(50)
    , EventProcessedUtcTime VARCHAR(50)
    , PartitionId int
    , EventEnqueuedUtcTime VARCHAR(50)
    )
WITH (DISTRIBUTION = ROUND_ROBIN)
```

Step 5

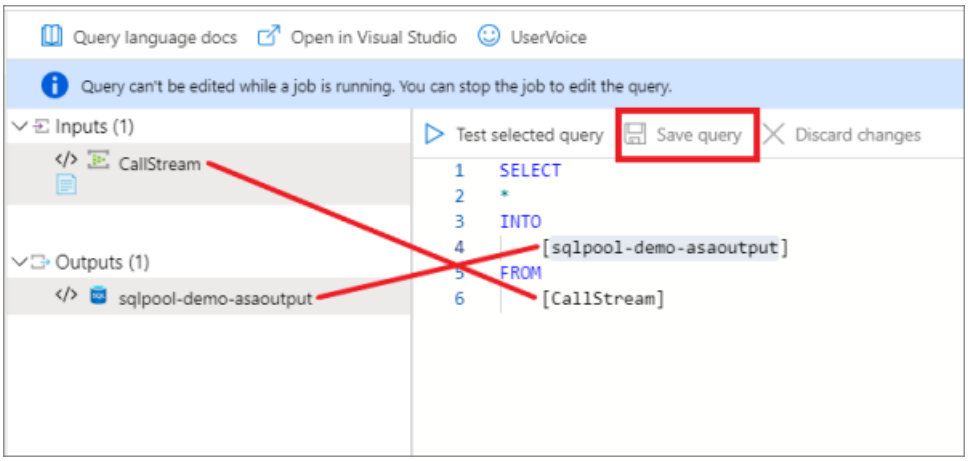
On the Azure portal for Stream Analytics job, click on your job name. Click on the *Test* button in the *Output details* pane.



When the connection to the database succeeds, you will see a notification in the portal.

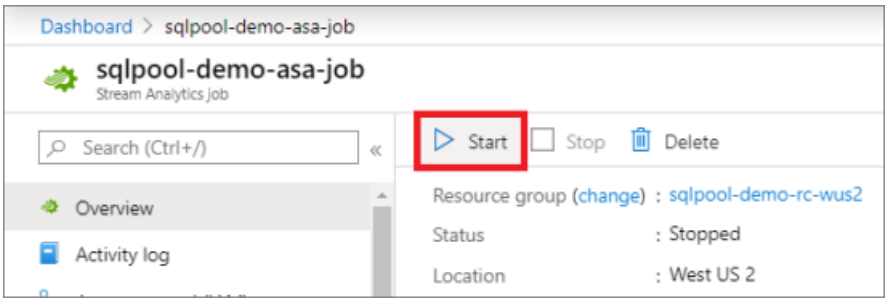
Step 6

Click on the *Query* menu under *Job topology* and change the query to insert data into the Stream output you created. Click on the *Test selected query* button to test your query. Click *Save Query* button when your query test is successful.

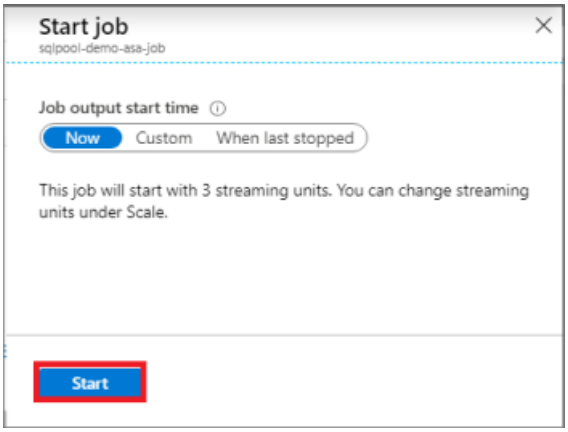


Step 7

Start the Azure Stream Analytics job. Click on the ***Start*** button on the ***Overview*** menu.



Click the ***Start*** button on the start job pane.



Next steps

For an overview of integration, see Integrate other services. For more development tips, see Design decisions and coding techniques for dedicated SQL pool.

Connect to Synapse SQL

8/19/2022 • 2 minutes to read • Edit Online

Get connected to the Synapse SQL capability in Azure Synapse Analytics.

Supported tools for serverless SQL pool

Azure Data Studio is fully supported starting from version 1.18.0. SSMS is partially supported starting from version 18.5, you can use it to connect and query only.

> **NOTE**
>
> If an AAD login has a connection open for more than 1 hour at time of query execution, any query that relies on AAD will fail. This includes querying storage using AAD pass-through and statements that interact with AAD (like CREATE EXTERNAL PROVIDER). This affects every tool that keeps connections open, like in query editor in SSMS and ADS. Tools that open new connections to execute a query, like Synapse Studio, are not affected.

> You can restart SSMS or connect and disconnect in ADS to mitigate this issue.

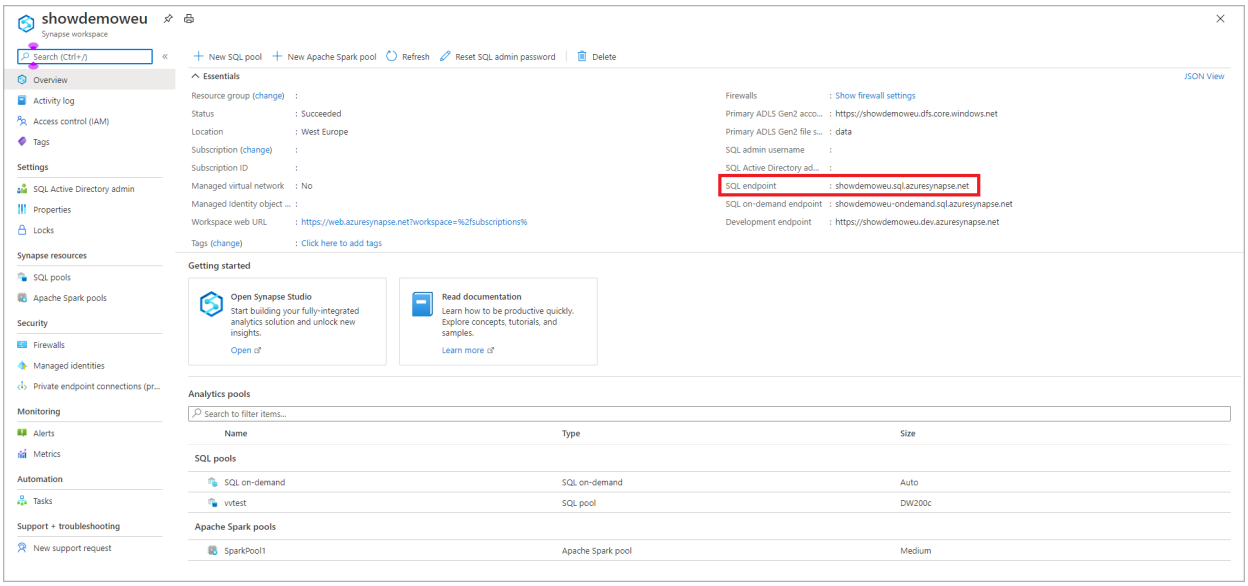
Find your server name

The server name for the dedicated SQL pool in the following example is: showdemoweu.sql.azuresynapse.net. The server name for the serverless SQL pool in the following example is: showdemoweu-ondemand.sql.azuresynapse.net.

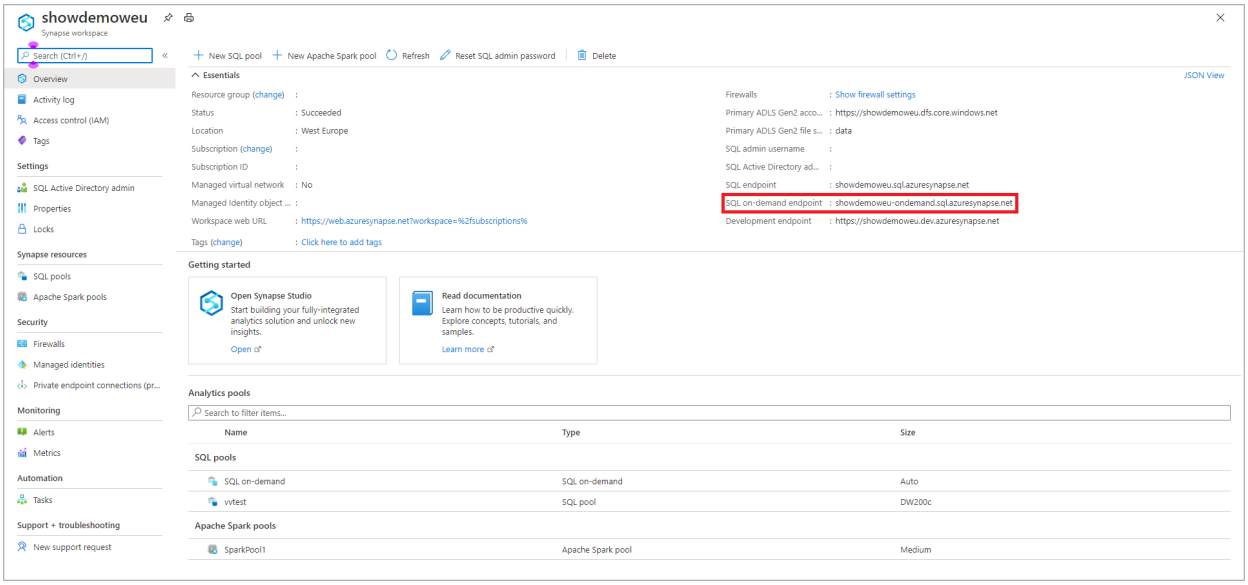
To find the fully qualified server name:

1. Go to the Azure portal.
2. Select on **Synapse workspaces**.
3. Select on the workspace you want to connect to.
4. Go to overview.
5. Locate the full server name.

SQL pool



serverless SQL pool



Supported drivers and connection strings

Synapse SQL supports ADO.NET, ODBC, PHP, and JDBC. To find the latest version and documentation, select one of the preceding drivers. To automatically generate the connection string for the driver that you're using from the Azure portal, select **Show database connection strings** from the preceding example. Following are also some examples of what a connection string looks like for each driver.

> **NOTE**
>
> Consider setting the connection timeout to 300 seconds to allow your connection to survive short periods of unavailability.

ADO.NET connection string example

```
Server=tcp:{your_server}.sql.azuresynapse.net,1433;Database={your_database};User ID=
{your_user_name};Password={your_password_here};Encrypt=True;TrustServerCertificate=False;Connection
Timeout=30;
```

ODBC connection string example

```
Driver={SQL Server Native Client 11.0};Server=tcp:{your_server}.sql.azuresynapse.net,1433;Database=
{your_database};Uid={your_user_name};Pwd=
{your_password_here};Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;
```

PHP connection string example

```
Server: {your_server}.sql.azuresynapse.net,1433 \r\nSQL Database: {your_database}\r\nUser Name:
{your_user_name}\r\n\r\nPHP Data Objects(PDO) Sample Code:\r\n\r\ntry {\r\n   $conn = new PDO (
\"sqlsrv:server = tcp:{your_server}.sql.azuresynapse.net,1433; Database = {your_database}\", \"
{your_user_name}\", \"{your_password_here}\");\r\n    $conn->setAttribute( PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION );\r\n}\r\ncatch ( PDOException $e ) {\r\n   print( \"Error connecting to SQL
Server.\" );\r\n   die(print_r($e));\r\n}\r\n\r\nSQL Server Extension Sample Code:\r\n\r\n$connectionInfo =
array(\"UID\" => \"{your_user_name}\", \"pwd\" => \"{your_password_here}\", \"Database\" => \"
{your_database}\", \"LoginTimeout\" => 30, \"Encrypt\" => 1, \"TrustServerCertificate\" =>
0);\r\n$serverName = \"tcp:{your_server}.sql.azuresynapse.net,1433\";\r\n$conn = sqlsrv_connect($serverName,
$connectionInfo);
```

JDBC connection string example

```
jdbc:sqlserver://yourserver.sql.azuresynapse.net:1433;database=yourdatabase;user={your_user_name};password=
{your_password_here};encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.sql.azuresynapse.net;
loginTimeout=30;
```

Connection settings

Synapse SQL standardizes some settings during connection and object creation. These settings can't be overridden and include:

DATABASE SETTING	VALUE
ANSI_NULLS	ON
QUOTED_IDENTIFIERS	ON
DATEFORMAT	mdy
DATEFIRST	7

Recommendations

For executing **serverless SQL pool** queries, recommended tools are Azure Data Studio and Azure Synapse Studio.

Next steps

To connect and query with Visual Studio, see Query with Visual Studio. To learn more about authentication options, see Authentication to Synapse SQL.

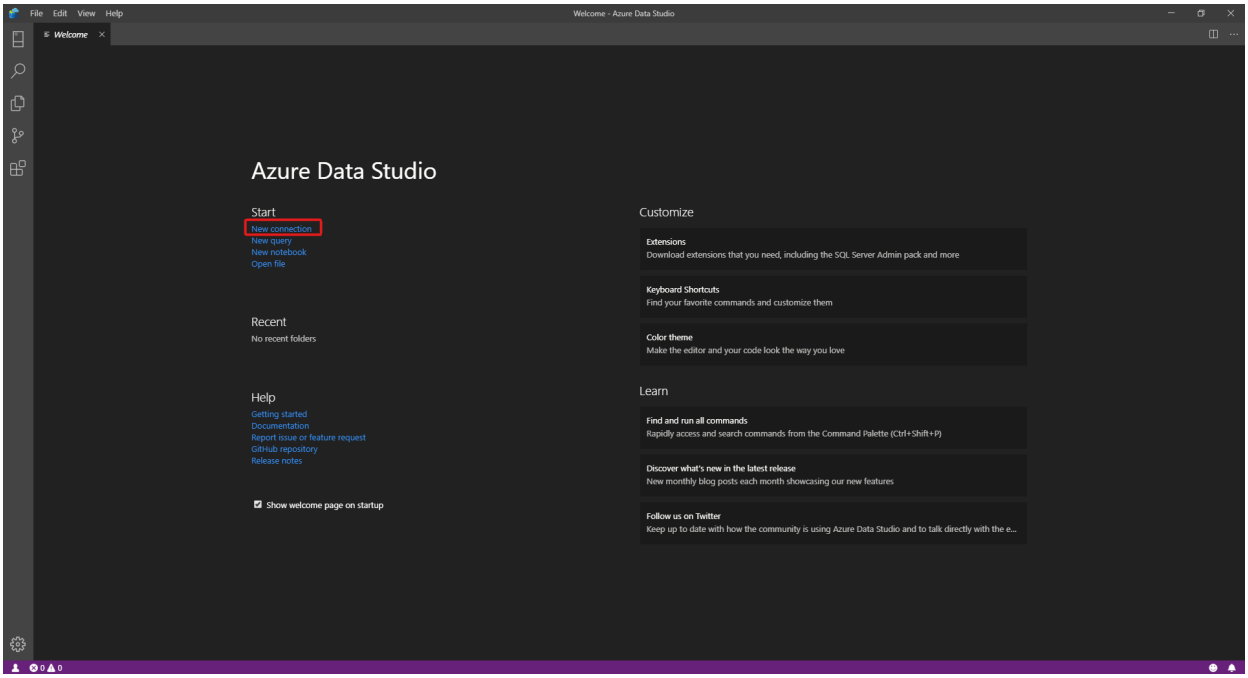
Connect to Synapse SQL with Azure Data Studio

8/19/2022 • 2 minutes to read • Edit Online

You can use Azure Data Studio to connect to and query Synapse SQL in Azure Synapse Analytics.

Connect

To connect to Synapse SQL, open Azure Data Studio and select **New Connection**.



Choose **Microsoft SQL Server** as the **Connection type**.

The connection requires the following parameters:

- **Server**: Server in the form `<Azure Synapse workspace name>`-ondemand.sql.azuresynapse.net
- **Database**: Database name

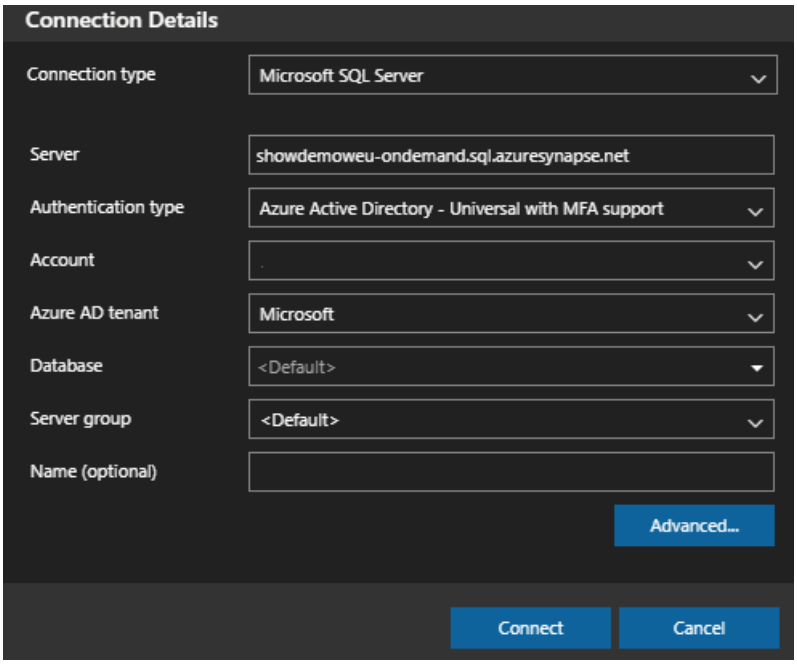
> **NOTE**
>
> If you would like to use **serverless SQL pool** the URL should look like:
>
> - `<Azure Synapse workspace name>`-ondemand.sql.azuresynapse.net.
>
> If you would like to use **dedicated SQL pool** the URL should look like:
>
> - `<Azure Synapse workspace name>`.sql.azuresynapse.net

Choose **Windows Authentication**, **Azure Active Directory**, or **SQL Login** as the **Authentication Type**.

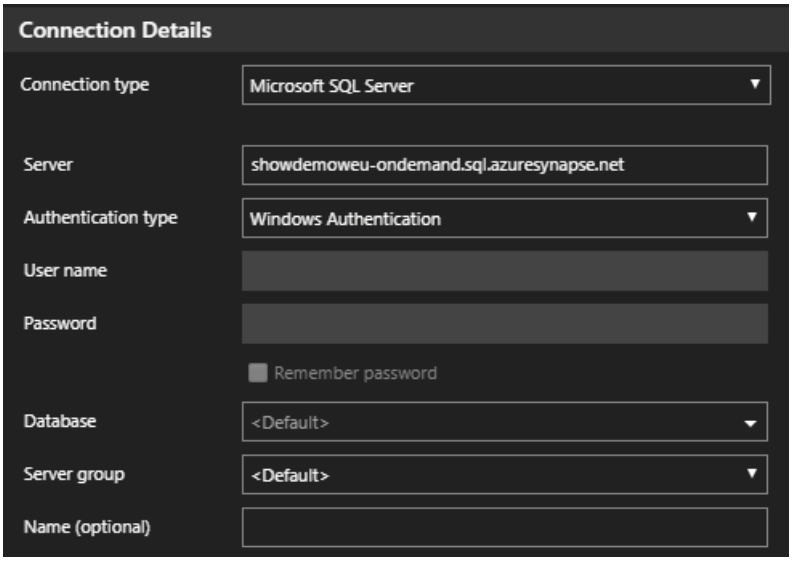
To use **SQL Login** as the authentication type, add the username/password parameters:

- **User**: Server user in the form `<User>`
- **Password**: Password associated with the user

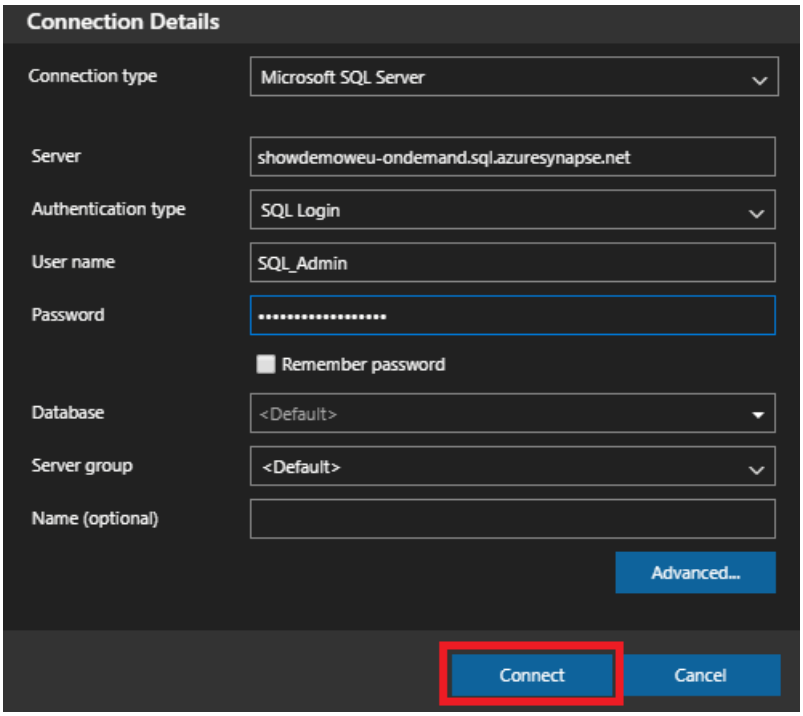
To use Azure Active Directory, you need to choose the needed authentication type.



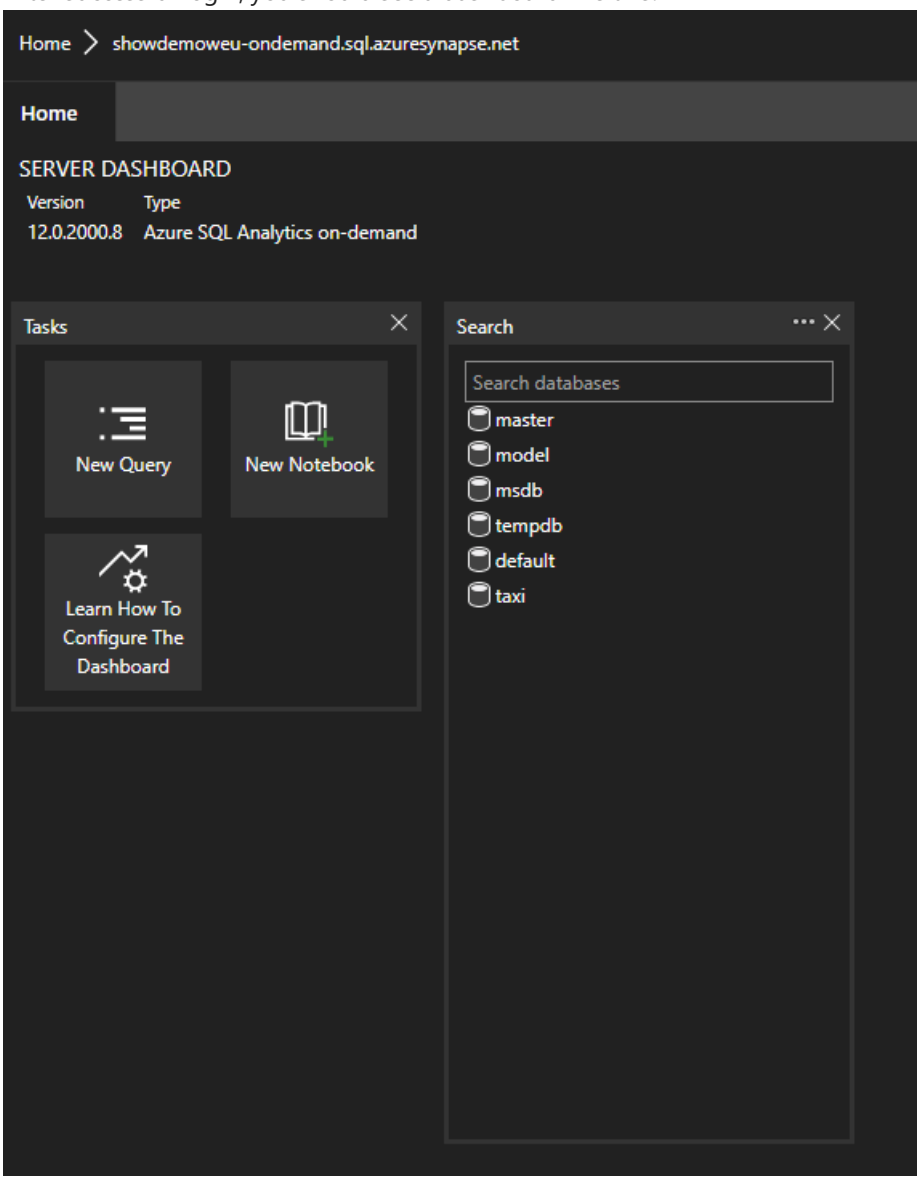
The following screenshot shows the **Connection Details** for **Windows Authentication**:



The following screenshot shows the **Connection Details** using **SQL Login**:

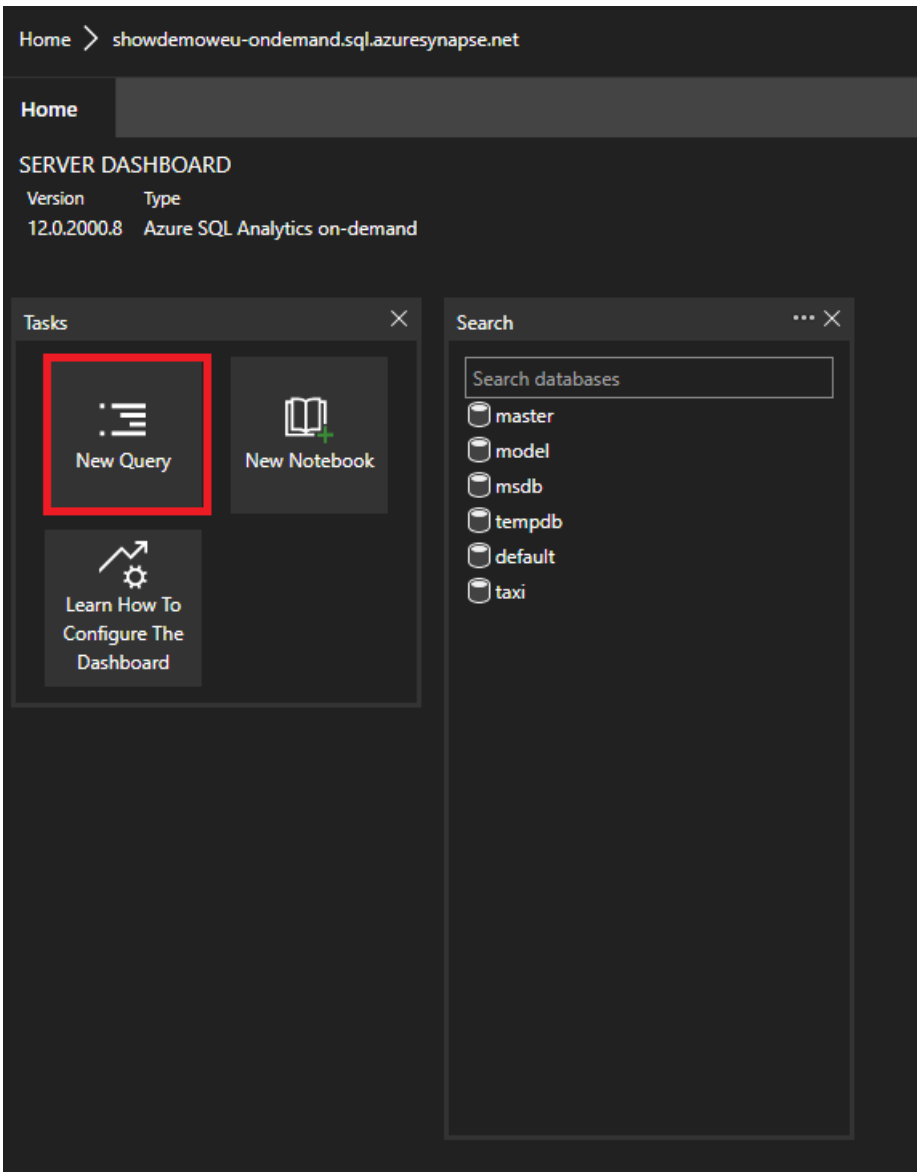


After successful login, you should see a dashboard like this:



Query

Once connected, you can query Synapse SQL using supported Transact-SQL (T-SQL) statements against the instance. Select **New query** from the dashboard view to get started.



For example, you can use the following Transact-SQL statement to query Parquet files using serverless SQL pool:

```
SELECT COUNT(*)
FROM
OPENROWSET(
    BULK
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us_population_county/year=20
*/*.parquet',
    FORMAT='PARQUET'
)
```

Next steps

Explore other ways to connect to Synapse SQL:

- SSMS
- Power BI
- Visual Studio
- sqlcmd

Visit Use Azure Data Studio to connect and query data using a dedicated SQL pool in Azure Synapse Analytics,

for more information.

Connection strings for Synapse SQL

8/19/2022 • 2 minutes to read • Edit Online

You can connect to Synapse SQL with several different application protocols such as, ADO.NET, ODBC, PHP, and JDBC. Below are some examples of connections strings for each protocol.

You can also use the Azure portal to build your connection string. To build your connection string using the Azure portal, navigate to your database blade, under *Essentials* select *Show database connection strings*.

Sample ADO.NET connection string

```
Server=tcp:{your_server}.sql.azuresynapse.net,1433;Database={your_database};User ID=
{your_user_name};Password={your_password_here};Encrypt=True;TrustServerCertificate=False;Connection
Timeout=30;
```

Sample ODBC connection string

```
Driver={SQL Server Native Client 11.0};Server=tcp:{your_server}.sql.azuresynapse.net,1433;Database=
{your_database};Uid={your_user_name};Pwd=
{your_password_here};Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;
```

Sample PHP connection string

```
Server: {your_server}.sql.azuresynapse.net,1433 \r\nSQL Database: {your_database}\r\nUser Name:
{your_user_name}\r\n\r\nPHP Data Objects(PDO) Sample Code:\r\n\r\ntry {\r\n   $conn = new PDO (
\"sqlsrv:server = tcp:{your_server}.sql.azuresynapse.net,1433; Database = {your_database}\", \"
{your_user_name}\", \"{your_password_here}\");\r\n    $conn->setAttribute( PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION );\r\n}\r\ncatch ( PDOException $e ) {\r\n   print( \"Error connecting to SQL
Server.\" );\r\n   die(print_r($e));\r\n}\r\n\r\nSQL Server Extension Sample Code:\r\n\r\n$connectionInfo =
array(\"UID\" => \"{your_user_name}\", \"pwd\" => \"{your_password_here}\", \"Database\" => \"
{your_database}\", \"LoginTimeout\" => 30, \"Encrypt\" => 1, \"TrustServerCertificate\" =>
0);\r\n$serverName = \"tcp:{your_server}.sql.azuresynapse.net,1433\";\r\n$conn = sqlsrv_connect($serverName,
$connectionInfo);
```

Sample JDBC connection string

```
jdbc:sqlserver://yourserver.sql.azuresynapse.net:1433;database=yourdatabase;user={your_user_name};password=
{your_password_here};encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.sql.azuresynapse.net;
loginTimeout=30;
```

NOTE

Consider setting the connection timeout to 300 seconds to allow the connection to survive short periods of unavailability.

Recommendations

For executing **serverless SQL pool** queries, recommended tools are Azure Data Studio and Azure Synapse Studio.

Next steps

To start querying your analytics with Visual Studio and other applications, see Query with Visual Studio.

Connect to Synapse SQL with SQL Server Management Studio (SSMS)

8/19/2022 • 3 minutes to read • Edit Online

You can use SQL Server Management Studio (SSMS) to connect to and query Synapse SQL in Azure Synapse Analytics through either serverless SQL pool or dedicated SQL pool resources.

Supported tools for serverless SQL pool

Azure Data Studio is fully supported starting from version 1.18.0. SSMS is partially supported starting from version 18.5, you can use it to connect and query only.

> **NOTE**
> If AAD login has connection open for more than 1 hour at time of query execution, any query that relies on AAD will fail. This includes querying storage using AAD pass-through and statements that interact with AAD (like CREATE EXTERNAL PROVIDER). This affects every tool that keeps connection open, like in query editor in SSMS and ADS. Tools that open new connection to execute query are not affected, like Synapse Studio. You can restart SSMS or connect and disconnect in ADS to mitigate this issue. .

Prerequisites

Before you begin, make sure you have the following prerequisites:

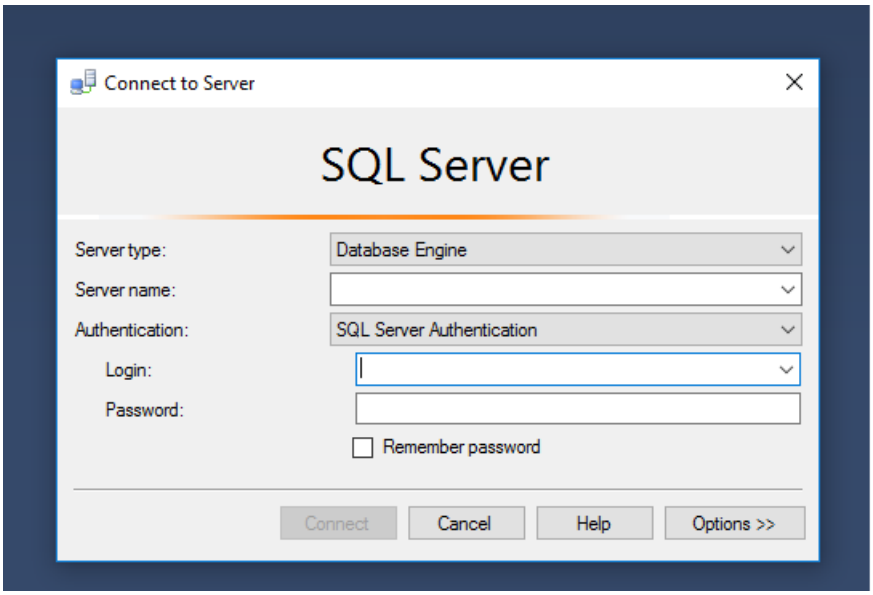
- SQL Server Management Studio (SSMS).
- For dedicated SQL pool, you need an existing data warehouse. To create one, see Create a dedicated SQL pool. For serverless SQL pool one is already provisioned, named Built-in, in your workspace at creation time.
- The fully qualified SQL Server name. To find this name, see Connect to Synapse SQL.

Connect

Dedicated SQL pool

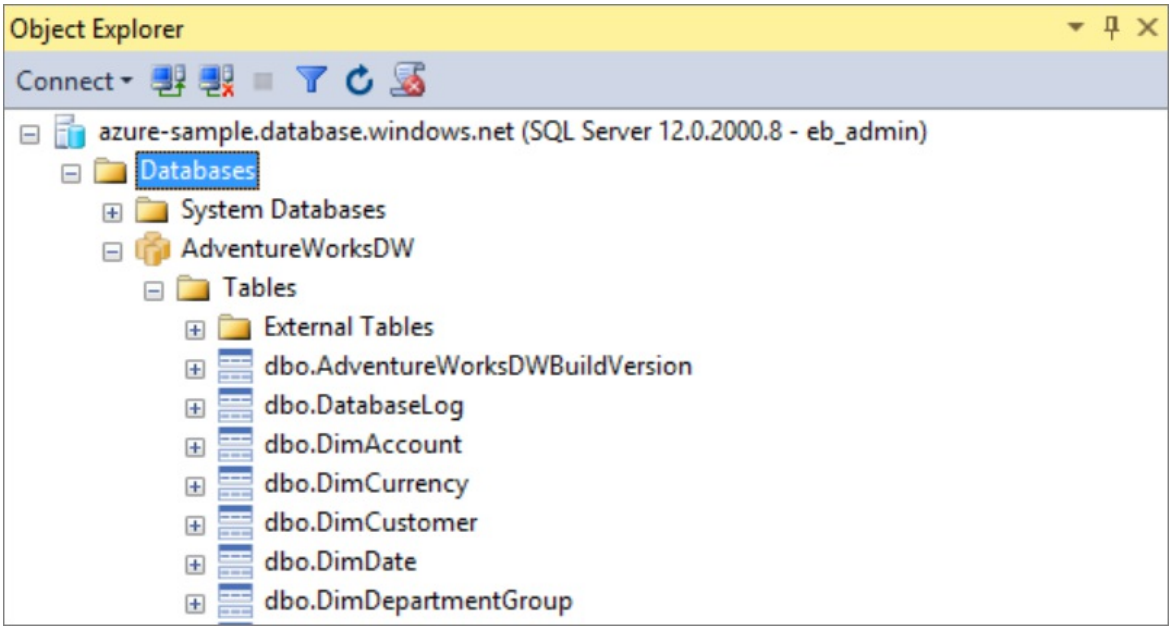
To connect to Synapse SQL using dedicated SQL pool, follow these steps:

1. Open SQL Server Management Studio (SSMS).
2. In the **Connect to Server** dialog box, fill in the fields, and then select **Connect**:



- **Server name**: Enter the **server name** previously identified.
- **Authentication**: Choose an authentication type, such as **SQL Server Authentication** or **Active Directory Integrated Authentication**.
- **User Name** and **Password**: Enter your user name and password if SQL Server Authentication was selected above.

3. Expand your Azure SQL Server in **Object Explorer**. You can view the databases associated with the server, such as the sample AdventureWorksDW database. You can expand the database to see the tables:

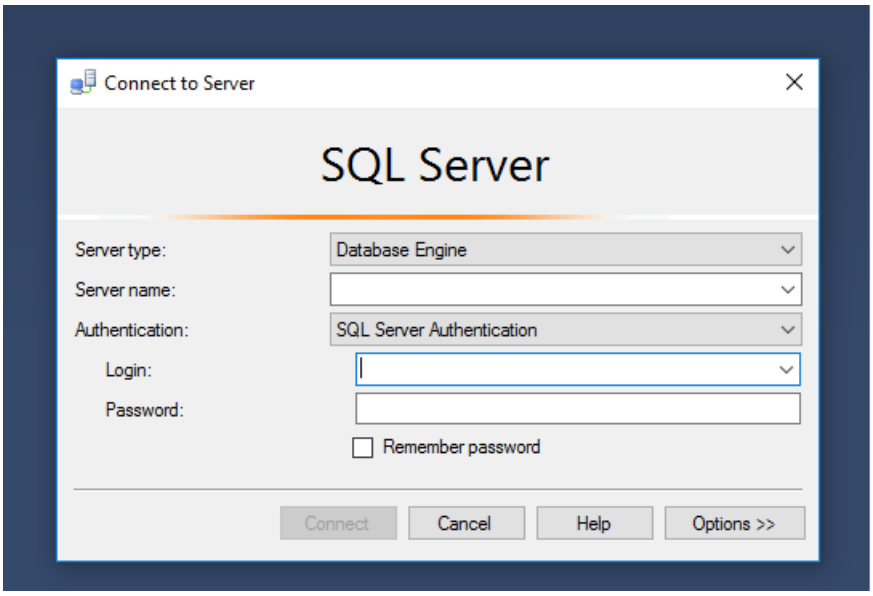


Serverless SQL pool

To connect to Synapse SQL using serverless SQL pool, follow these steps:

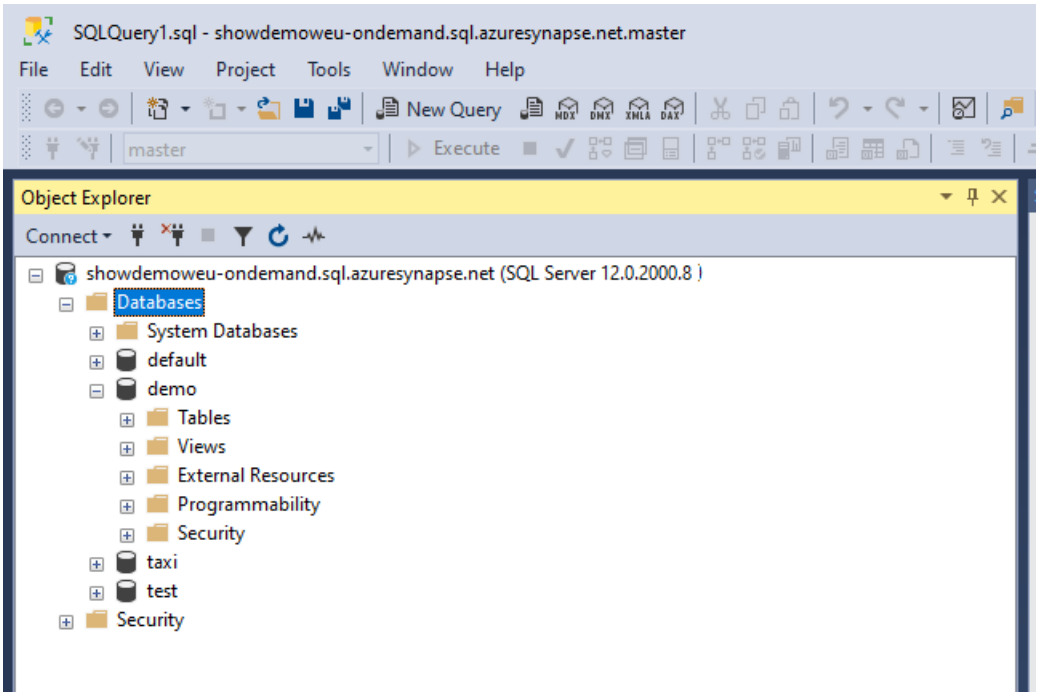
1. Open SQL Server Management Studio (SSMS).

2. In the **Connect to Server** dialog box, fill in the fields, and then select **Connect**:



- **Server name**: Enter the **server name** previously identified.
- **Authentication**: Choose an authentication type, such as **SQL Server Authentication** or **Active Directory Integrated Authentication**:
- **User Name** and **Password**: Enter your user name and password if SQL Server Authentication was selected above.
- Select **Connect**.

3. To explore, expand your Azure SQL server. You can view the databases associated with the server. Expand *demo* to see the content in your sample database.



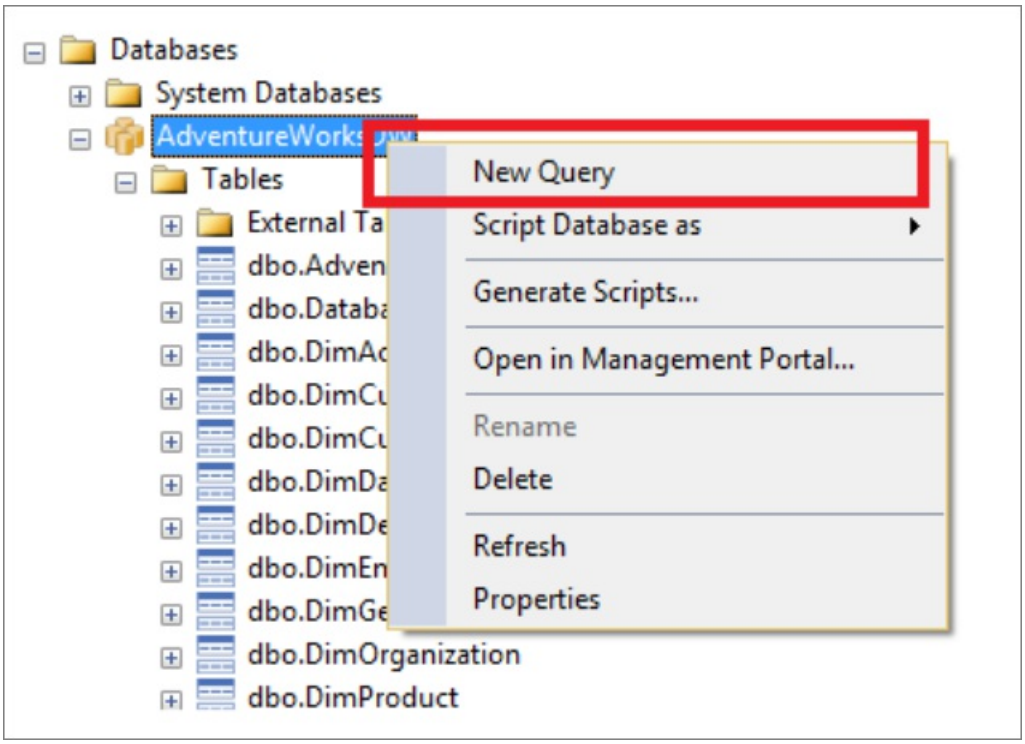
Run a sample query

Dedicated SQL pool

Now that a database connection has been established, you can query the data.

1. Right-click your database in SQL Server Object Explorer.

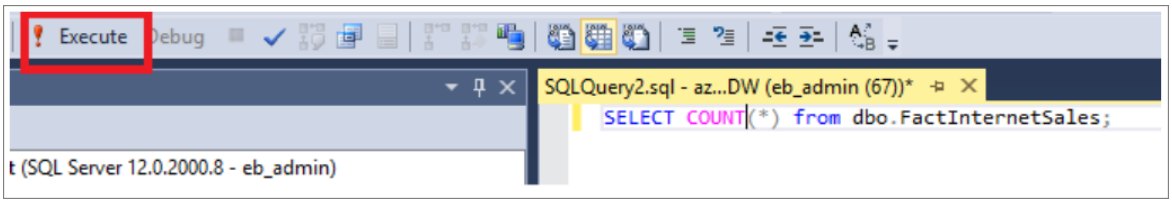
2. Select **New Query**. A new query window opens.



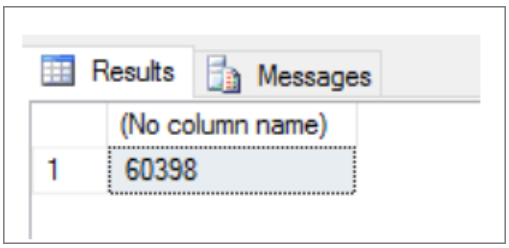
3. Copy the following T-SQL query into the query window:

```
SELECT COUNT(*) FROM dbo.FactInternetSales;
```

4. Run the query by selecting `Execute` or use the following shortcut: `F5`.



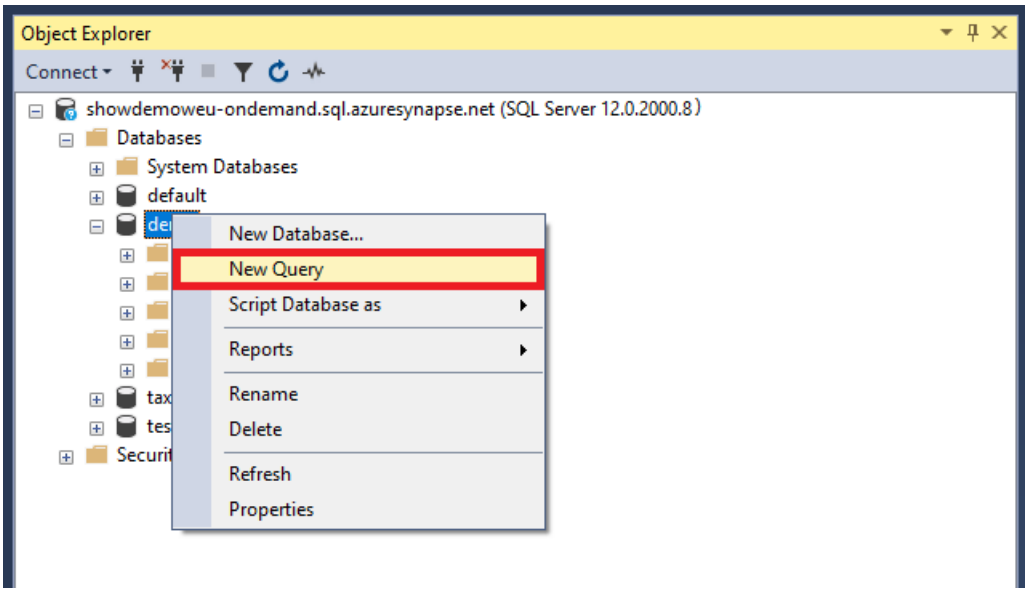
5. Look at the query results. In the following example, the FactInternetSales table has 60398 rows.



Serverless SQL pool

Now that you've established a database connection, you can query the data.

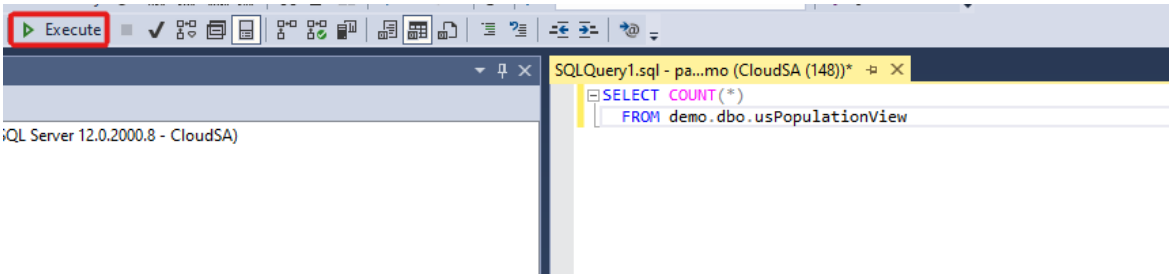
1. Right-click your database in SQL Server Object Explorer.
2. Select **New Query**. A new query window opens.



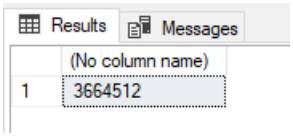
3. Copy the following T-SQL query into the query window:

```
SELECT COUNT(*) FROM demo.dbo.usPopulationView
```

4. Run the query by selecting `Execute` or use the following shortcut: `F5`.



5. Look at the query results. In this example, the usPopulationView view has 3664512 rows.



Next steps

Now that you can connect and query, try visualizing the data with Power BI.

To configure your environment for Azure Active Directory authentication, see Authenticate to Synapse SQL.

Connect to serverless SQL pool with Power BI Professional

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, we will go through the steps for connecting Power BI desktop to serverless SQL pool.

Prerequisites

You need the following tools to issue queries:

- SQL client of your choice:
  - Azure Data Studio
  - SQL Server Management Studio
- Power BI desktop installed

Parameters:

PARAMETER	DESCRIPTION
Serverless SQL pool service endpoint address	Will be used as server name
Serverless SQL pool service endpoint region	Will be used to determine what storage will we use in samples
Username and password for endpoint access	Will be used to access endpoint
Database you'll use to create views	This database will be used as starting point in samples

First-time setup

There are two steps prior to using samples:

1. Create database for your views
2. Create credentials to be used by serverless SQL pool to access files in storage

Create database

For this getting started article, you should create your own database to utilize as a demo. A database is needed for views creation. You'll use this database in some of the sample queries within this documentation.

> **NOTE**
>
> Databases are used only for viewing metadata, not for actual data.
>
> Write down the database name you're using, you'll need it later on.

```
DROP DATABASE IF EXISTS demo;
```

Create credentials

We need to create credentials before you can run queries. This credential will be used by serverless SQL pool service to access files in storage.

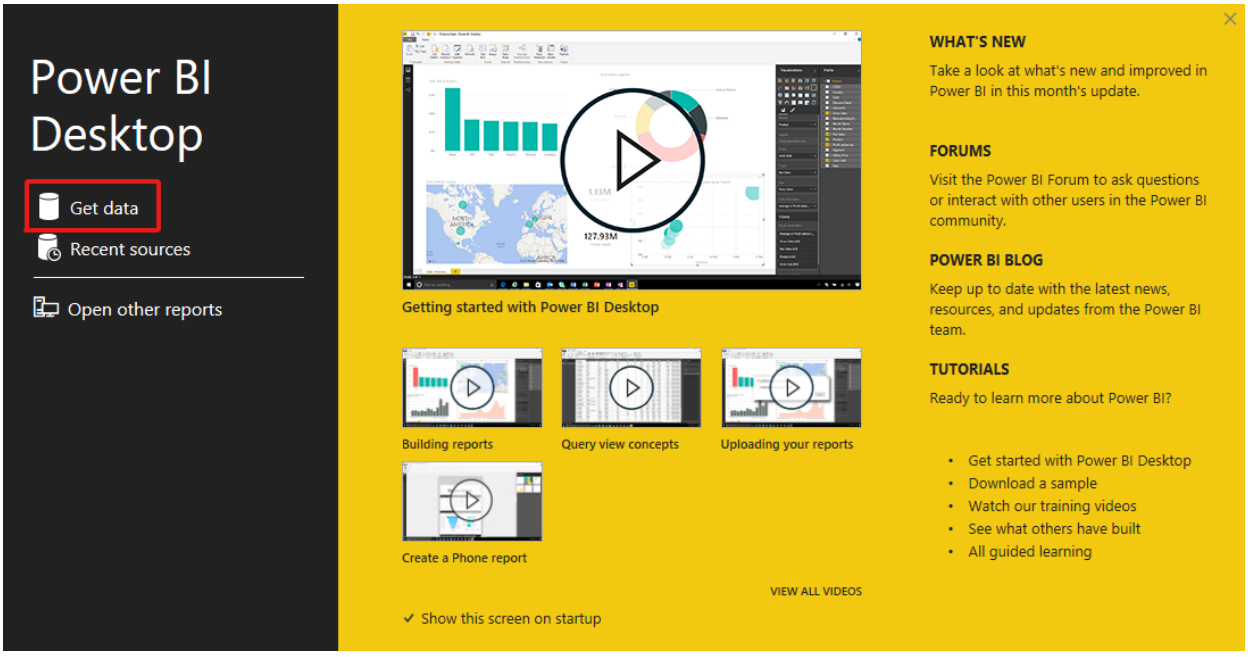
> **NOTE**
>
> You need to create credentials for storage account access. Although serverless SQL pool can access storage from different regions, having storage and Azure Synapse workspace in the same region will provide a better performance experience.

Code snippet on how to create credentials for Census data containers, run:

```
IF EXISTS (SELECT * FROM sys.credentials WHERE name =
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer')
DROP CREDENTIAL [https://azureopendatastorage.blob.core.windows.net/censusdatacontainer];
GO

-- Create credentials for Census Data container which resides in a azure open data storage account
-- There is no secret. We are using public storage account which doesn't need secret
CREATE CREDENTIAL [https://azureopendatastorage.blob.core.windows.net/censusdatacontainer]
WITH IDENTITY='SHARED ACCESS SIGNATURE',
SECRET = '';
GO
```

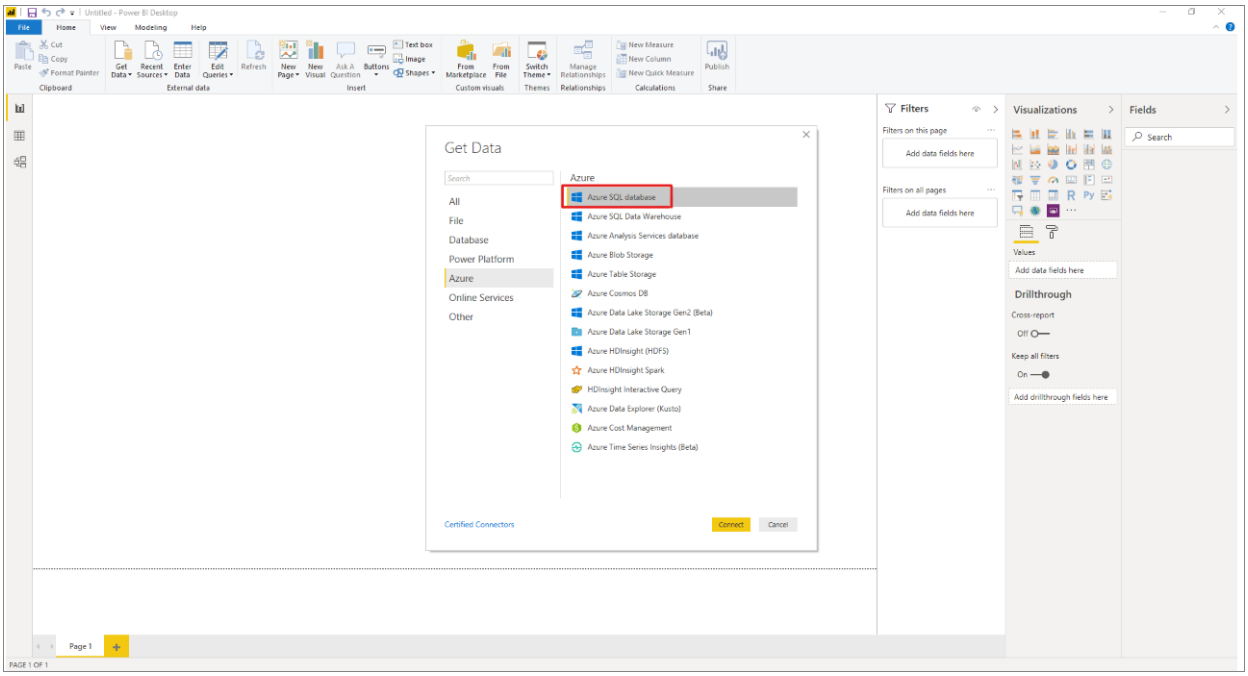
Create a Power BI desktop report

Open the Power BI desktop application and select the **Get data** option.



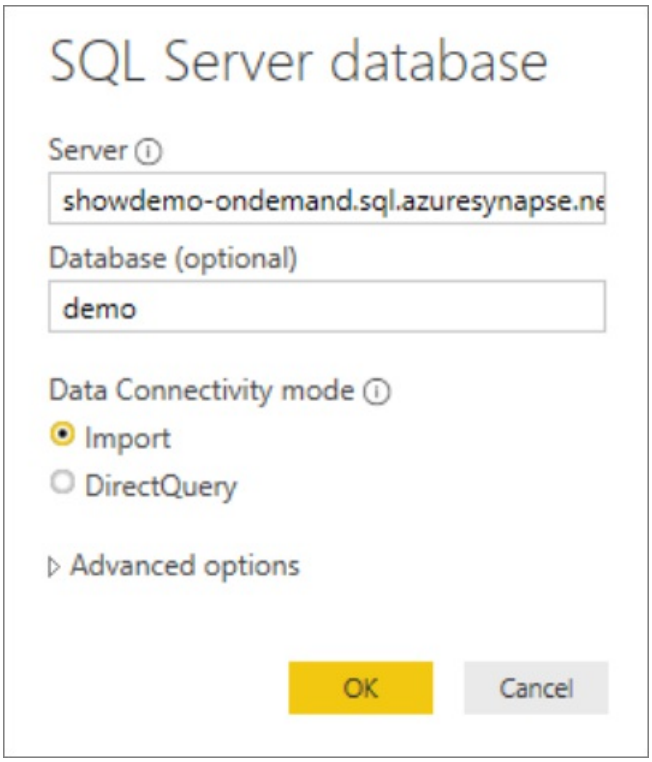
Step 1 - Select data source

Select **Azure** in the menu and then **Azure SQL Database**.



Step 2 - Select database

Write the URL for the database and the name of the database where the view resides.



Next steps

Advance to Query storage files to learn how to connect to serverless SQL pool using Azure Data Studio.

Connect to Synapse SQL with Visual Studio and SSDT

8/19/2022 • 2 minutes to read • Edit Online

Use Visual Studio to query dedicated SQL pool using Azure Synapse Analytics. This method uses the SQL Server Data Tools (SSDT) extension in Visual Studio 2019.

> **NOTE**
> Serverless SQL pool is not supported by SSDT.

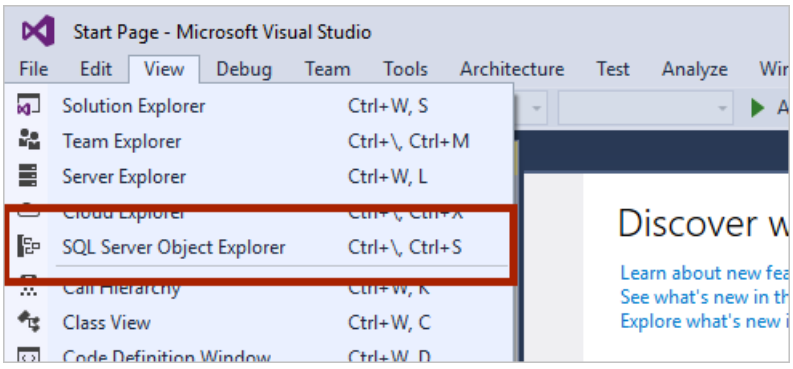
Prerequisites

To use this tutorial, you need to have the following components:

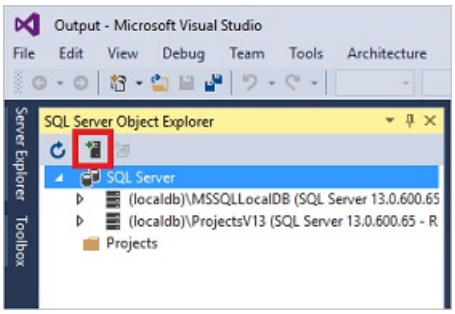
- An existing Synapse workspace, If you need to create one see Creating a Synapse workspace
- A dedicated SQL pool. If you do not have one, see Create a dedicated SQL pool.
- SSDT for Visual Studio. If you have Visual Studio, you probably already have this component. For installation instructions and options, see Installing Visual Studio and SSDT.
- The fully qualified SQL server name. To find this server name, see Connect to a dedicated SQL pool.

1. Connect to a dedicated SQL pool

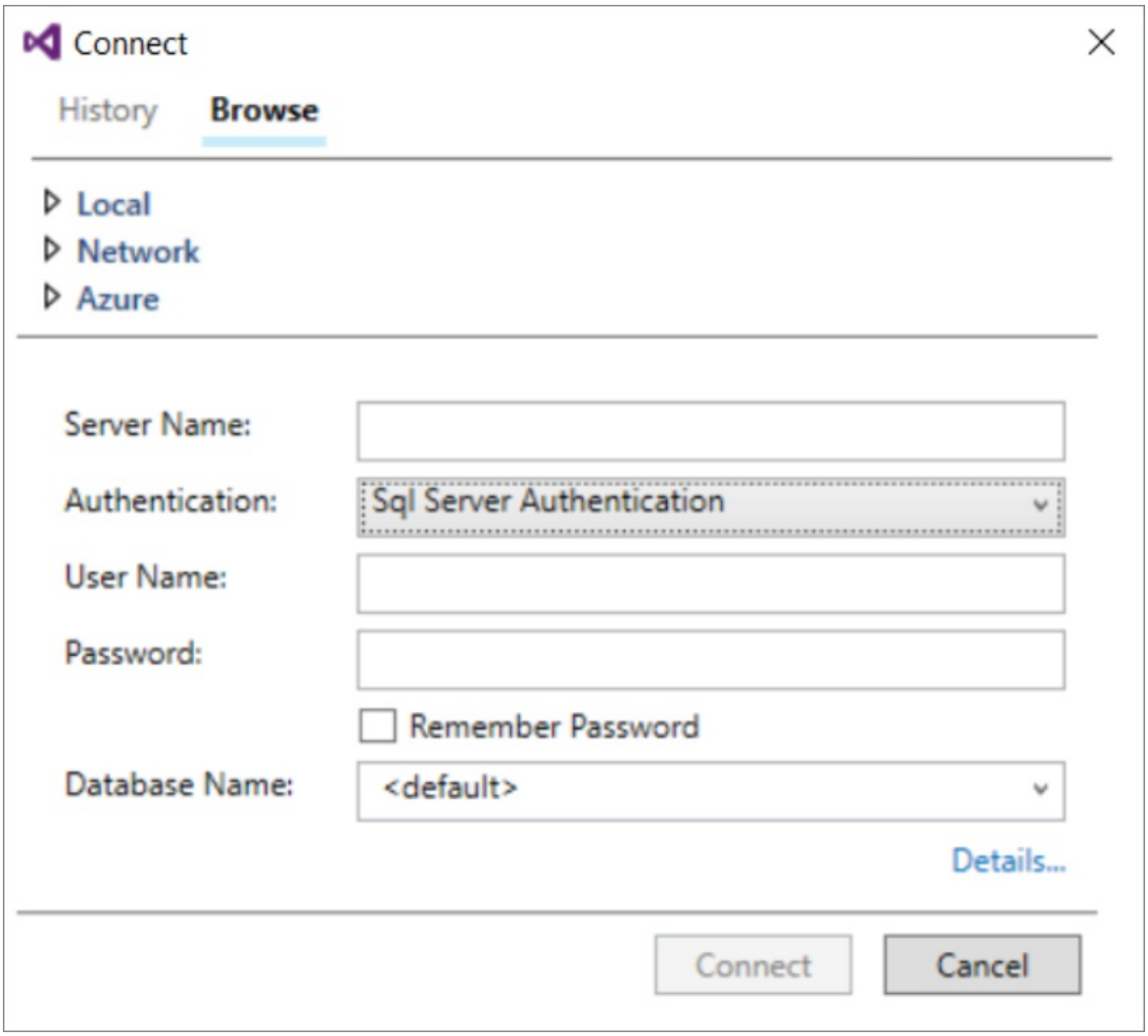
1. Open Visual Studio 2019.
2. Open the SQL Server Object Explorer by selecting **View** > **SQL Server Object Explorer**.



3. Click the **Add SQL Server** icon.

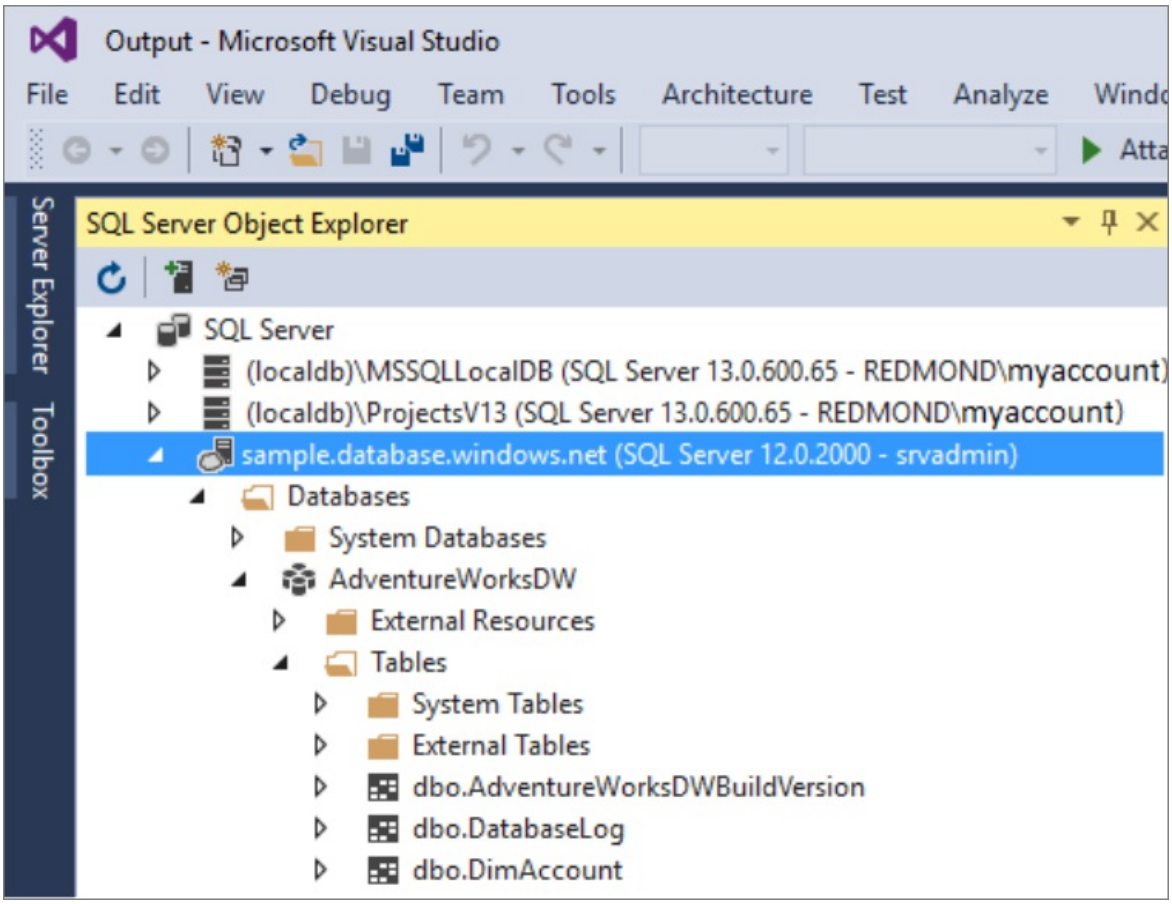


4. Fill in the fields in the Connect to Server window.



- **Server name**: Enter the **server name** previously identified.
- **Authentication**: Select **SQL Server Authentication** or **Active Directory Integrated Authentication**:
- **User Name** and **Password**: Enter your user name and password if SQL Server Authentication was selected above.
- Click **Connect**.

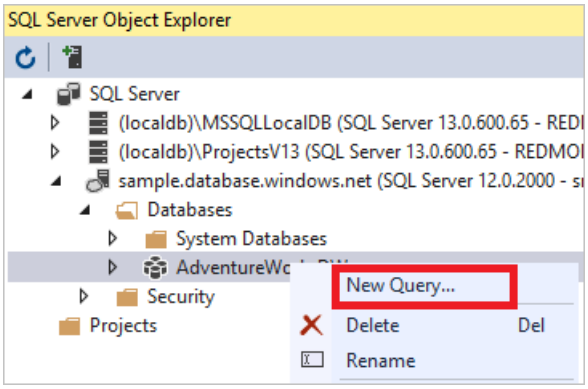
5. To explore, expand your Azure SQL server. You can view the databases associated with the server. Expand AdventureWorksDW to see the tables in your sample database.



2. Run a sample query

Now that a connection has been established to your database, you'll write a query.

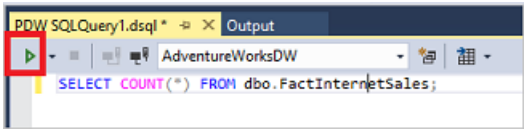
1. Right-click your database in SQL Server Object Explorer.
2. Select **New Query**. A new query window opens.



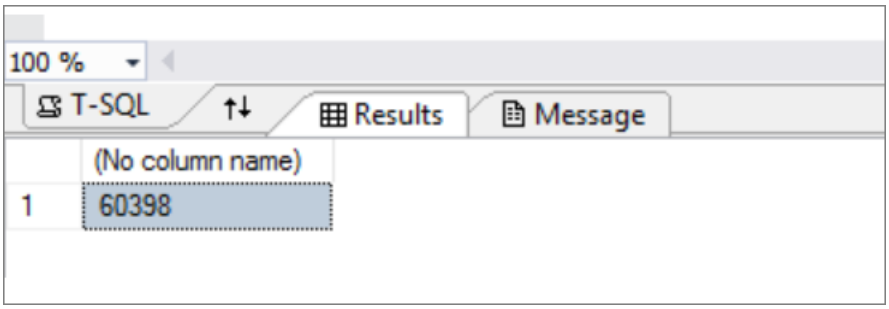
3. Copy the following T-SQL query into the query window:

```
SELECT COUNT(*) FROM dbo.FactInternetSales;
```

4. Run the query by clicking the green arrow or use the following shortcut: `CTRL` + `SHIFT` + `E`.



5. Look at the query results. In this example, the FactInternetSales table has 60398 rows.



Next steps

Now that you can connect and query, try visualizing the data with Power BI. To configure your environment for Azure Active Directory authentication, see Authenticate to dedicated SQL pool.

Connect to Synapse SQL with sqlcmd

8/19/2022 • 2 minutes to read • Edit Online

You can use the sqlcmd command-line utility to connect to and query serverless SQL pool and dedicated SQL pool within Synapse SQL.

1. Connect

To get started with sqlcmd, open the command prompt and enter **sqlcmd** followed by the connection string for your Synapse SQL database. The connection string requires the following parameters:

- **Server (-S):** Server in the form `<Server Name>`.database.windows.net
- **Database (-d):** Database name
- **Enable Quoted Identifiers (-I):** Quoted identifiers must be enabled to connect to a Synapse SQL instance

To use SQL Server Authentication, you need to add the username and password parameters:

- **User (-U):** Server user in the form `<User>`
- **Password (-P):** Password associated with the user

Your connection string might look like the following example:

Serverless SQL pool

```
C:\>sqlcmd -S partyeunrt.database.windows.net -d demo -U Enter_Your_Username_Here -P
Enter_Your_Password_Here -I
```

Dedicated SQL pool

```
C:\>sqlcmd -S MySqlDw.database.windows.net -d Adventure_Works -U myuser -P myP@ssword -I
```

To use Azure Active Directory Integrated authentication, you need to add the Azure Active Directory parameters:

- **Azure Active Directory Authentication (-G):** use Azure Active Directory for authentication

Your connection string might look like on of the following examples:

Serverless SQL pool

```
C:\>sqlcmd -S partyeunrt.database.windows.net -d demo -G -I
```

Dedicated SQL pool

```
C:\>sqlcmd -S MySqlDw.database.windows.net -d Adventure_Works -G -I
```

> **NOTE**
> You need to enable Azure Active Directory Authentication to authenticate using Active Directory.

2. Query

Use dedicated SQL pool

After connection, you can issue any supported Transact-SQL (T-SQL) statements against the instance. In this example, queries are submitted in interactive mode:

```
C:\>sqlcmd -S MySqlDw.database.windows.net -d Adventure_Works -U myuser -P myP@ssword -I
1> SELECT name FROM sys.tables;
2> GO
3> QUIT
```

For dedicated SQL pool, the following examples show you how to run queries in batch mode using the -Q option or piping your SQL to sqlcmd:

```
sqlcmd -S MySqlDw.database.windows.net -d Adventure_Works -U myuser -P myP@ssword -I -Q "SELECT name FROM
sys.tables;"
```

```
"SELECT name FROM sys.tables;" | sqlcmd -S MySqlDw.database.windows.net -d Adventure_Works -U myuser -P
myP@ssword -I > .\tables.out
```

Use serverless SQL pool

After connecting, you can issue any supported Transact-SQL (T-SQL) statements against the instance. In the following example, queries are submitted in interactive mode:

```
C:\>sqlcmd -S partyeunrt.database.windows.net -d demo -U Enter_Your_Username_Here -P
Enter_Your_Password_Here -I
1> SELECT COUNT(*) FROM  OPENROWSET(BULK
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us_population_county/year=20
*/*.parquet', FORMAT='PARQUET')
2> GO
3> QUIT
```

For serverless SQL pool, the examples that follow show you how to run queries in batch mode using the -Q option or piping your SQL to sqlcmd:

```
sqlcmd -S partyeunrt.database.windows.net -d demo -U Enter_Your_Username_Here -P 'Enter_Your_Password_Here'
-I -Q "SELECT COUNT(*) FROM  OPENROWSET(BULK
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us_population_county/year=20
*/*.parquet', FORMAT='PARQUET')"
```

```
"SELECT COUNT(*) FROM  OPENROWSET(BULK
'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us_population_county/year=20
*/*.parquet', FORMAT='PARQUET')" | sqlcmd -S partyeunrt.database.windows.net -d demo -U
Enter_Your_Username_Here -P 'Enter_Your_Password_Here' -I > ./tables.out
```

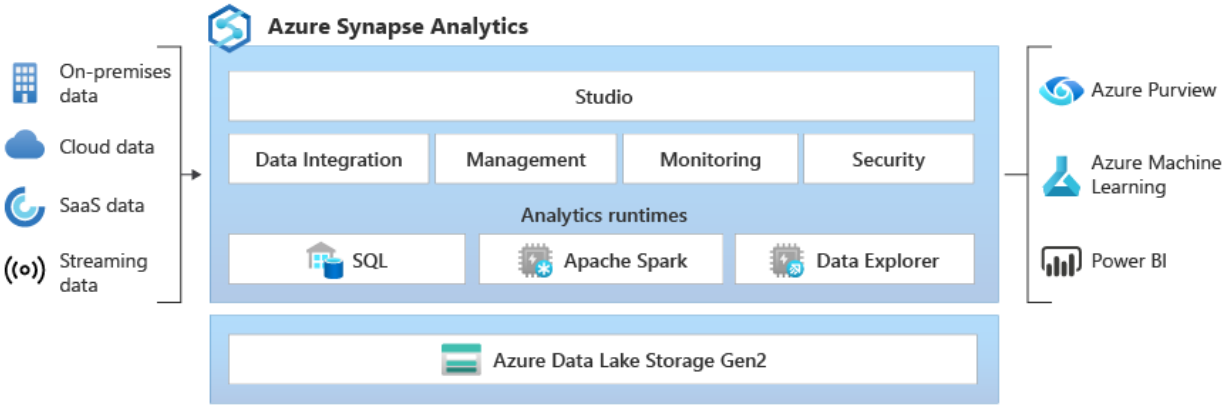
Next steps

For more information about sqlcmd options, see the sqlcmd documentation.

What is Azure Synapse Data Explorer? (Preview)

8/19/2022 • 5 minutes to read • Edit Online

Azure Synapse Data Explorer provides customers with an interactive query experience to unlock insights from log and telemetry data. To complement existing SQL and Apache Spark analytics runtime engines, the Data Explorer analytics runtime is optimized for efficient log analytics using powerful indexing technology to automatically index free-text and semi-structured data commonly found in telemetry data.



To learn more, see the following video:

What makes Azure Synapse Data Explorer unique?

- **Easy ingestion** - Data Explorer offers built-in integrations for no-code/low-code, high-throughput data ingestion, and caching data from real-time sources. Data can be ingested from sources such as Event Hub, Kafka, Azure Data Lake, open source agents such as Fluentd/Fluent Bit, and a wide variety of cloud and on-premises data sources.
- **No complex data modeling** - With Data Explorer, there is no need to build complex data models and no need for complex scripting to transform data before it's consumed.
- **No index maintenance** - There is no need for maintenance tasks to optimize data for query performance and no need for index maintenance. With Data Explorer, all raw data is available immediately, allowing you to run high-performance and high-concurrency queries on your streaming and persistent data. You can use these queries to build near real-time dashboards and alerts, and connect operational analytics data with the rest of data analytics platform.
- **Democratizing data analytics** - Data Explorer democratizes self-service, big data analytics with the intuitive Kusto Query Language (KQL) that provides the expressiveness and power of SQL with the simplicity of Excel. KQL is highly optimized for exploring raw telemetry and time series data by leveraging Data Explorer's best-in-class text indexing technology for efficient free-text and regex search, and comprehensive parsing capabilities for querying traces\text data and JSON semi-structured data including arrays and nested structures. KQL offers advanced time series support for creating, manipulating, and analyzing multiple time series with in-engine Python execution support for model scoring.
- **Proven technology at petabyte scale** - Data Explorer is a distributed system with compute and

storage that can scale independently, enabling analytics on gigabytes or petabytes of data.

- **Integrated** - Azure Synapse Analytics provides interoperability across data between Data Explorer, Apache Spark, and SQL engines empowering data engineers, data scientists, and data analysts to easily, and securely, access and collaborate on the same data in the data lake.

When to use Azure Synapse Data Explorer?

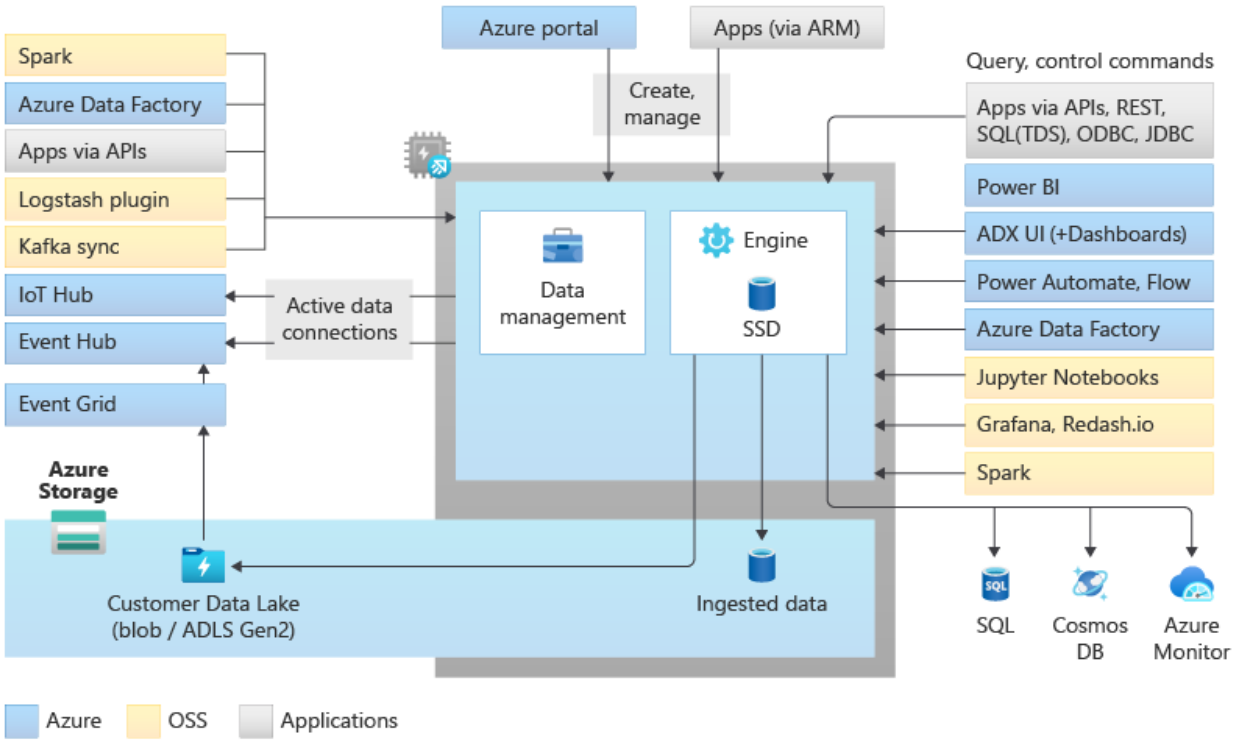
Use Data Explorer as a data platform for building near real-time log analytics and IoT analytics solutions to:

- Consolidate and correlate your logs and events data across on-premises, cloud, and third-party data sources.
- Accelerate your AI Ops journey (pattern recognition, anomaly detection, forecasting, and more).
- Replace infrastructure-based log search solutions to save cost and increase productivity.
- Build IoT analytics solutions for your IoT data.
- Build analytics SaaS solutions to offer services to your internal and external customers.

Data Explorer pool architecture

Data Explorer pools implement a scale out architecture by separating the compute and storage resources. This enables you to independently scale each resource and, for example, run multiple read only computes on the same data. Data Explorer pools consist of a set of computes running the engine that is responsible for automatically indexing, compressing, caching, and serving distributed queries. They also have a second set of computes running the data management service responsible for background system jobs, and managed and queued data ingestion. All data is persisted on managed blob storage accounts using a compressed columnar format.

Data Explorer pools support a rich ecosystem for ingesting data using connectors, SDKs, REST APIs, and other managed capabilities. It offers various ways to consume data for adhoc queries, reports, dashboards, alerts, REST APIs, and SDKs.



There are many unique capabilities that makes Data Explore the best analytical engine for log and time series

analytics on Azure. If you are interested in learning more about how Data Explorer works, see Azure Data Explorer white paper.

The following sections highlight the key differentiators.

Free-text and semi-structured data indexing enables near real time high performance and high concurrent queries

Data Explorer indexes semi-structured data (JSON) and unstructured data (free text) that makes running queries very performant on this type of data. By default, every field is indexed during the data ingestion with the option to use a low-level encoding policy to fine tune or disable the index for specific fields. The scope of the index is a single data shard.

The implementation of the index depends on the type of the field, as follows:

FIELD TYPE	INDEXING IMPLEMENTATION
String	The engine builds an inverted term index for string column values. Each string value is analyzed and split into normalized terms and an ordered list of logical positions, containing record ordinals, is recorded for each term. The resulting sorted list of terms and their associated positions is stored as an immutable B-tree.
Numeric **Datelime** **TimeSpan**	The engine builds a simple range-based forward index. The index records the min/max values for each block, for a group of blocks and for the entire column within the data shard.
Dynamic	The ingestion process enumerates all "atomic" elements within the dynamic value, such as property names, values, and array elements, and forwards them to the index builder. Dynamic fields have the same inverted term index as string fields.

These efficient indexing capabilities enables Data Explore to make the data available in near-real-time for high-performance and high-concurrency queries. The system automatically optimizes data shards to further boost performance.

Kusto Query Language

KQL has a large, growing community with the rapid adoption of Azure Monitor Log Analytics and Application Insights, Microsoft Sentinel, Azure Data Explorer, and other Microsoft offerings. The language is well designed with an easy-to-read syntax and provides a smooth transition from simple one-liner to complex data processing queries. This allows Data Explorer to provide rich Intellisense support and a rich set of language construct and built-in capabilities for aggregations, time series, and user analytics that aren't available in SQL for rapid exploration of telemetry data.

Next Steps

- Create a Data Explorer pool using Synapse Studio
- Create a Data Explorer pool using the Azure portal (Preview)

What is the difference between Azure Synapse Data Explorer and Azure Data Explorer? (Preview)

8/19/2022 • 2 minutes to read • Edit Online

Azure Data Explorer is a stand-alone, fast, and highly scalable data exploration service for log and telemetry data. The same underlying technology that runs the service is available in Azure Synapse as an integrated analytics service to complement its existing SQL and Spark services geared for data warehouse and data engineering machine learning scenarios.

The native integration of Data Explore capabilities brings analytics log and time series workloads to Synapse in the form of Synapse Data Explorer pools. These pools offer close integration with, and easier management of connectivity to, other Synapse capabilities. Having all your machine and user data available and accessible in one place, and by leveraging the near real-time exploratory capabilities of Data Explorer, enables you to build solutions that drive business value.

We recommend starting with Synapse Data Explorer if you are looking for a unified solution that combines big data and analytics capabilities.

Capability support

CATEGORY	CAPABILITY	AZURE DATA EXPLORER	SYNAPSE DATA EXPLORER
Security	VNET	Supports VNet Injection and Azure Private Link	Support for Azure Private link automatically integrated as part of Synapse Managed VNET
	CMK	✓	Automatically inherited from Synapse workspace configuration
	Firewall	✗	Automatically inherited from Synapse workspace configuration
Business Continuity	Availability Zones	Optional	Enabled by default where Availability Zones are available
SKU	Compute options	22+ Azure VM SKUs to choose from	Simplified to Synapse workload types SKUs
Integrations	Built-in ingestion pipelines	Event Hub, Event Grid, IoT Hub	Event Hub, Event Grid, and IoT Hub supported via the Azure portal for non-managed VNet

CATEGORY	CAPABILITY	AZURE DATA EXPLORER	SYNAPSE DATA EXPLORER
	Spark integration	Azure Data Explorer linked service: Built-in Kusto Spark integration with support for Azure Active Directory pass-though authentication, Synapse Workspace MSI, and Service Principal	Built-in Kusto Spark connector integration with support for Azure Active Directory pass-though authentication, Synapse Workspace MSI, and Service Principal
	KQL artifacts management	✗	Save KQL queries and integrate with Git
	Metadata sync	✗	✗
Features	KQL queries	✓	✓
	API and SDKs	✓	✓
	Connectors	✓	✓
	Query tools	✓	✓
Pricing	Business Model	Cost plus billing model with multiple meters: Azure Data Explorer IP markup, Compute, Storage, and Networking	VCore billing model with two meters: VCore and Storage
	Reserved Instances	✓	✗

Next steps

Data Explorer in Azure Synapse Analytics

Azure Synapse Data Explorer data ingestion overview (Preview)

8/19/2022 • 5 minutes to read • Edit Online

Data ingestion is the process used to load data records from one or more sources to import data into a table in Azure Synapse Data Explorer pool. Once ingested, the data becomes available for query.

The Azure Synapse Data Explorer data management service, which is responsible for data ingestion, implements the following process:

- Pulls data in batches or streaming from an external source and reads requests from a pending Azure queue.
- Batch data flowing to the same database and table is optimized for ingestion throughput.
- Initial data is validated and the format is converted where necessary.
- Further data manipulation including matching schema, organizing, indexing, encoding, and compressing the data.
- Data is persisted in storage according to the set retention policy.
- Ingested data is committed into the engine, where it's available for query.

Supported data formats, properties, and permissions

- **Supported data formats**
- **Ingestion properties**: The properties that affect how the data will be ingested (for example, tagging, mapping, creation time).
- **Permissions**: To ingest data, the process requires database ingestor level permissions. Other actions, such as query, may require database admin, database user, or table admin permissions.

Batching vs streaming ingestions

- Batching ingestion does data batching and is optimized for high ingestion throughput. This method is the preferred and most performant type of ingestion. Data is batched according to ingestion properties. Small batches of data are merged and optimized for fast query results. The ingestion batching policy can be set on databases or tables. By default, the maximum batching value is 5 minutes, 1000 items, or a total size of 1 GB. The data size limit for a batch ingestion command is 4 GB.
- Streaming ingestion is ongoing data ingestion from a streaming source. Streaming ingestion allows near real-time latency for small sets of data per table. Data is initially ingested to row store, then moved to column store extents.

Ingestion methods and tools

Azure Synapse Data Explorer supports several ingestion methods, each with its own target scenarios. These methods include ingestion tools, connectors and plugins to diverse services, managed pipelines, programmatic ingestion using SDKs, and direct access to ingestion.

Ingestion using managed pipelines

For organizations who wish to have management (throttling, retries, monitors, alerts, and more) done by an external service, using a connector is likely the most appropriate solution. Queued ingestion is appropriate for large data volumes. Azure Synapse Data Explorer supports the following Azure Pipelines:

- **Event Hub**: A pipeline that transfers events from services to Azure Synapse Data Explorer. For more information, see Ingest data from Event Hub into Azure Synapse Data Explorer.

- **Synapse pipelines**: A fully managed data integration service for analytic workloads in Synapse pipelines connects with over 90 supported sources to provide efficient and resilient data transfer. Synapse pipelines prepares, transforms, and enriches data to give insights that can be monitored in different kinds of ways. This service can be used as a one-time solution, on a periodic timeline, or triggered by specific events.

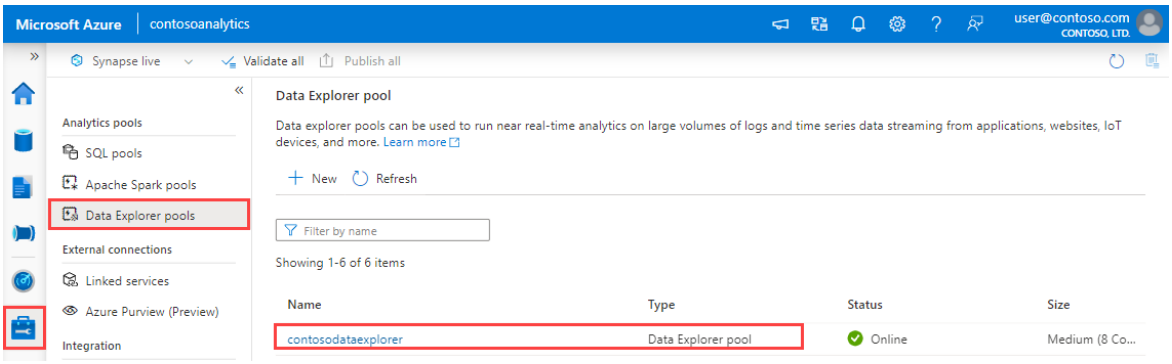
Programmatic ingestion using SDKs

Azure Synapse Data Explorer provides SDKs that can be used for query and data ingestion. Programmatic ingestion is optimized for reducing ingestion costs (COGs), by minimizing storage transactions during and following the ingestion process.

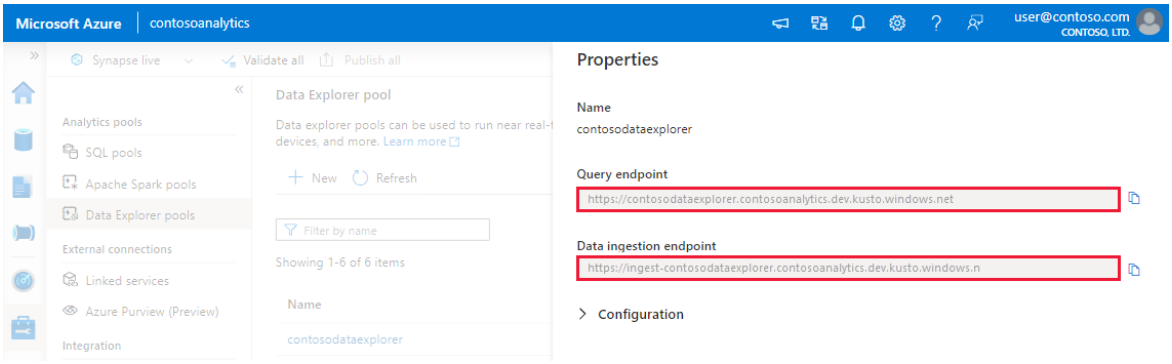
Before you start, use the following steps to get the Data Explorer pool endpoints for configuring programmatic ingestion.

1. In Synapse Studio, on the left-side pane, select **Manage** > **Data Explorer pools**.

2. Select the Data Explorer pool you want to use to view its details.



3. Make a note of the Query and Data Ingestion endpoints. Use the Query endpoint as the cluster when configuring connections to your Data Explorer pool. When configuring SDKs for data ingestion, use the data ingestion endpoint.



Available SDKs and open-source projects

- Python SDK
- .NET SDK
- Java SDK
- Node SDK
- REST API
- GO SDK

Tools

- **One-click ingestion**: Enables you to quickly ingest data by creating and adjusting tables from a wide range of source types. One-click ingestion automatically suggests tables and mapping structures based on the data source in Azure Synapse Data Explorer. One-click ingestion can be used for one-time ingestion, or to define continuous ingestion via Event Grid on the container to which the data was ingested.

Kusto Query Language ingest control commands

There are a number of methods by which data can be ingested directly to the engine by Kusto Query Language (KQL) commands. Because this method bypasses the Data Management services, it's only appropriate for exploration and prototyping. Don't use this method in production or high-volume scenarios.

- **Inline ingestion**: A control command .ingest inline is sent to the engine, with the data to be ingested being a part of the command text itself. This method is intended for improvised testing purposes.
- **Ingest from query**: A control command .set, .append, .set-or-append, or .set-or-replace is sent to the engine, with the data specified indirectly as the results of a query or a command.
- **Ingest from storage (pull)**: A control command .ingest into is sent to the engine, with the data stored in some external storage (for example, Azure Blob Storage) accessible by the engine and pointed-to by the command.

For an example of using ingest control commands, see Analyze with Data Explorer.

Ingestion process

Once you have chosen the most suitable ingestion method for your needs, do the following steps:

1. **Set retention policy**

   Data ingested into a table in Azure Synapse Data Explorer is subject to the table's effective retention policy. Unless set on a table explicitly, the effective retention policy is derived from the database's retention policy. Hot retention is a function of cluster size and your retention policy. Ingesting more data than you have available space will force the first in data to cold retention.

   Make sure that the database's retention policy is appropriate for your needs. If not, explicitly override it at the table level. For more information, see retention policy.

2. **Create a table**

   In order to ingest data, a table needs to be created beforehand. Use one of the following options:

   - Create a table with a command. For an example of using the create a table command, see Analyze with Data Explorer.
   - Create a table using One-click Ingestion.

   > **NOTE**
   > If a record is incomplete or a field cannot be parsed as the required data type, the corresponding table columns will be populated with null values.

3. **Create schema mapping**

   Schema mapping helps bind source data fields to destination table columns. Mapping allows you to take data from different sources into the same table, based on the defined attributes. Different types of mappings are supported, both row-oriented (CSV, JSON and AVRO), and column-oriented (Parquet). In most methods, mappings can also be pre-created on the table and referenced from the ingest command

parameter.

4. **Set update policy** (optional)

   Some of the data format mappings (Parquet, JSON, and Avro) support simple and useful ingest-time transformations. Where the scenario requires more complex processing at ingest time, use update policy, which allows for lightweight processing using Kusto Query Language commands. The update policy automatically runs extractions and transformations on ingested data on the original table, and ingests the resulting data into one or more destination tables. Set your update policy.

Next steps

- Supported data formats
- Supported ingestion properties

Azure Synapse Data Explorer data ingestion properties (Preview)

8/19/2022 • 3 minutes to read • Edit Online

Data ingestion is the process by which data is added to a table and is made available for query in Data Explorer. You add properties to the ingestion command after the `with` keyword.

Ingestion properties

The following table lists the properties supported by Data Explorer, describes them, and provides examples:

PROPERTY	DESCRIPTION	EXAMPLE
`ingestionMapping`	A string value that indicates how to map data from the source file to the actual columns in the table. Define the `format` value with the relevant mapping type. See data mappings.	`with (format="json", ingestionMapping = "[{\"column\":\"rownumber\", \"Properties\": {\"Path\":\"$.RowNumber\"}}, {\"column\":\"rowguid\", \"Properties\": {\"Path\":\"$.RowGuid\"}}]")` (deprecated: `avroMapping`, `csvMapping`, `jsonMapping`)
`ingestionMappingReference`	A string value that indicates how to map data from the source file to the actual columns in the table using a named mapping policy object. Define the `format` value with the relevant mapping type. See data mappings.	`with (format="csv", ingestionMappingReference = "Mapping1")` (deprecated: `avroMappingReference`, `csvMappingReference`, `jsonMappingReference`)
`creationTime`	The datetime value (formatted as an ISO8601 string) to use at the creation time of the ingested data extents. If unspecified, the current value (`now()`) will be used. Overriding the default is useful when ingesting older data, so that the retention policy will be applied correctly. When specified, make sure the `Lookback` property in the target table's effective Extents merge policy is aligned with the specified value.	`with (creationTime="2017-02-13")`
`extend_schema`	A Boolean value that, if specified, instructs the command to extend the schema of the table (defaults to `false`). This option applies only to `.append` and `.set-or-append` commands. The only allowed schema extensions have additional columns added to the table at the end.	If the original table schema is `(a:string, b:int)` , a valid schema extension would be `(a:string, b:int, c:datetime, d:string)` , but `(a:string, c:datetime)` wouldn't be valid

PROPERTY	DESCRIPTION	EXAMPLE
`folder`	For ingest-from-query commands, the folder to assign to the table. If the table already exists, this property will override the table's folder.	`with (folder="Tables/Temporary")`
`format`	The data format (see supported data formats).	`with (format="csv")`
`ingestIfNotExists`	A string value that, if specified, prevents ingestion from succeeding if the table already has data tagged with an `ingest-by:` tag with the same value. This ensures idempotent data ingestion. For more information, see ingest-by: tags.	The properties `with (ingestIfNotExists='["Part0001"]', tags='["ingest-by:Part0001"]')` indicate that if data with the tag `ingest-by:Part0001` already exists, then don't complete the current ingestion. If it doesn't already exist, this new ingestion should have this tag set (in case a future ingestion attempts to ingest the same data again.)
`ignoreFirstRecord`	A Boolean value that, if set to `true`, indicates that ingestion should ignore the first record of every file. This property is useful for files in `csv` and similar formats, if the first record in the file are the column names. By default, `false` is assumed.	`with (ignoreFirstRecord=false)`
`persistDetails`	A Boolean value that, if specified, indicates that the command should persist the detailed results (even if successful) so that the .show operation details command could retrieve them. Defaults to `false`.	`with (persistDetails=true)`
`policy_ingestiontime`	A Boolean value that, if specified, describes whether to enable the Ingestion Time Policy on a table that is created by this command. The default is `true`.	`with (policy_ingestiontime=false)`
`recreate_schema`	A Boolean value that, if specified, describes whether the command may recreate the schema of the table. This property applies only to the `.set-or-replace` command. This property takes precedence over the `extend_schema` property if both are set.	`with (recreate_schema=true)`
`tags`	A list of tags to associate with the ingested data, formatted as a JSON string	`with (tags="['Tag1', 'Tag2']")`
`validationPolicy`	A JSON string that indicates which validations to run during ingestion. See Data ingestion for an explanation of the different options.	`with (validationPolicy='{"ValidationOptions":1, "ValidationImplications":1}')` (this is actually the default policy)

PROPERTY	DESCRIPTION	EXAMPLE
`zipPattern`	Use this property when ingesting data from storage that has a ZIP archive. This is a string value indicating the regular expression to use when selecting which files in the ZIP archive to ingest. All other files in the archive will be ignored.	`with (zipPattern="*.csv")`

Next steps

- Learn more about data ingestion.
- Learn more about supported data formats.

Data formats supported by Azure Synapse Data Explorer for ingestion (Preview)

8/19/2022 • 2 minutes to read • Edit Online

Data ingestion is the process by which data is added to a table and is made available for query in Data Explorer. For all ingestion methods, other than ingest-from-query, the data must be in one of the supported formats. The following table lists and describes the formats that Data Explorer supports for data ingestion.

NOTE

Before you ingest data, make sure that your data is properly formatted and defines the expected fields. We recommend using your preferred validator to confirm the format is valid. For example, you may find the following validators useful to check CSV or JSON files:

- CSV: http://csvlint.io/
- JSON: https://jsonlint.com/

For more information about why ingestion might fail, see Ingestion failures and Ingestion error codes in Data Explorer.

FORMAT	EXTENSION	DESCRIPTION
ApacheAvro	`.avro`	An AVRO format with support for logical types. The following compression codecs are supported: `null`, `deflate`, and `snappy`. Reader implementation of the `apacheavro` format is based on the official Apache Avro library.
Avro	`.avro`	A legacy implementation for AVRO format based on .NET library. The following compression codecs are supported: `null`, `deflate` (for `snappy` - use `ApacheAvro` data format).
CSV	`.csv`	A text file with comma-separated values (`,`). See RFC 4180: *Common Format and MIME Type for Comma-Separated Values (CSV) Files*.
JSON	`.json`	A text file with JSON objects delimited by `\n` or `\r\n`. See JSON Lines (JSONL).

FORMAT	EXTENSION	DESCRIPTION
MultiJSON	`.multijson`	A text file with a JSON array of property bags (each representing a record), or any number of property bags delimited by whitespace, `\n` or `\r\n`. Each property bag can be spread on multiple lines. This format is preferred over `JSON`, unless the data is non-property bags.
ORC	`.orc`	An ORC file.
Parquet	`.parquet`	A Parquet file.
PSV	`.psv`	A text file with pipe-separated values (`|`).
RAW	`.raw`	A text file whose entire contents is a single string value.
SCsv	`.scsv`	A text file with semicolon-separated values (`;`).
SOHsv	`.sohsv`	A text file with SOH-separated values. (SOH is ASCII codepoint 1; this format is used by Hive on HDInsight.)
TSV	`.tsv`	A text file with tab-separated values (`\t`).
TSVE	`.tsv`	A text file with tab-separated values (`\t`). A backslash character (`\`) is used for escaping.
TXT	`.txt`	A text file with lines delimited by `\n`. Empty lines are skipped.
W3CLOGFILE	`.log`	Web log file format standardized by the W3C.

Supported data compression formats

Blobs and files can be compressed through any of the following compression algorithms:

COMPRESSION	EXTENSION
GZip	.gz
Zip	.zip

Indicate compression by appending the extension to the name of the blob or file.

For example:

- `MyData.csv.zip` indicates a blob or a file formatted as CSV, compressed with ZIP (archive or a single file)
- `MyData.json.gz` indicates a blob or a file formatted as JSON, compressed with GZip.

Blob or file names that don't include the format extensions but just compression (for example, `MyData.zip`) is also supported. In this case, the file format must be specified as an ingestion property because it cannot be inferred.

> **NOTE**
>
> - Some compression formats keep track of the original file extension as part of the compressed stream. This extension is generally ignored for determining the file format. If the file format can't be determined from the (compressed) blob or file name, it must be specified through the `format` ingestion property.
> - Not to be confused with internal (chunk level) compression codec used by `Parquet`, `AVRO` and `ORC` formats. Internal compression name is usually added to a file name before file format extension, for example: `file1.gz.parquet`, `file1.snappy.avro`, etc.

Next steps

- Learn more about data ingestion
- Learn more about data ingestion properties

Quickstart: Create a Data Explorer pool using the Azure portal (Preview)

8/19/2022 • 2 minutes to read • Edit Online

Azure Synapse Data Explorer is a fast, fully managed data analytics service for real-time analysis on large volumes of data streaming from applications, websites, IoT devices, and more. To use Data Explorer, you first create a Data Explorer pool.

This article describes the steps to create a Data Explorer pool in a Synapse workspace by using the Azure portal.

> **IMPORTANT**
> Billing for Data Explorer instances is prorated per minute, whether you are using them or not. Be sure to shutdown your Data Explorer instance after you have finished using it, or set a short timeout. For more information, see **Clean up resources**.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

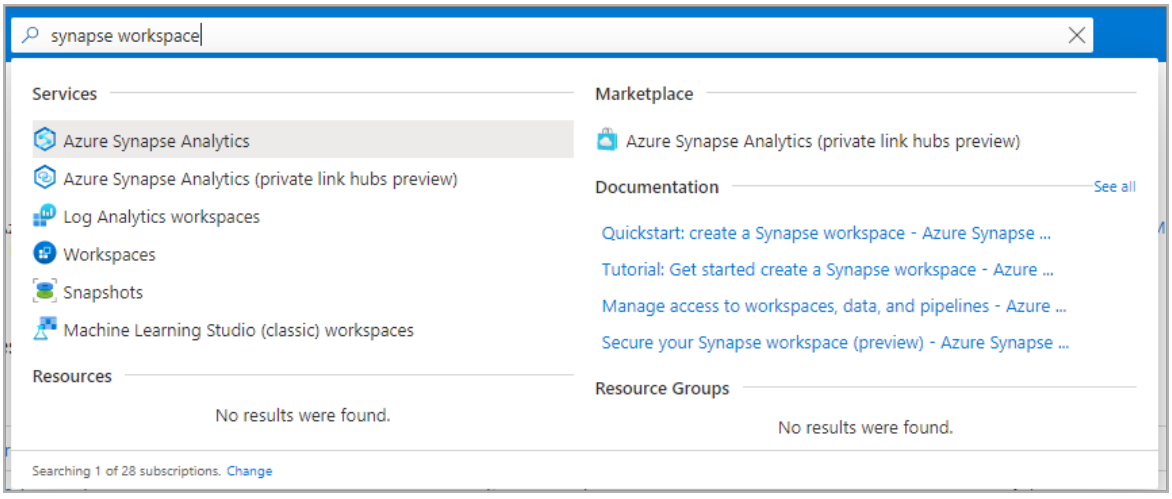
- Azure subscription - create one for free
- Synapse workspace

Sign in to the Azure portal

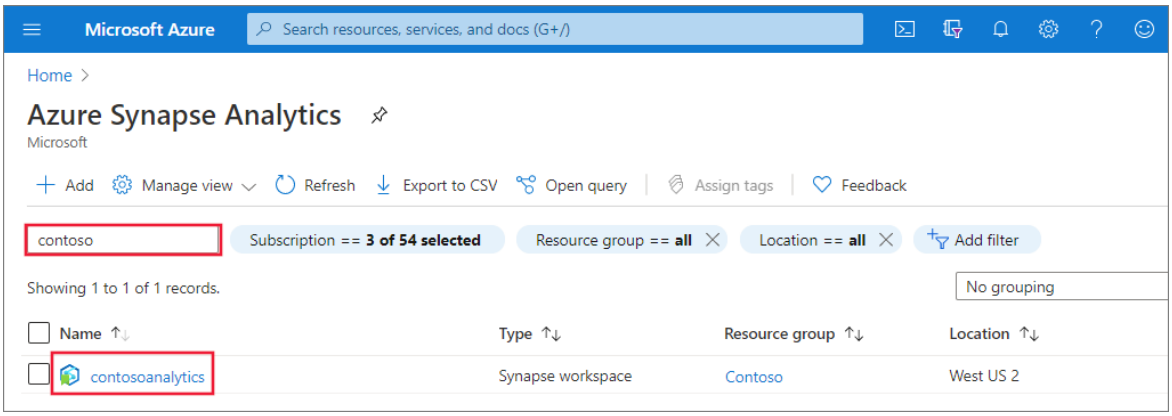
Sign in to the Azure portal

Navigate to the Synapse workspace

1. Navigate to the Synapse workspace where the Data Explorer pool will be created by typing the service name (or resource name directly) into the search bar.

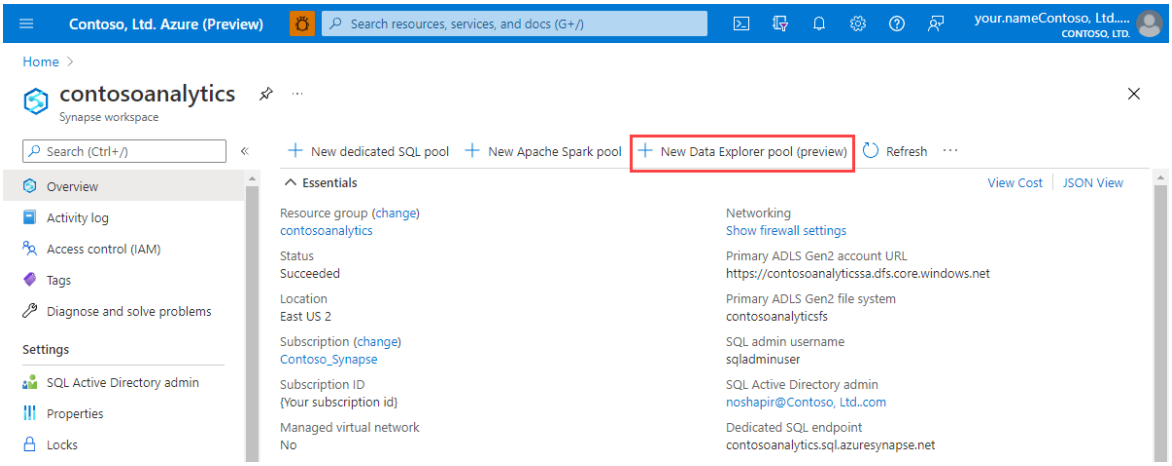


2. From the list of workspaces, type the name (or part of the name) of the workspace to open. For this example, we'll use a workspace named **contosoanalytics**.



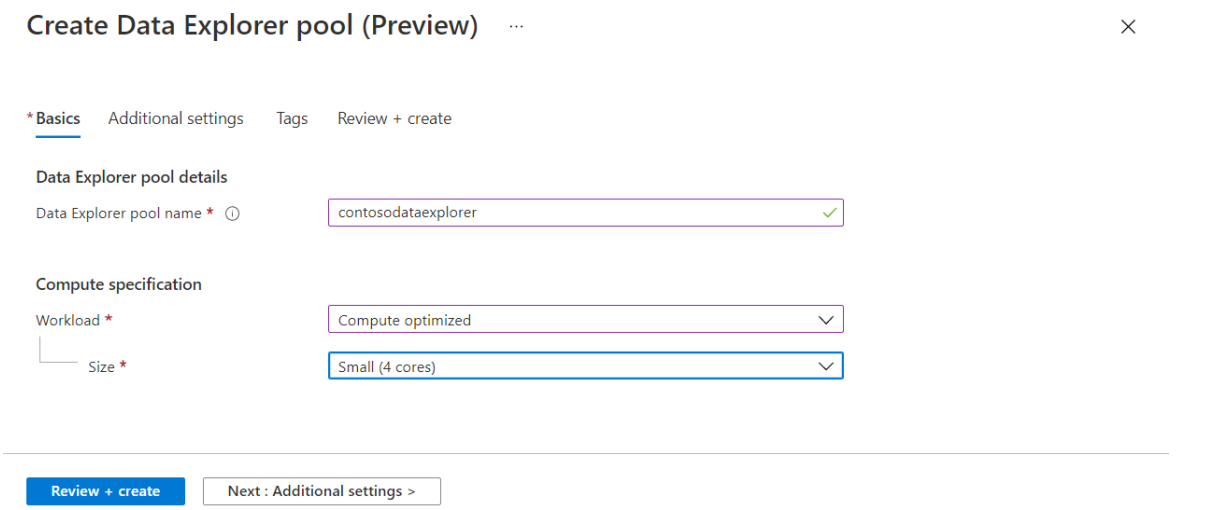
Create a new Data Explorer pool

1. In the Synapse workspace where you want to create the Data Explorer pool, select **New Data Explorer pool**.



2. Enter the following details in the **Basics** tab:

SETTING	SUGGESTED VALUE	DESCRIPTION
Data Explorer pool name	contosodataexplorer	This is the name that the Data Explorer pool will have.
Workload	Compute optimized	This workload provides a higher CPU to SSD storage ratio.
Node size	Small (4 cores)	Set this to the smallest size to reduce costs for this quickstart

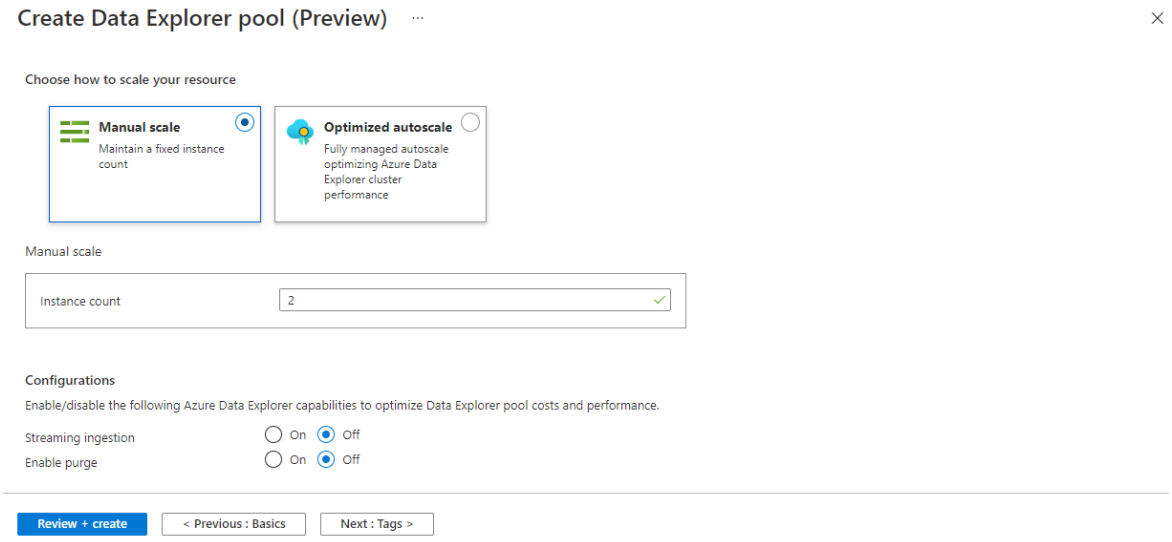


IMPORTANT

Note that there are specific limitations for the names that Data Explorer pools can use. Names must contain lowercase letters and numbers only, must be between 4 and 15 characters, and must start with a letter.

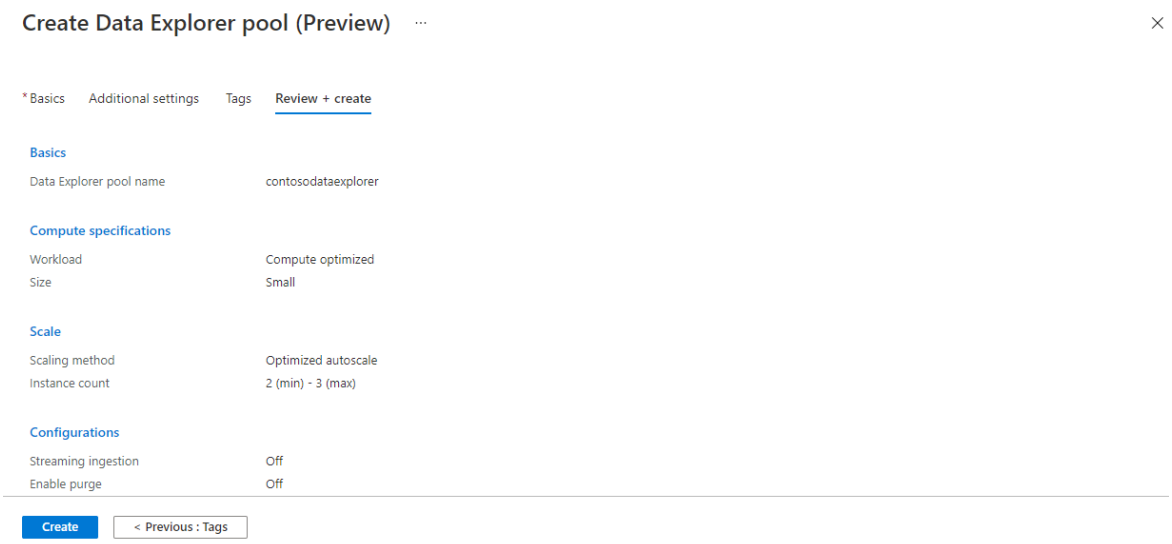
3. Select **Next: Additional settings**. Use the following settings and leave the defaults for the remaining settings.

SETTING	SUGGESTED VALUE	DESCRIPTION
Scaling	Manual scale	We won't need autoscale in this quickstart
Number count	2	Set this to the smallest size to reduce costs for this quickstart

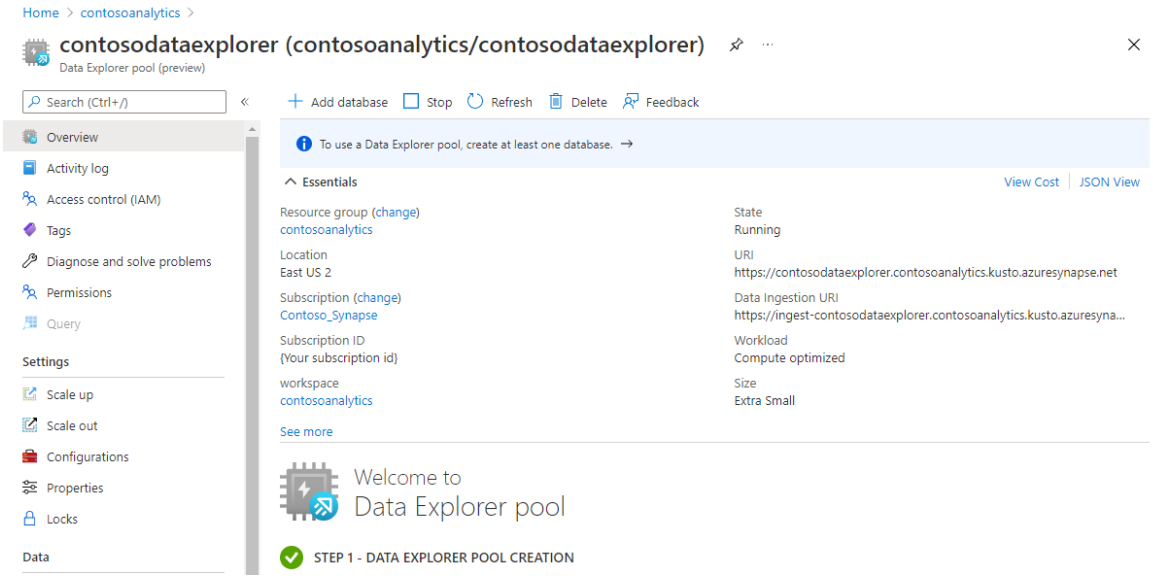


4. Select **Next: tags**. Don't add any tags.
5. Select **Review + create**.
6. Review the detail making sure they are correct, and then select **Create**.

   The Data Explorer pool will start the provisioning process.



7. Once the provisioning is complete, navigate back to the workspace **Overview** page and verify that the new Data Explorer pool welcome page appears.



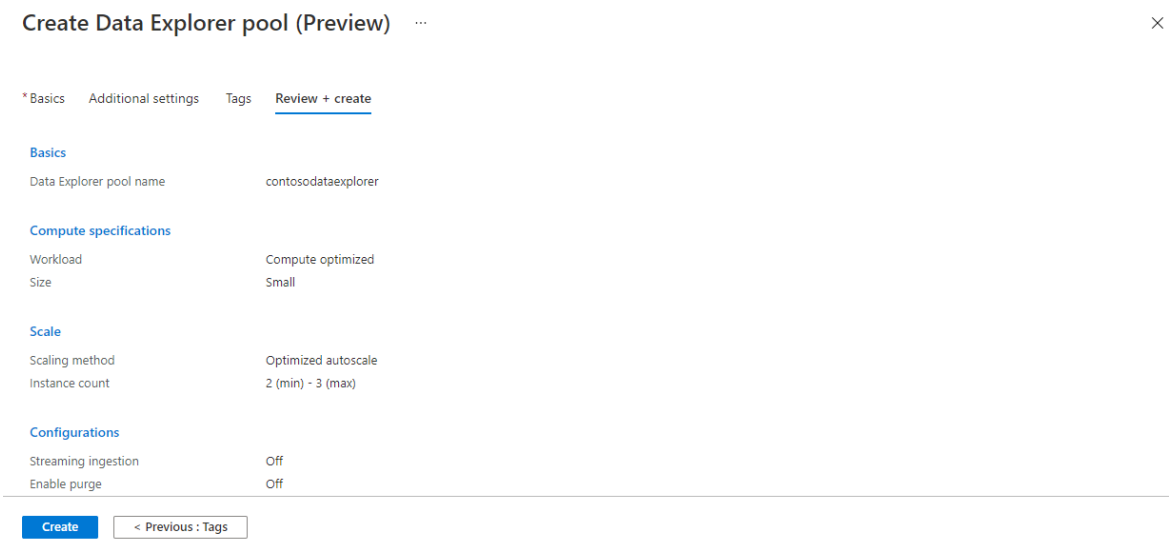
Clean up Data Explorer pool resources using the Azure portal

Use the following steps to delete the Data Explorer pool from the workspace using the Azure portal.

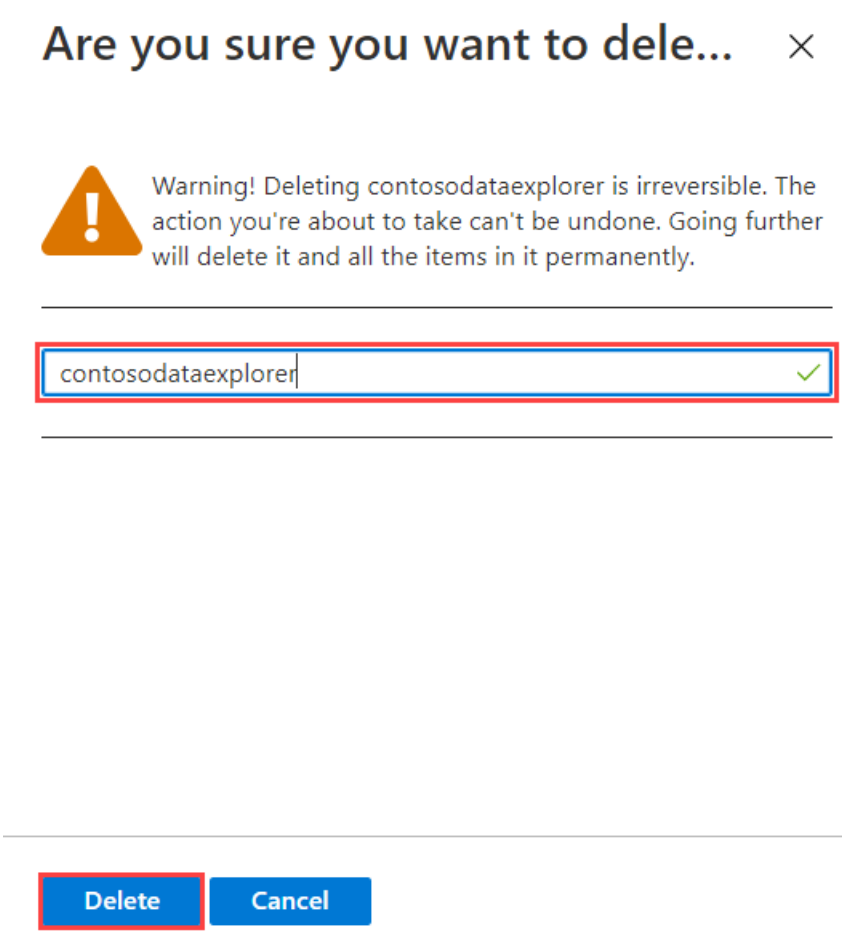
> **WARNING**
> Deleting a data-explorer pool will remove the analytics engine from the workspace. It will no longer be possible to connect to the pool, and all queries, pipelines, and notebooks that use the deleted pool will no longer work.

Delete the Data Explorer pool

1. Navigate to the Data Explorer pools in the workspace.
2. To remove the Data Explorer pool (in this case, **contosodataexplorer**), select **More [...]** > **Delete**.



3. To confirm the deletion, enter the name of the pool being deleted and then select **Delete**.



4. Once the process completes successfully, verify that the pool no longer appears in the list.

Next steps

- Ingestion overview
- Analyze with Data Explorer
- Monitor Data Explorer pools

Quickstart: Create a Data Explorer pool using Synapse Studio (Preview)

8/19/2022 • 3 minutes to read • Edit Online

Azure Synapse Data Explorer is a fast, fully managed data analytics service for real-time analysis on large volumes of data streaming from applications, websites, IoT devices, and more. To use Data Explorer, you first create a Data Explorer pool.

This quickstart describes the steps to create a Data Explorer pool in a Synapse workspace by using Synapse Studio.

> **IMPORTANT**
> Billing for Data Explorer instances is prorated per minute, whether you are using them or not. Be sure to shutdown your Data Explorer instance after you have finished using it, or set a short timeout. For more information, see **Clean up resources**.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

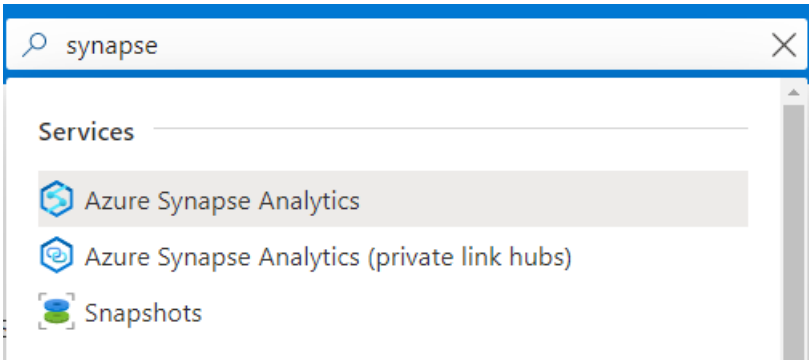
- Azure subscription - create one for free
- Synapse workspace

Sign in to the Azure portal

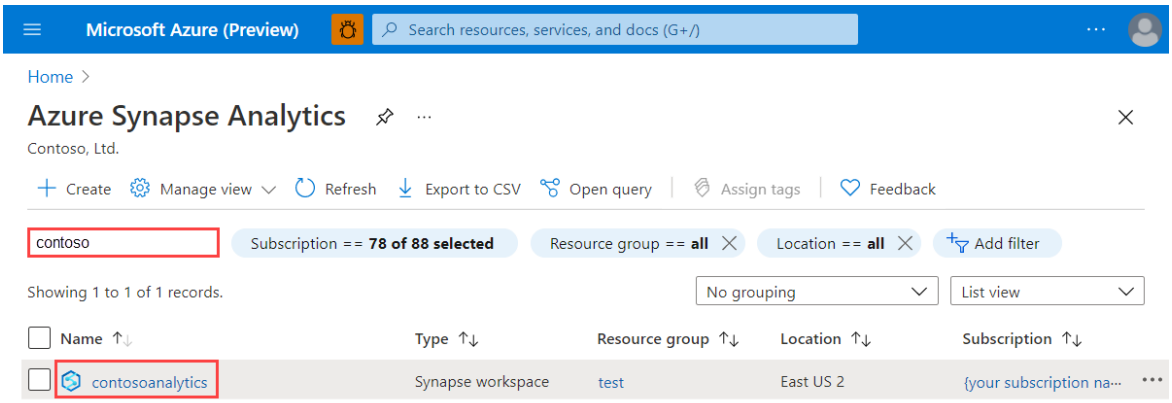
Sign in to the Azure portal

Navigate to the Synapse workspace

1. Navigate to the Synapse workspace where the Data Explorer pool will be created by typing the service name (or resource name directly) into the search bar.

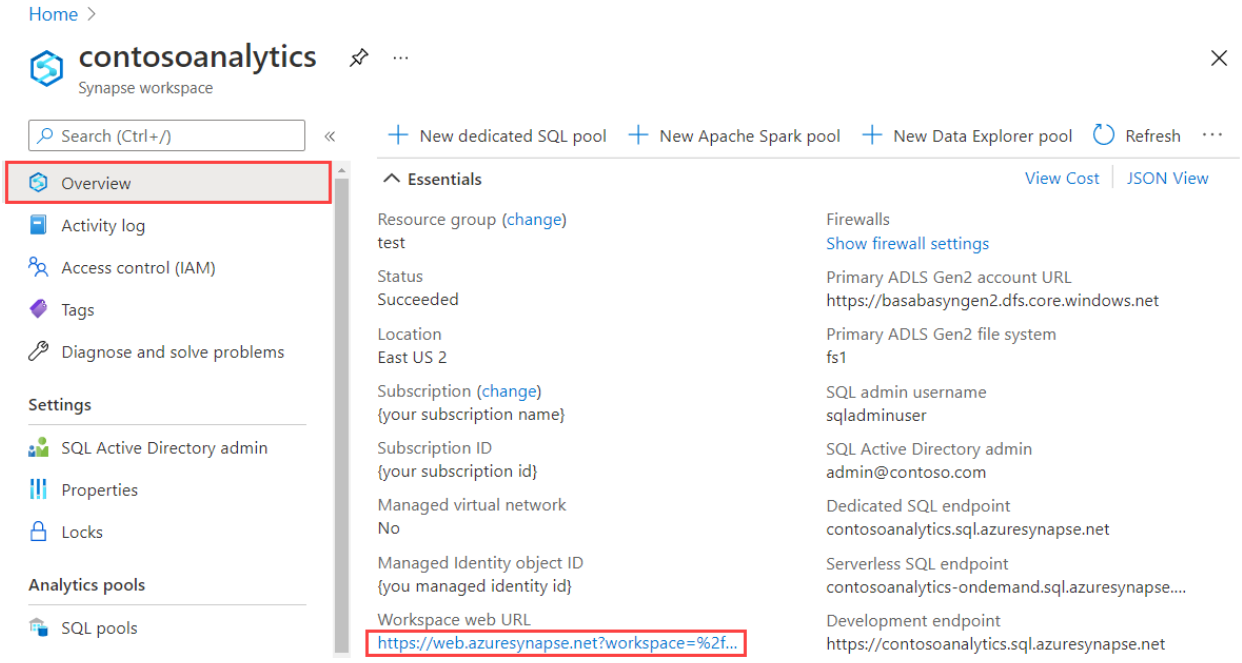


2. From the list of workspaces, type the name (or part of the name) of the workspace to open. For this example, we'll use a workspace named **contosoanalytics**.



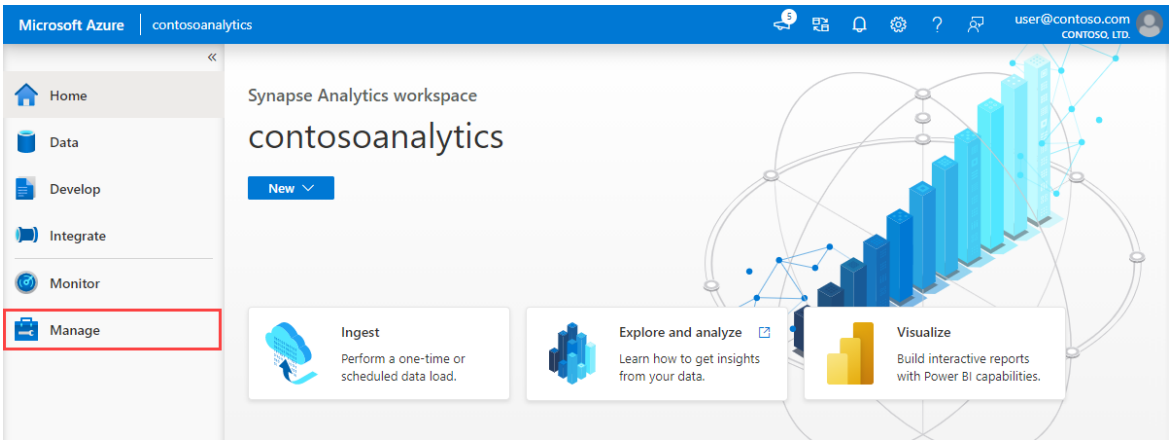
Launch Synapse Studio

From the workspace overview, select the **Workspace web URL** to open Synapse Studio.

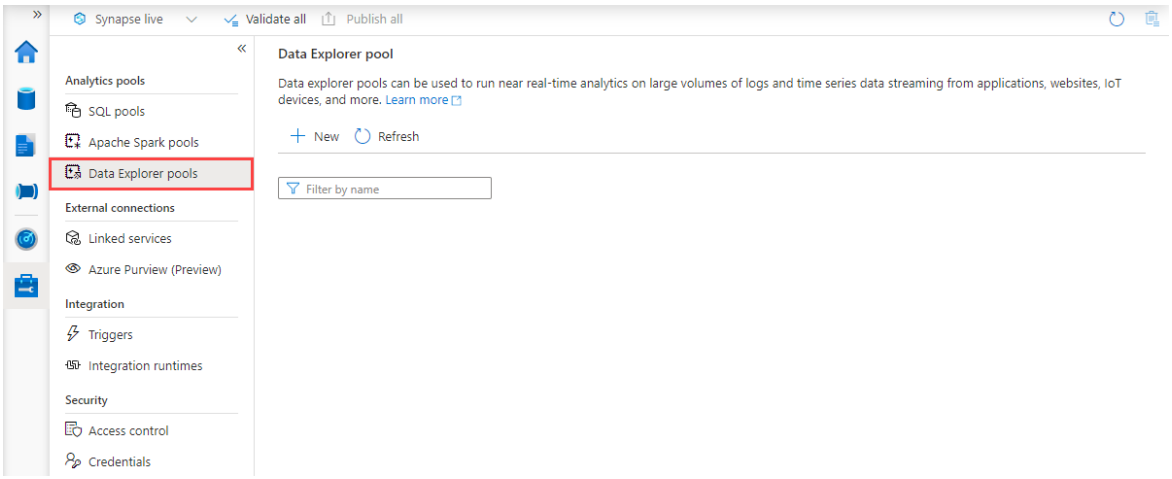


Create a new Data Explorer pool

1. On the Synapse Studio home page, navigate to the **Management Hub** in the left navigation by selecting the **Manage** icon.



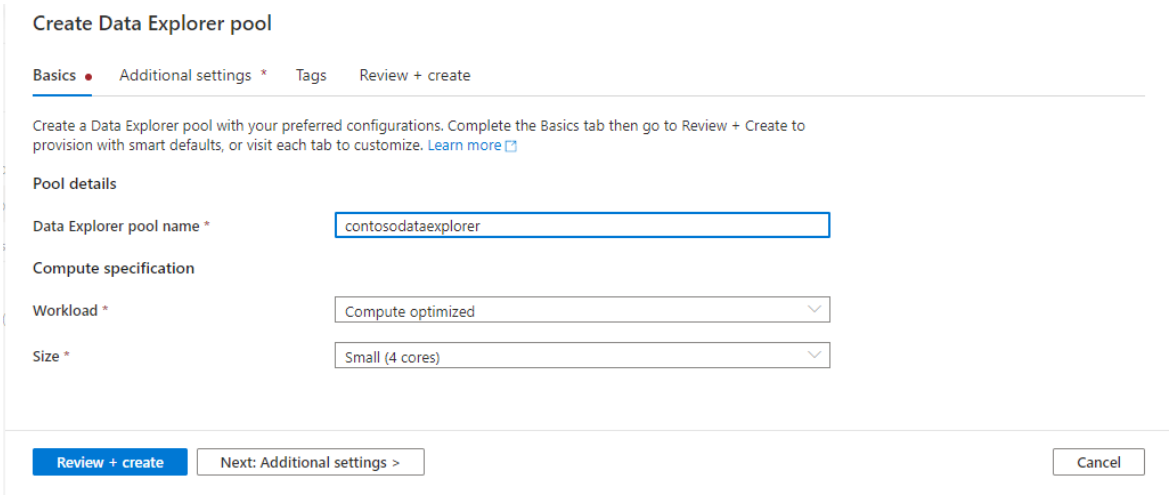
2. Once in the Management Hub, navigate to the **Data Explorer pools** section to see the current list of Data Explorer pools that are available in the workspace.



3. Select **+ New**. The new Data Explorer pool create wizard appears.

4. Enter the following details in the **Basics** tab:

SETTING	SUGGESTED VALUE	DESCRIPTION
Data Explorer pool name	contosodataexplorer	This is the name that the Data Explorer pool will have.
Workload	Compute optimized	This workload provides a higher CPU to SSD storage ratio.
Node size	Small (4 cores)	Set this to the smallest size to reduce costs for this quickstart

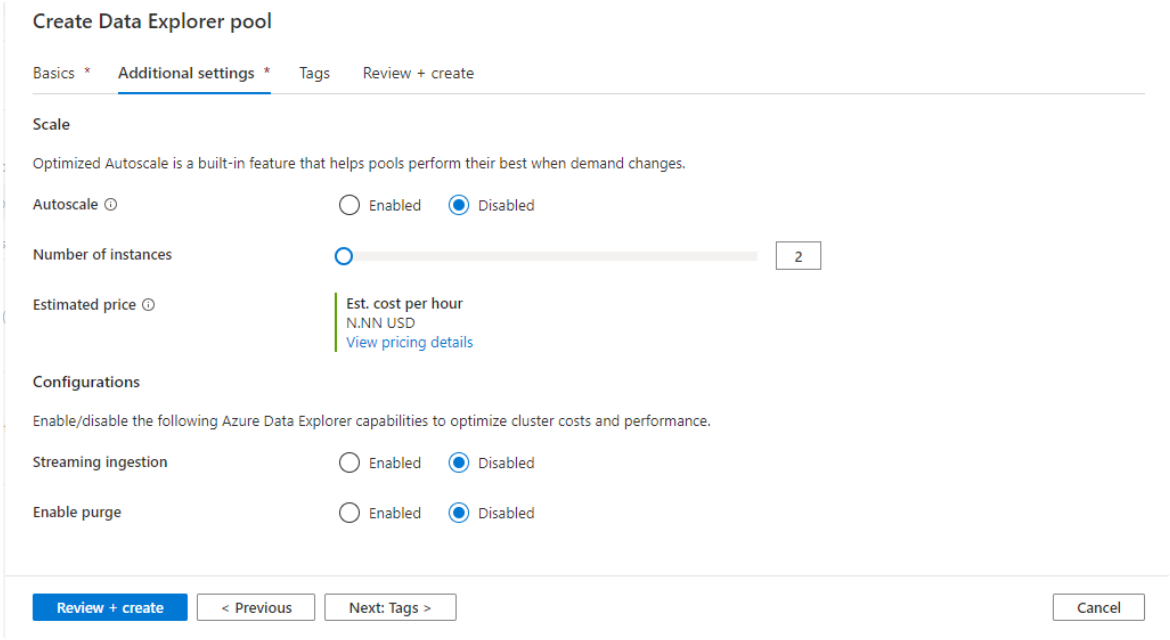


> **IMPORTANT**
> Note that there are specific limitations for the names that Data Explorer pools can use. Names must contain lowercase letters and numbers only, must be between 4 and 15 characters, and must start with a letter.

5. Select **Next: Additional settings**. Use the following settings and leave the defaults for the remaining settings.

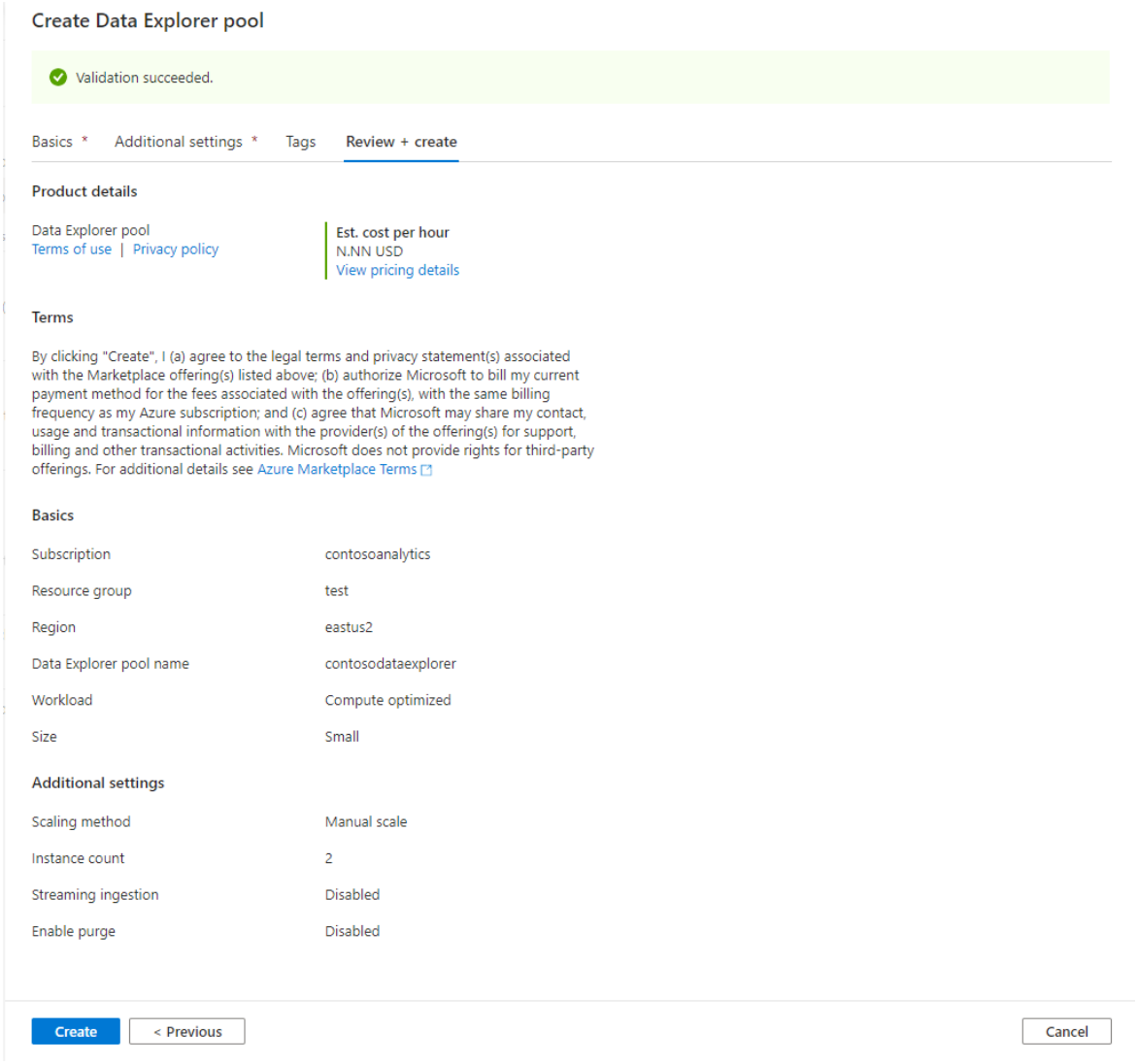
SETTING	SUGGESTED VALUE	DESCRIPTION
Autoscale	Disabled	We won't need autoscale in this quickstart

SETTING	SUGGESTED VALUE	DESCRIPTION
Number of instances	2	Set this to the smallest size to reduce costs for this quickstart

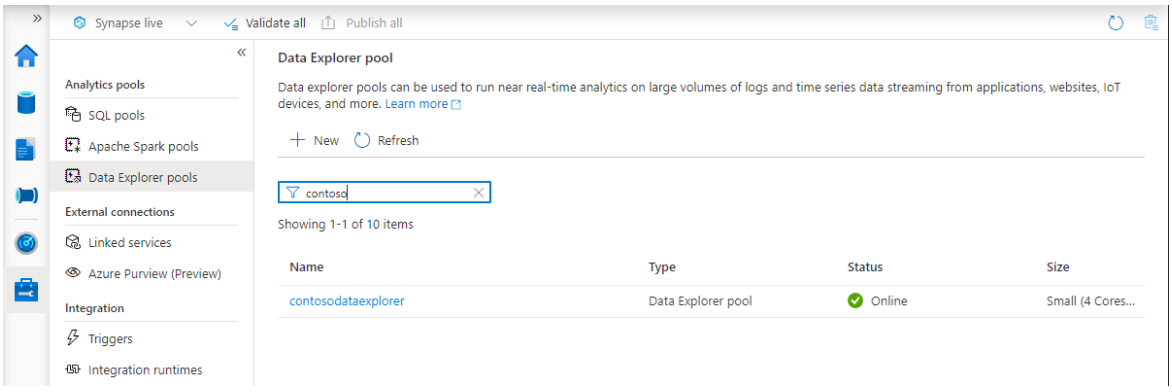


6. Select **Next: tags**. Don't add any tags.
7. Select **Review + create**.
8. Review the detail making sure they are correct, and then select **Create**.

   The Data Explorer pool will start the provisioning process.



9. Once the provisioning is complete, navigate back to the workspace and verify that the new Data Explorer pool appears in the list.



Clean up Data Explorer pool resources using Synapse Studio

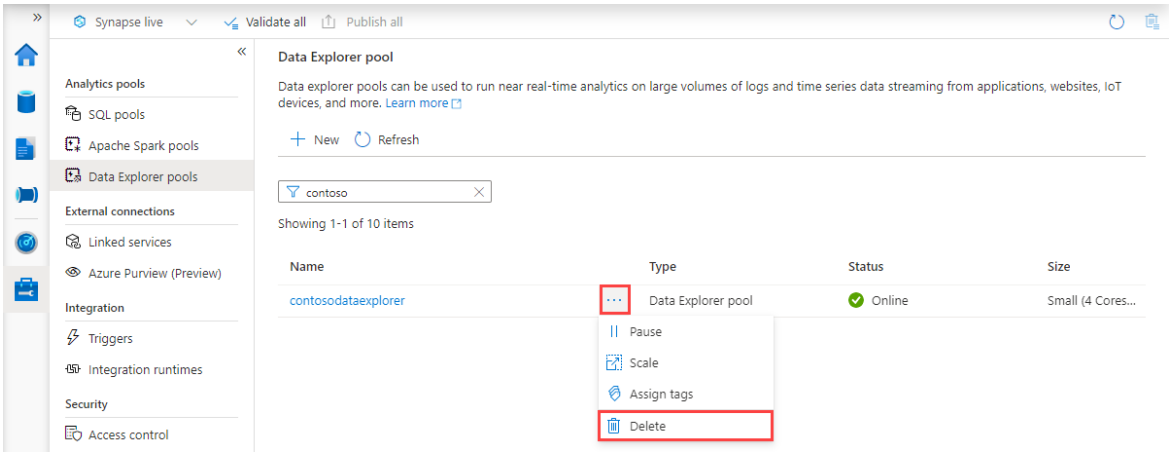
Use the following steps to delete the Data Explorer pool from the workspace using the Synapse Studio.

> **WARNING**
> Deleting a data-explorer pool will remove the analytics engine from the workspace. It will no longer be possible to connect to the pool, and all queries, pipelines, and notebooks that use the deleted pool will no longer work.

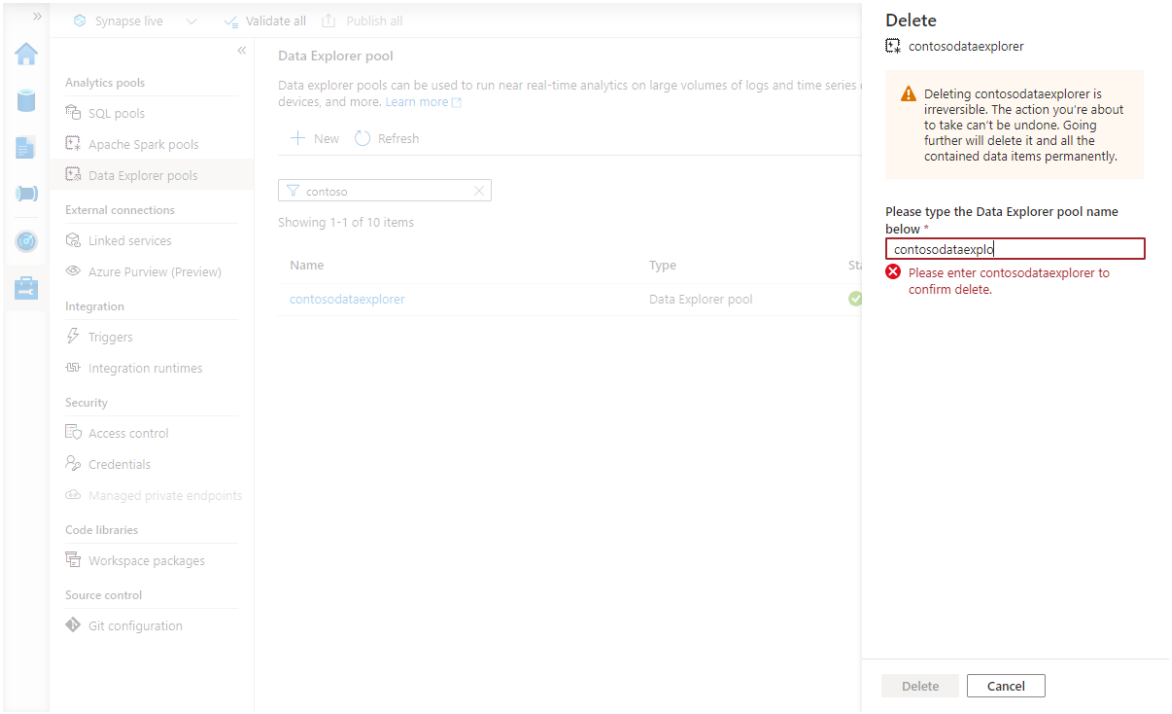
Delete the Data Explorer pool

1. Navigate to the Data Explorer pools in the workspace.

2. To remove the Data Explorer pool (in this case, **contosodataexplorer**), select **More [...]** > **Delete**.



3. To confirm the deletion, enter the name of the pool being deleted and then select **Delete**.



4. Once the process completes successfully, verify that the pool no longer appears in the list.

Next steps

- Ingestion overview
- Analyze with Data Explorer
- Monitor Data Explorer pools

Azure Synapse Data Explorer data ingestion overview (Preview)

8/19/2022 • 5 minutes to read • Edit Online

Data ingestion is the process used to load data records from one or more sources to import data into a table in Azure Synapse Data Explorer pool. Once ingested, the data becomes available for query.

The Azure Synapse Data Explorer data management service, which is responsible for data ingestion, implements the following process:

- Pulls data in batches or streaming from an external source and reads requests from a pending Azure queue.
- Batch data flowing to the same database and table is optimized for ingestion throughput.
- Initial data is validated and the format is converted where necessary.
- Further data manipulation including matching schema, organizing, indexing, encoding, and compressing the data.
- Data is persisted in storage according to the set retention policy.
- Ingested data is committed into the engine, where it's available for query.

Supported data formats, properties, and permissions

- **Supported data formats**
- **Ingestion properties**: The properties that affect how the data will be ingested (for example, tagging, mapping, creation time).
- **Permissions**: To ingest data, the process requires database ingestor level permissions. Other actions, such as query, may require database admin, database user, or table admin permissions.

Batching vs streaming ingestions

- Batching ingestion does data batching and is optimized for high ingestion throughput. This method is the preferred and most performant type of ingestion. Data is batched according to ingestion properties. Small batches of data are merged and optimized for fast query results. The ingestion batching policy can be set on databases or tables. By default, the maximum batching value is 5 minutes, 1000 items, or a total size of 1 GB. The data size limit for a batch ingestion command is 4 GB.
- Streaming ingestion is ongoing data ingestion from a streaming source. Streaming ingestion allows near real-time latency for small sets of data per table. Data is initially ingested to row store, then moved to column store extents.

Ingestion methods and tools

Azure Synapse Data Explorer supports several ingestion methods, each with its own target scenarios. These methods include ingestion tools, connectors and plugins to diverse services, managed pipelines, programmatic ingestion using SDKs, and direct access to ingestion.

Ingestion using managed pipelines

For organizations who wish to have management (throttling, retries, monitors, alerts, and more) done by an external service, using a connector is likely the most appropriate solution. Queued ingestion is appropriate for large data volumes. Azure Synapse Data Explorer supports the following Azure Pipelines:

- **Event Hub**: A pipeline that transfers events from services to Azure Synapse Data Explorer. For more information, see Ingest data from Event Hub into Azure Synapse Data Explorer.

- **Synapse pipelines**: A fully managed data integration service for analytic workloads in Synapse pipelines connects with over 90 supported sources to provide efficient and resilient data transfer. Synapse pipelines prepares, transforms, and enriches data to give insights that can be monitored in different kinds of ways. This service can be used as a one-time solution, on a periodic timeline, or triggered by specific events.

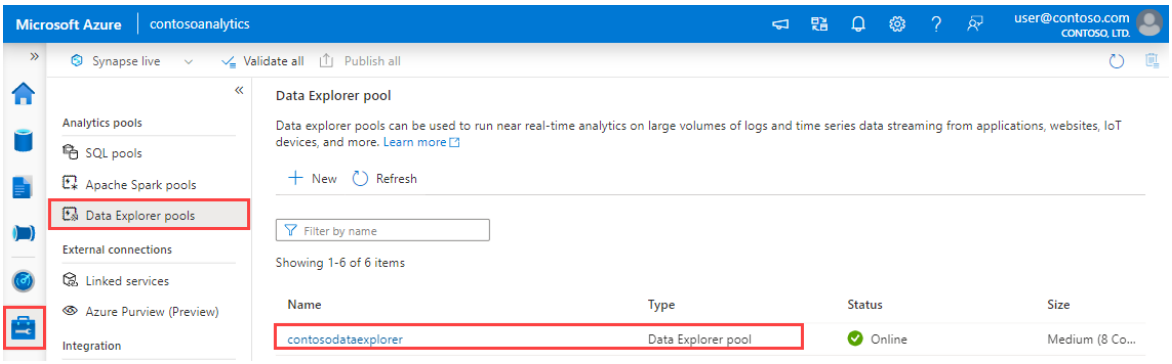
Programmatic ingestion using SDKs

Azure Synapse Data Explorer provides SDKs that can be used for query and data ingestion. Programmatic ingestion is optimized for reducing ingestion costs (COGs), by minimizing storage transactions during and following the ingestion process.

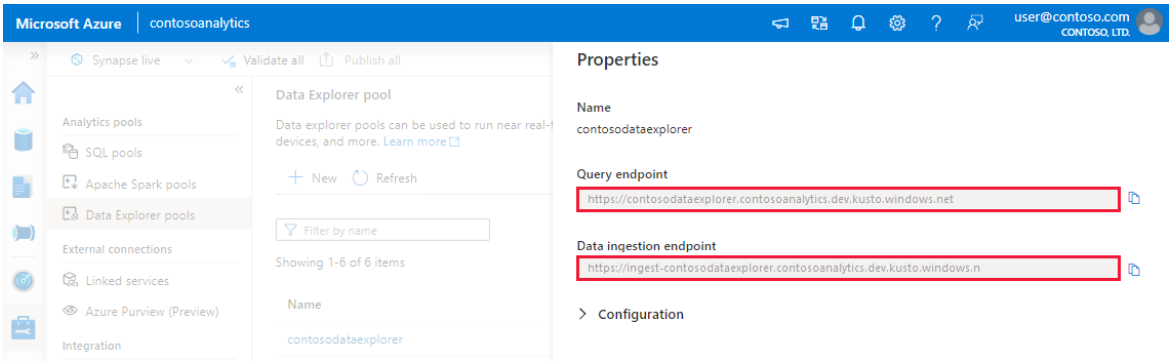
Before you start, use the following steps to get the Data Explorer pool endpoints for configuring programmatic ingestion.

1. In Synapse Studio, on the left-side pane, select **Manage** > **Data Explorer pools**.

2. Select the Data Explorer pool you want to use to view its details.



3. Make a note of the Query and Data Ingestion endpoints. Use the Query endpoint as the cluster when configuring connections to your Data Explorer pool. When configuring SDKs for data ingestion, use the data ingestion endpoint.



Available SDKs and open-source projects

- Python SDK
- .NET SDK
- Java SDK
- Node SDK
- REST API
- GO SDK

Tools

- **One-click ingestion**: Enables you to quickly ingest data by creating and adjusting tables from a wide range of source types. One-click ingestion automatically suggests tables and mapping structures based on the data source in Azure Synapse Data Explorer. One-click ingestion can be used for one-time ingestion, or to define continuous ingestion via Event Grid on the container to which the data was ingested.

Kusto Query Language ingest control commands

There are a number of methods by which data can be ingested directly to the engine by Kusto Query Language (KQL) commands. Because this method bypasses the Data Management services, it's only appropriate for exploration and prototyping. Don't use this method in production or high-volume scenarios.

- **Inline ingestion**: A control command .ingest inline is sent to the engine, with the data to be ingested being a part of the command text itself. This method is intended for improvised testing purposes.
- **Ingest from query**: A control command .set, .append, .set-or-append, or .set-or-replace is sent to the engine, with the data specified indirectly as the results of a query or a command.
- **Ingest from storage (pull)**: A control command .ingest into is sent to the engine, with the data stored in some external storage (for example, Azure Blob Storage) accessible by the engine and pointed-to by the command.

For an example of using ingest control commands, see Analyze with Data Explorer.

Ingestion process

Once you have chosen the most suitable ingestion method for your needs, do the following steps:

1. **Set retention policy**

   Data ingested into a table in Azure Synapse Data Explorer is subject to the table's effective retention policy. Unless set on a table explicitly, the effective retention policy is derived from the database's retention policy. Hot retention is a function of cluster size and your retention policy. Ingesting more data than you have available space will force the first in data to cold retention.

   Make sure that the database's retention policy is appropriate for your needs. If not, explicitly override it at the table level. For more information, see retention policy.

2. **Create a table**

   In order to ingest data, a table needs to be created beforehand. Use one of the following options:

   - Create a table with a command. For an example of using the create a table command, see Analyze with Data Explorer.
   - Create a table using One-click Ingestion.

   > **NOTE**
   > If a record is incomplete or a field cannot be parsed as the required data type, the corresponding table columns will be populated with null values.

3. **Create schema mapping**

   Schema mapping helps bind source data fields to destination table columns. Mapping allows you to take data from different sources into the same table, based on the defined attributes. Different types of mappings are supported, both row-oriented (CSV, JSON and AVRO), and column-oriented (Parquet). In most methods, mappings can also be pre-created on the table and referenced from the ingest command

parameter.

4. **Set update policy** (optional)

   Some of the data format mappings (Parquet, JSON, and Avro) support simple and useful ingest-time transformations. Where the scenario requires more complex processing at ingest time, use update policy, which allows for lightweight processing using Kusto Query Language commands. The update policy automatically runs extractions and transformations on ingested data on the original table, and ingests the resulting data into one or more destination tables. Set your update policy.

Next steps

- Supported data formats
- Supported ingestion properties

Quickstart: Ingest data using One-click (Preview)

8/19/2022 • 5 minutes to read • Edit Online

One-click ingestion makes the data ingestion process easy, fast, and intuitive. One-click ingestion helps you ramp-up quickly to start ingesting data, creating database tables, mapping structures. Select data from different kinds of sources in different data formats, either as a one-time or continuous ingestion process.

The following features make one-click ingestion so useful:

- Intuitive experience guided by the ingestion wizard
- Ingest data in a matter of minutes
- Ingest data from different kinds of sources: local file, blobs, and containers (up to 10,000 blobs)
- Ingest data in a variety of formats
- Ingest data into new or existing tables
- Table mapping and schema are suggested to you and easy to change

One-click ingestion is particularly useful when ingesting data for the first time, or when your data's schema is unfamiliar to you.

Prerequisites

- An Azure subscription. Create a free Azure account.
- Create a Data Explorer pool using Synapse Studio or the Azure portal
- Create a Data Explorer database.
  1. In Synapse Studio, on the left-side pane, select **Data**.
  2. Select **+** (Add new resource) > **Data Explorer pool**, and use the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Pool name	*contosodataexplorer*	The name of the Data Explorer pool to use
Name	*TestDatabase*	The database name must be unique within the cluster.
Default retention period	*365*	The time span (in days) for which it's guaranteed that the data is kept available to query. The time span is measured from the time that data is ingested.
Default cache period	*31*	The time span (in days) for which to keep frequently queried data available in SSD storage or RAM, rather than in longer-term storage.

  3. Select **Create** to create the database. Creation typically takes less than a minute.
- Create a table

1. In Synapse Studio, on the left-side pane, select **Develop**.
2. Under **KQL scripts**, Select **+** (Add new resource) > **KQL script**. On the right-side pane, you can name your script.
3. In the **Connect to** menu, select *contosodataexplorer*.
4. In the **Use database** menu, select *TestDatabase*.
5. Paste in the following command, and select **Run** to create the table.

```
.create table StormEvents (StartTime: datetime, EndTime: datetime, EpisodeId: int, EventId: int,
State: string, EventType: string, InjuriesDirect: int, InjuriesIndirect: int, DeathsDirect: int,
DeathsIndirect: int, DamageProperty: int, DamageCrops: int, Source: string, BeginLocation: string,
EndLocation: string, BeginLat: real, BeginLon: real, EndLat: real, EndLon: real, EpisodeNarrative:
string, EventNarrative: string, StormSummary: dynamic)
```

TIP

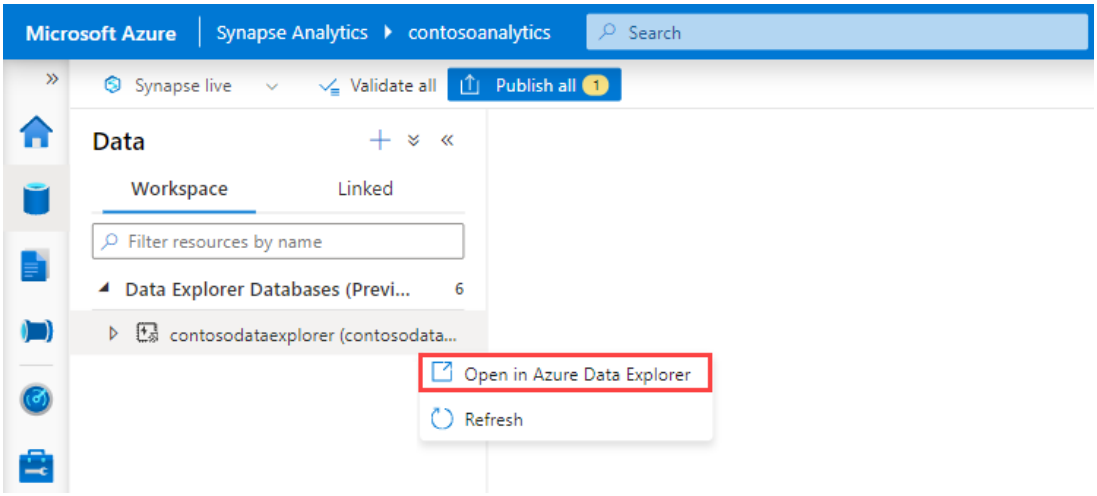
Verify that the table was successfully created. On the left-side pane, select **Data**, select the *contosodataexplorer* more menu, and then select **Refresh**. Under *contosodataexplorer*, expand **Tables** and make sure that the *StormEvents* table appears in the list.

Access the one-click wizard

The one-click ingestion wizard guides you through the one-click ingestion process.

- To access the wizard from Azure Synapse:

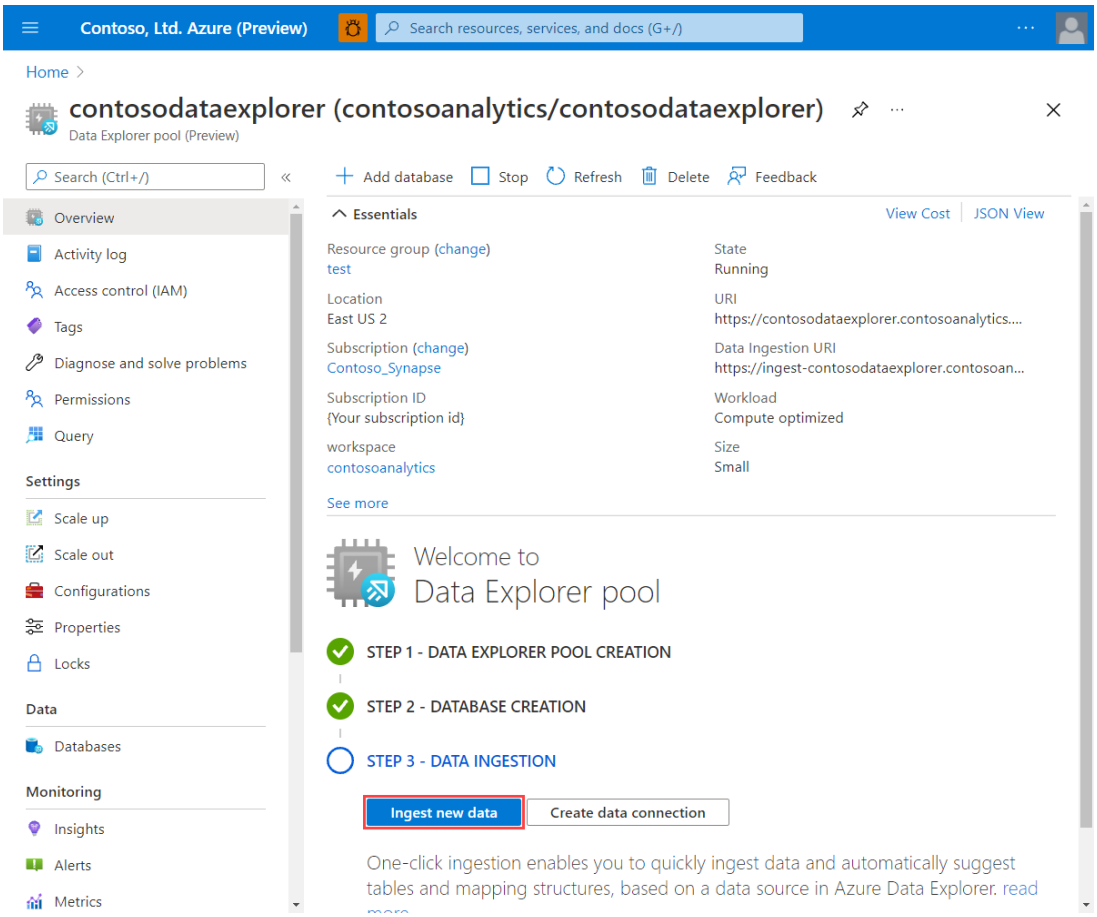
  1. In Synapse Studio, on the left-side pane, select **Data**.
  2. Under **Data Explorer Databases**, right-click the relevant database, and then select **Open in Azure Data Explorer**.



  3. Right-click the relevant pool, and then select **Ingest new data**.

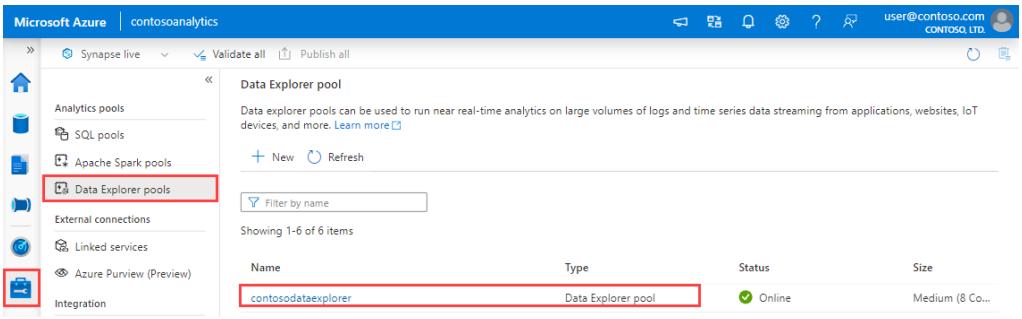
- To access the wizard from the Azure portal:

  1. In the Azure portal, search for and select the relevant Synapse workspace.
  2. Under **Data Explorer pools**, select the relevant pool.
  3. On the **Welcome to Data Explorer pool** home screen, select **Ingest new data**.

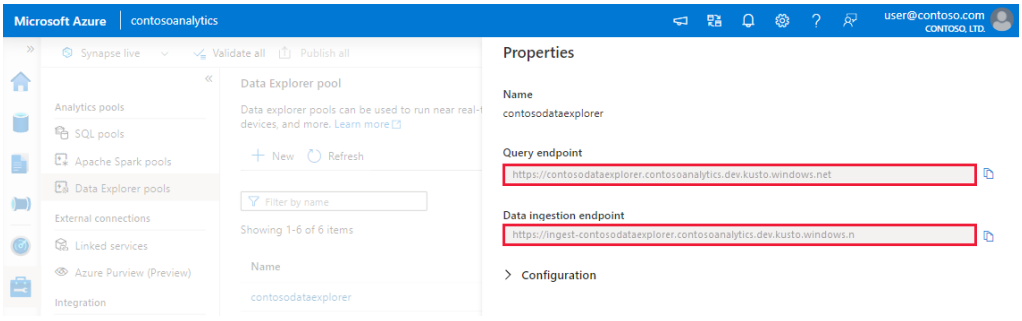


- To access the wizard from the Azure Data Explorer web ui:

1. Before you start, use the following steps to get the Query and Data Ingestion endpoints.
   a. In Synapse Studio, on the left-side pane, select **Manage** > **Data Explorer pools**.
   b. Select the Data Explorer pool you want to use to view its details.



   c. Make a note of the Query and Data Ingestion endpoints. Use the Query endpoint as the cluster when configuring connections to your Data Explorer pool. When configuring SDKs for data ingestion, use the data ingestion endpoint.

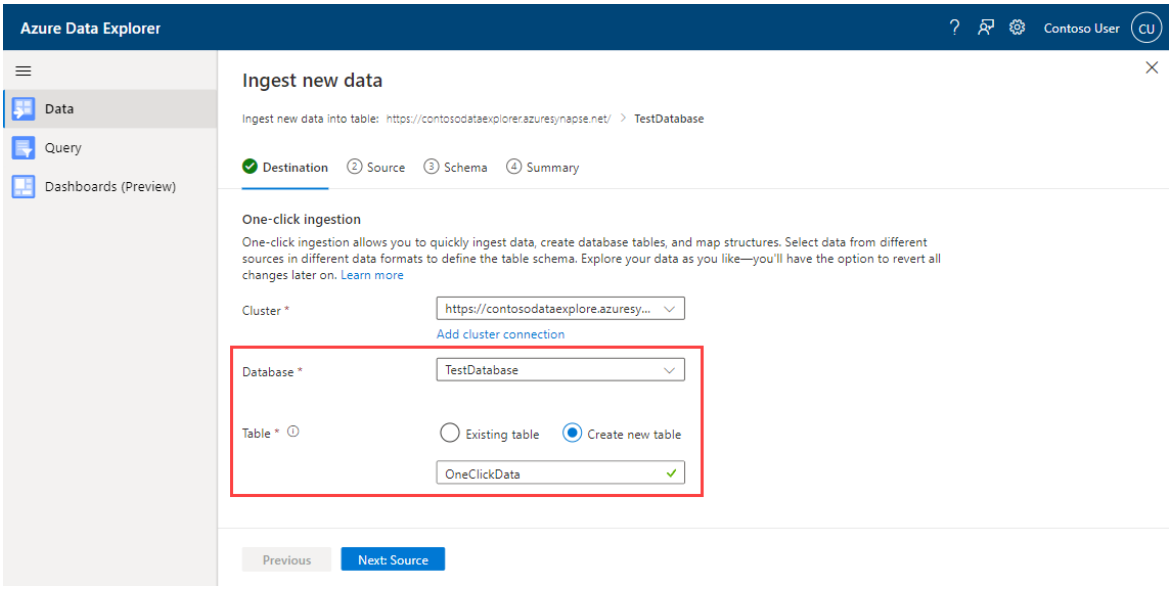


2. In the Azure Data Explorer web ui, add a connection to the *Query endpoint*.
3. Select **Query** from the left menu, right-click on the **database** or **table**, and select **Ingest new data**.

One-click ingestion wizard

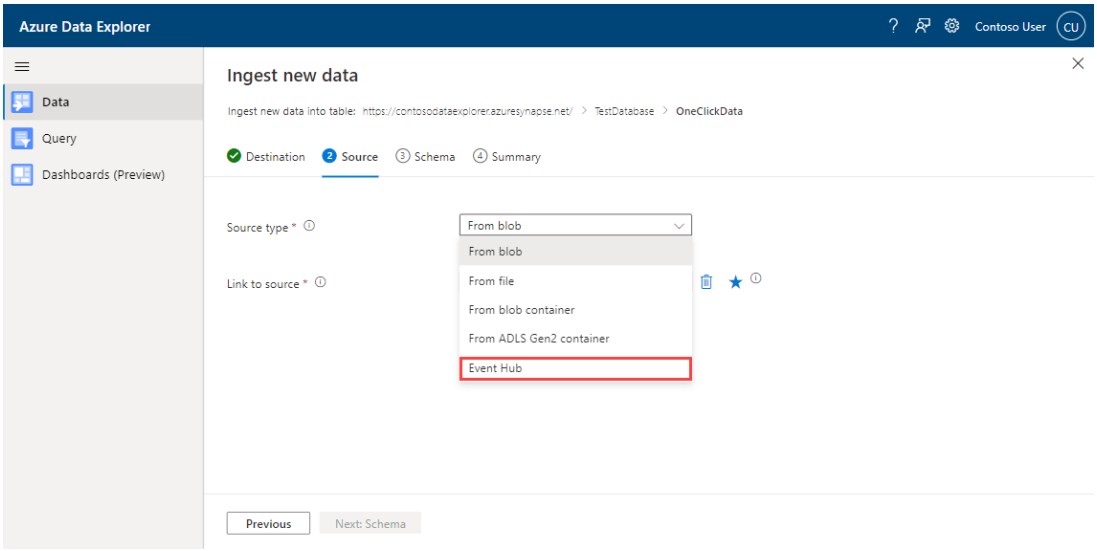
> **NOTE**
>
> This section describes the wizard using Event Hub as the data source. You can also use these steps to ingest data from a blob, file, blob container, and a ADLS Gen2 container.
>
> Replace the example values with actual values for your Synapse workspace.

1. On the **Destination** tab, choose the database and table for the ingested data.



2. On the **Source** tab:

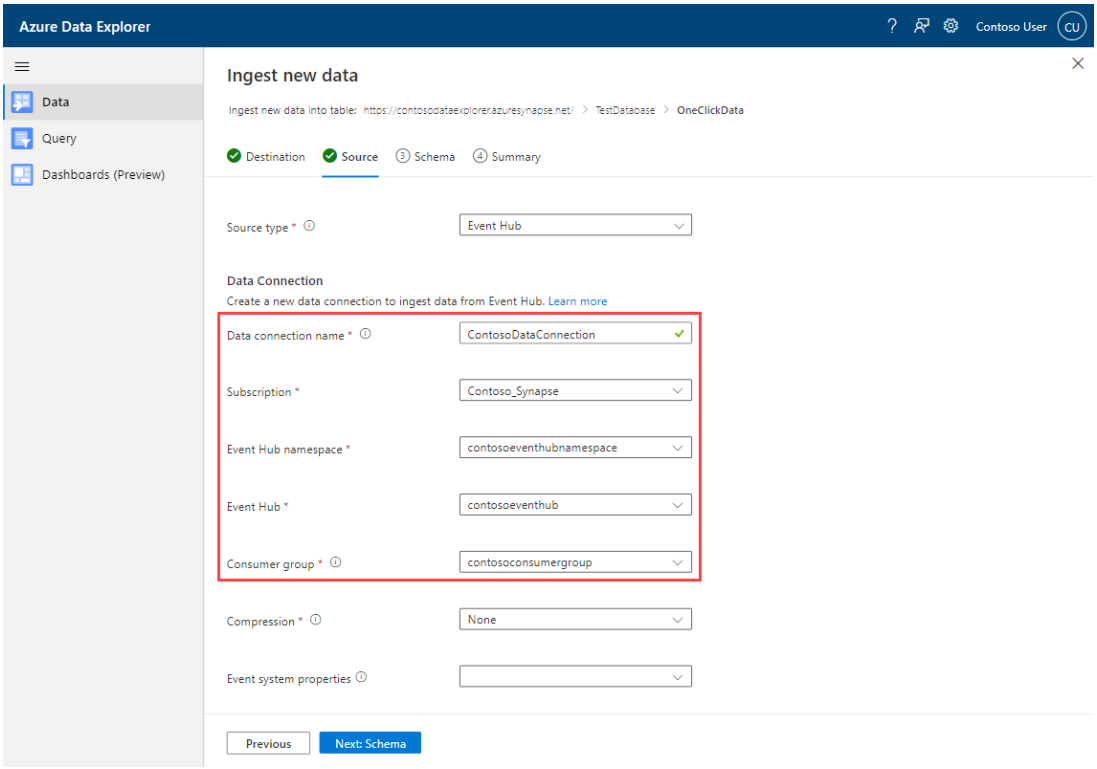
   a. Select *Event Hub* as the **Source type** for the ingestion.



   b. Fill out the Event Hub data connection details using the following information:

SETTING	EXAMPLE VALUE	DESCRIPTION
Data connection name	*ContosoDataConnection*	The name of the Event Hub data connection
Subscription	*Contoso_Synapse*	The subscription where the Event Hub resides.

SETTING	EXAMPLE VALUE	DESCRIPTION
Even Hub namespace	*contosoeventhubnamespace*	The namespace of the Event Hub.
Consumer group	*contosoconsumergroup*	The name of the Even Hub consumer group.



c. Select **Next**.

Schema mapping

The service automatically generates schema and ingestion properties, which you can change. You can use an existing mapping structure or create a new one, depending on if you're ingesting to a new or existing table.

In the **Schema** tab, do the following actions:

1. Confirm the autogenerated compression type.
2. Choose the format of your data. Different formats will allow you to make further changes.
3. Change mapping in the Editor window.

File formats

One-click ingestion supports ingesting from source data in all data formats supported by Data Explorer for ingestion.

Editor window

In the **Editor** window of the **Schema** tab, you can adjust data table columns as necessary.

The changes you can make in a table depend on the following parameters:

- **Table** type is new or existing
- **Mapping** type is new or existing

TABLE TYPE	MAPPING TYPE	AVAILABLE ADJUSTMENTS

TABLE TYPE	MAPPING TYPE	AVAILABLE ADJUSTMENTS
New table	New mapping	Change data type, Rename column, New column, Delete column, Update column, Sort ascending, Sort descending
Existing table	New mapping	New column (on which you can then change data type, rename, and update), Update column, Sort ascending, Sort descending
	Existing mapping	Sort ascending, Sort descending

> **NOTE**
> When adding a new column or updating a column, you can change mapping transformations. For more information, see Mapping transformations

Mapping transformations

Some data format mappings (Parquet, JSON, and Avro) support simple ingest-time transformations. To apply mapping transformations, create or update a column in the Editor window.

Mapping transformations can be performed on a column of **Type** string or datetime, with the **Source** having data type int or long. Supported mapping transformations are:

- DateTimeFromUnixSeconds
- DateTimeFromUnixMilliseconds
- DateTimeFromUnixMicroseconds
- DateTimeFromUnixNanoseconds

Data ingestion

Once you have completed schema mapping and column manipulations, the ingestion wizard will start the data ingestion process.

- When ingesting data from **non-container** sources, the ingestion will take immediate effect.
- If your data source is a **container**:
  - Data Explorer's batching policy will aggregate your data.
  - After ingestion, you can download the ingestion report and review the performance of each blob that was addressed.

Initial data exploration

After ingestion, the wizard gives you options to use **Quick commands** for initial exploration of your data.

Next steps

- Analyze with Data Explorer
- Monitor Data Explorer pools

Configure streaming ingestion on your Azure Synapse Data Explorer pool (Preview)

8/19/2022 • 11 minutes to read • Edit Online

Streaming ingestion is useful for loading data when you need low latency between ingestion and query. Consider using streaming ingestion in the following scenarios:

- Latency of less than a second is required.
- To optimize operational processing of many tables where the stream of data into each table is relatively small (a few records per second), but the overall data ingestion volume is high (thousands of records per second).

If the stream of data into each table is high (over 4 GB per hour), consider using batch ingestion.

To learn more about different ingestion methods, see data ingestion overview.

Choose the appropriate streaming ingestion type

Two streaming ingestion types are supported:

INGESTION TYPE	DESCRIPTION
Event Hub or **IoT Hub**	Hubs are configured as table streaming data sources. For information about setting these up, see **Event Hub**.
Custom ingestion	Custom ingestion requires you to write an application that uses one of the Azure Synapse Data Explorer client libraries. Use the information in this topic to configure custom ingestion. You may also find the C# streaming ingestion sample application helpful.

Use the following table to help you choose the ingestion type that's appropriate for your environment:

CRITERION	EVENT HUB / IOT HUB	CUSTOM INGESTION
Data delay between ingestion initiation and the data available for query	Longer delay	Shorter delay
Development overhead	Fast and easy setup, no development overhead	High development overhead to create an application ingest the data, handle errors, and ensure data consistency

> **NOTE**
> Ingesting data from an Event Hub into Data Explorer pools will not work if your Synapse workspace uses a managed virtual network with data exfiltration protection enabled.

Prerequisites

- An Azure subscription. Create a free Azure account.
- Create a Data Explorer pool using Synapse Studio or the Azure portal

- Create a Data Explorer database.

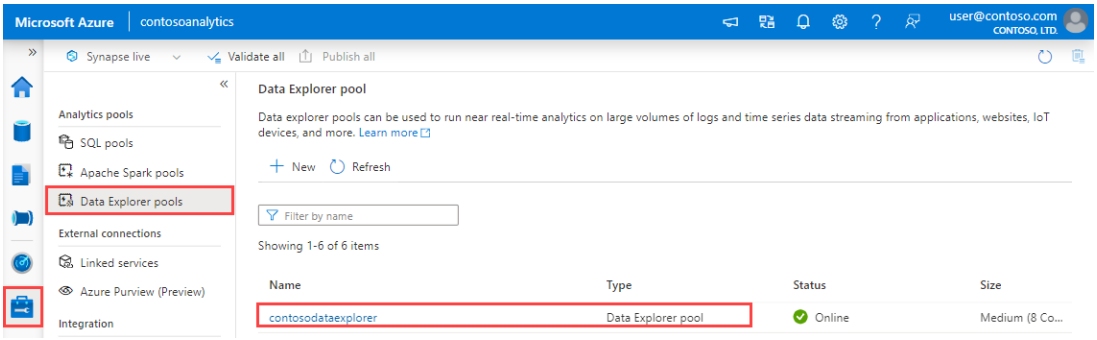
  1. In Synapse Studio, on the left-side pane, select **Data**.
  2. Select **+** (Add new resource) > **Data Explorer pool**, and use the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Pool name	*contosodataexplorer*	The name of the Data Explorer pool to use
Name	*TestDatabase*	The database name must be unique within the cluster.
Default retention period	*365*	The time span (in days) for which it's guaranteed that the data is kept available to query. The time span is measured from the time that data is ingested.
Default cache period	*31*	The time span (in days) for which to keep frequently queried data available in SSD storage or RAM, rather than in longer-term storage.

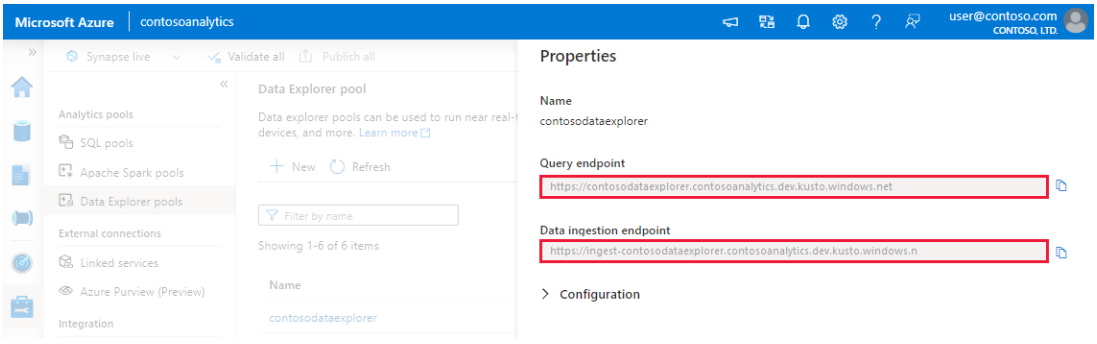
  3. Select **Create** to create the database. Creation typically takes less than a minute.

- Get the Query and Data Ingestion endpoints.

  1. In Synapse Studio, on the left-side pane, select **Manage** > **Data Explorer pools**.
  2. Select the Data Explorer pool you want to use to view its details.



  3. Make a note of the Query and Data Ingestion endpoints. Use the Query endpoint as the cluster when configuring connections to your Data Explorer pool. When configuring SDKs for data ingestion, use the data ingestion endpoint.



Performance and operational considerations

The main contributors that can impact streaming ingestion are:

- **Compute Specification**: Streaming ingestion performance and capacity scales with increased Data Explorer pool sizes. The number of concurrent ingestion requests is limited to six per core. For example, for 16 core workload type, such as Compute Optimized (Large) and Storage Optimized (Large), the maximal supported load is 96 concurrent ingestion requests. For two core workload type, such as Compute Optimized (Extra Small), the maximal supported load is 12 concurrent ingestion requests.
- **Data size limit**: The data size limit for a streaming ingestion request is 4 MB.
- **Schema updates**: Schema updates, such as creation and modification of tables and ingestion mappings, may take up to five minutes for the streaming ingestion service. For more information see Streaming ingestion and schema changes.
- **SSD capacity**: Enabling streaming ingestion on a Data Explorer pool, even when data isn't ingested via streaming, uses part of the local SSD disk of the Data Explorer pool machines for streaming ingestion data and reduces the storage available for hot cache.

Enable streaming ingestion on your Data Explorer pool

Before you can use streaming ingestion, you must enable the capability on your Data Explorer pool and define a streaming ingestion policy. You can enable the capability when creating the Data Explorer pool, or add it to an existing Data Explorer pool.

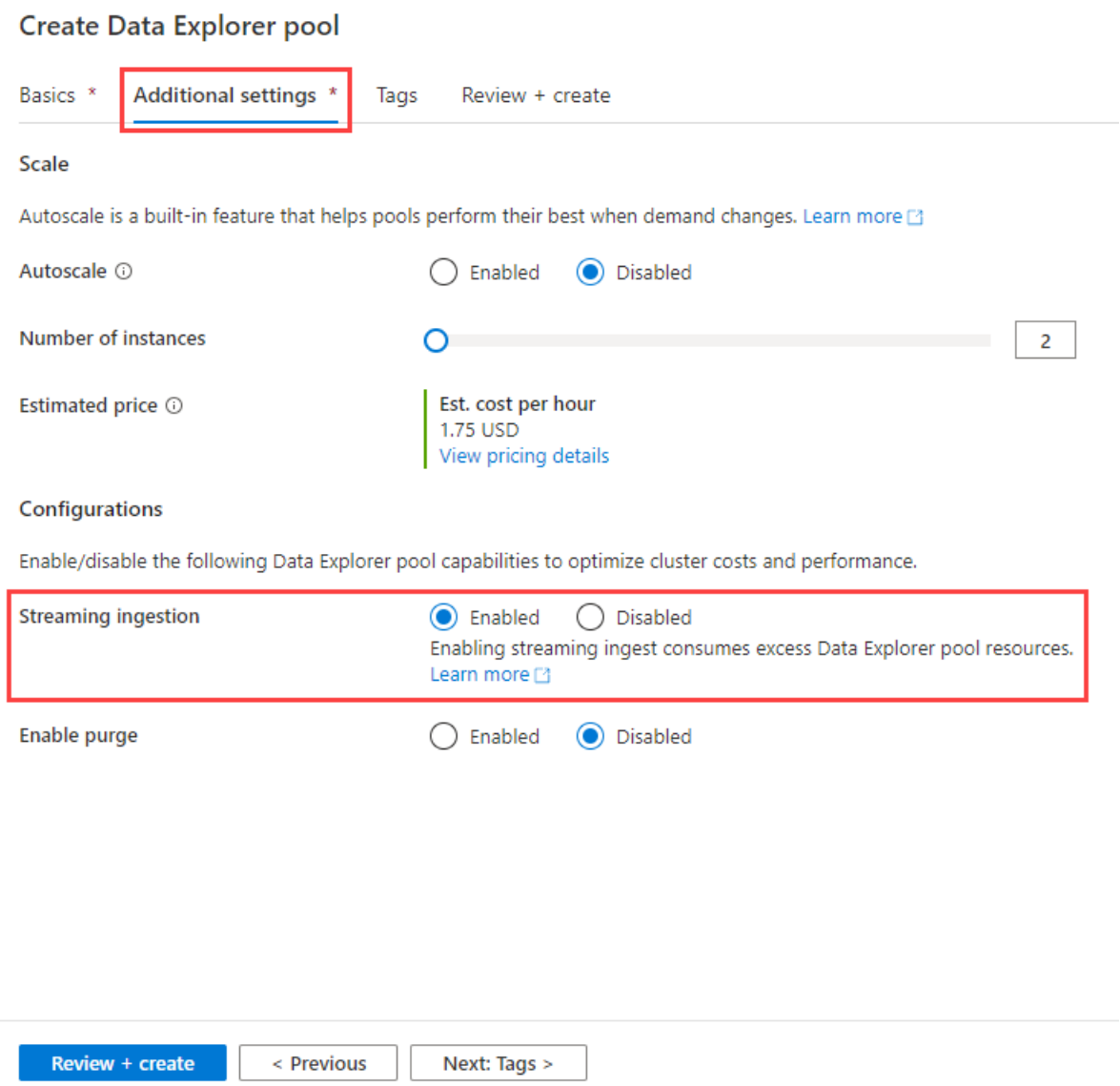
> **WARNING**
> Review the limitations prior to enabling streaming ingestion.

Enable streaming ingestion while creating a new Data Explorer pool

You can enable streaming ingestion while creating a new Data Explorer pool using Azure Synapse Studio or the Azure portal.

- Studio
- Portal

While creating a Data Explorer pool using the steps in Create a Data Explorer pool using Synapse Studio, in the **Additional settings** tab, select **Streaming ingestion** > **Enabled**.



Enable streaming ingestion on an existing Data Explorer pool

If you have an existing Data Explorer pool, you can enable streaming ingestion using the Azure portal.

1. In the Azure portal, go to your Data Explorer pool.
2. In **Settings**, select **Configurations**.
3. In the **Configurations** pane, select **On** to enable **Streaming ingestion**.
4. Select **Save**.

Create a target table and define the policy

Create a table to receive the streaming ingestion data and define its related policy using Azure Synapse Studio or the Azure portal.

- Studio
- Portal

1. In Synapse Studio, on the left-side pane, select **Develop**.
2. Under **KQL scripts**, Select **+** (Add new resource) > **KQL script**. On the right-side pane, you can name your script.
3. In the **Connect to** menu, select *contosodataexplorer*.
4. In the **Use database** menu, select *TestDatabase*.
5. Paste in the following command, and select **Run** to create the table.

```
.create table TestTable (TimeStamp: datetime, Name: string, Metric: int, Source:string)
```

6. Copy one of the following commands into the **Query pane** and select **Run**. This defines the streaming ingestion policy on the table you created or on the database that contains the table.

> **TIP**
> A policy that is defined at the database level applies to all existing and future tables in the database.

- To define the policy on the table you created, use:

```
.alter table TestTable policy streamingingestion enable
```

- To define the policy on the database containing the table you created, use:

```
.alter database StreamingTestDb policy streamingingestion enable
```

Create a streaming ingestion application to ingest data to your Data Explorer pool

Create your application for ingesting data to your Data Explorer pool using your preferred language. For the *poolPath* variable, use the Query endpoint you made a note of in the Prerequisites.

- C#
- Python
- Node.js
- Go
- Java

```
using Kusto.Data;
using Kusto.Ingest;
using System.IO;
using Kusto.Data.Common;

namespace StreamingIngestion
{
    class Program
    {
        static void Main(string[] args)
        {
            string poolPath = "https://<Poolname>.<WorkspaceName>.kusto.windows.net";
            string appId = "<appId>";
            string appKey = "<appKey>";
            string appTenant = "<appTenant>";
            string dbName = "<dbName>";
            string tableName = "<tableName>";

            // Create Kusto connection string with App Authentication
            var csb =
                new KustoConnectionStringBuilder(poolPath)
                    .WithAadApplicationKeyAuthentication(
                        applicationClientId: appId,
                        applicationKey: appKey,
                        authority: appTenant
                    );

            // Create a disposable client that will execute the ingestion
            using (IKustoIngestClient client = KustoIngestFactory.CreateStreamingIngestClient(csb))
            {
                // Initialize client properties
                var ingestionProperties =
                    new KustoIngestionProperties(
                        databaseName: dbName,
                        tableName: tableName
                    );

                // Ingest from a compressed file
                var fileStream = File.Open("MyFile.gz", FileMode.Open);
                // Create source options
                var sourceOptions = new StreamSourceOptions()
                {
                    CompressionType = DataSourceCompressionType.GZip,
                };
                // Ingest from stream
                var status = client.IngestFromStreamAsync(fileStream, ingestionProperties,
sourceOptions).GetAwaiter().GetResult();
            }
        }
    }
}
```

Disable streaming ingestion on your Data Explorer pool

> **WARNING**
> Disabling streaming ingestion may take a few hours.

Before disabling streaming ingestion on your Data Explorer pool, drop the streaming ingestion policy from all relevant tables and databases. The removal of the streaming ingestion policy triggers data rearrangement inside your Data Explorer pool. The streaming ingestion data is moved from the initial storage to permanent storage in the column store (extents or shards). This process can take between a few seconds to a few hours, depending on

the amount of data in the initial storage.

Drop the streaming ingestion policy

You can drop the streaming ingestion policy using Azure Synapse Studio or the Azure portal.

- Studio
- Portal

1. In Synapse Studio, on the left-side pane, select **Develop**.
2. Under **KQL scripts**, Select **+** (Add new resource) > **KQL script**. On the right-side pane, you can name your script.
3. In the **Connect to** menu, select *contosodataexplorer*.
4. In the **Use database** menu, select *TestDatabase*.
5. Paste in the following command, and select **Run** to create the table.

```
.delete table TestTable policy streamingingestion
```

6. In the Azure portal, go to your Data Explorer pool.
7. In **Settings**, select **Configurations**.
8. In the **Configurations** pane, select **On** to enable **Streaming ingestion**.
9. Select **Save**.

Limitations

- Database cursors aren't supported for a database if the database itself or any of its tables have the Streaming ingestion policy defined and enabled.
- Data mappings must be pre-created for use in streaming ingestion. Individual streaming ingestion requests don't accommodate inline data mappings.
- Extent tags can't be set on the streaming ingestion data.
- Update policy. The update policy can reference only the newly-ingested data in the source table and not any other data or tables in the database.
- If streaming ingestion is used on any of the tables of the database, this database cannot be used as leader for follower databases or as a data provider for Azure Synapse Analytics Data Share.

Next steps

- Analyze with Data Explorer
- Monitor Data Explorer pools

Event Hub data connection (Preview)

8/19/2022 • 6 minutes to read • Edit Online

Azure Event Hubs is a big data streaming platform and event ingestion service. Azure Synapse Data Explorer offers continuous ingestion from customer-managed Event Hubs.

The Event Hub ingestion pipeline transfers events to Azure Synapse Data Explorer in several steps. You first create an Event Hub in the Azure portal. You then create a target table in Azure Synapse Data Explorer into which the data in a particular format, will be ingested using the given ingestion properties. The Event Hub connection needs to know events routing. Data is embedded with selected properties according to the event system properties mapping. Create a connection to Event Hub to create an Event Hub and send events. This process can be managed through the Azure portal, programmatically with C# or Python, or with the Azure Resource Manager template.

For general information about data ingestion in Azure Synapse Data Explorer, see Azure Synapse Data Explorer data ingestion overview.

Data format

- Data is read from the Event Hub in form of EventData objects.
- See supported formats.

  > **NOTE**
  > Event Hub doesn't support the .raw format.

- Data can be compressed using the `GZip` compression algorithm. Specify `Compression` in ingestion properties.
  - Data compression isn't supported for compressed formats (Avro, Parquet, ORC).
  - Custom encoding and embedded system properties aren't supported on compressed data.

Ingestion properties

Ingestion properties instruct the ingestion process, where to route the data, and how to process it. You can specify ingestion properties of the events ingestion using the EventData.Properties. You can set the following properties:

PROPERTY	DESCRIPTION
Table	Name (case sensitive) of the existing target table. Overrides the `Table` set on the `Data Connection` pane.
Format	Data format. Overrides the `Data format` set on the `Data Connection` pane.
IngestionMappingReference	Name of the existing ingestion mapping to be used. Overrides the `Column mapping` set on the `Data Connection` pane.

PROPERTY	DESCRIPTION
Compression	Data compression, `None` (default), or `GZip` compression.
Encoding	Data encoding, the default is UTF8. Can be any of .NET supported encodings.
Tags	A list of tags to associate with the ingested data, formatted as a JSON array string. There are performance implications when using tags.

> **NOTE**
> Only events enqueued after you create the data connection are ingested.

Events routing

When you set up an Event Hub connection to Azure Synapse Data Explorer cluster, you specify target table properties (table name, data format, compression, and mapping). The default routing for your data is also referred to as `static routing`. You can also specify target table properties for each event, using event properties. The connection will dynamically route the data as specified in the EventData.Properties, overriding the static properties for this event.

In the following example, set Event Hub details and send weather metric data to table `WeatherMetrics`. Data is in `json` format. `mapping1` is pre-defined on the table `WeatherMetrics`.

```
var eventHubNamespaceConnectionString=<connection_string>;
var eventHubName=<event_hub>;

// Create the data
var metric = new Metric { Timestamp = DateTime.UtcNow, MetricName = "Temperature", Value = 32 };
var data = JsonConvert.SerializeObject(metric);

// Create the event and add optional "dynamic routing" properties
var eventData = new EventData(Encoding.UTF8.GetBytes(data));
eventData.Properties.Add("Table", "WeatherMetrics");
eventData.Properties.Add("Format", "json");
eventData.Properties.Add("IngestionMappingReference", "mapping1");
eventData.Properties.Add("Tags", "['mydatatag']");

// Send events
var eventHubClient = EventHubClient.CreateFromConnectionString(eventHubNamespaceConnectionString,
eventHubName);
eventHubClient.Send(eventData);
eventHubClient.Close();
```

Event system properties mapping

System properties store properties that are set by the Event Hubs service, at the time the event is enqueued. The Azure Synapse Data Explorer Event Hub connection will embed the selected properties into the data landing in your table.

NOTE

- System properties are supported for `json` and tabular formats (`csv`, `tsv` etc.) and aren't supported on compressed data. When using a non-supported format, the data will still be ingested, but the properties will be ignored.
- For tabular data, system properties are supported only for single-record event messages.
- For JSON data, system properties are also supported for multiple-record event messages. In such cases, the system properties are added only to the first record of the event message.
- For `csv` mapping, properties are added at the beginning of the record in the order listed in the System properties table.
- For `json` mapping, properties are added according to property names in the System properties table.

System properties

Event Hub exposes the following system properties:

PROPERTY	DATA TYPE	DESCRIPTION
x-opt-enqueued-time	datetime	UTC time when the event was enqueued
x-opt-sequence-number	long	The logical sequence number of the event within the partition stream of the Event Hub
x-opt-offset	string	The offset of the event from the Event Hub partition stream. The offset identifier is unique within a partition of the Event Hub stream
x-opt-publisher	string	The publisher name, if the message was sent to a publisher endpoint
x-opt-partition-key	string	The partition key of the corresponding partition that stored the event

If you selected **Event system properties** in the **Data Source** section of the table, you must include the properties in the table schema and mapping.

Schema mapping examples

Table schema mapping example

If your data includes three columns (`Timespan`, `Metric`, and `Value`) and the properties you include are `x-opt-enqueued-time` and `x-opt-offset`, create or alter the table schema by using this command:

```
    .create-merge table TestTable (TimeStamp: datetime, Metric: string, Value: int,
EventHubEnqueuedTime:datetime, EventHubOffset:string)
```

CSV mapping example

Run the following commands to add data to the beginning of the record. Note ordinal values.

```
.create table TestTable ingestion csv mapping "CsvMapping1"
'['
'   { "column" : "Timespan", "Properties":{"Ordinal":"2"}},'
'   { "column" : "Metric", "Properties":{"Ordinal":"3"}},'
'   { "column" : "Value", "Properties":{"Ordinal":"4"}},'
'   { "column" : "EventHubEnqueuedTime", "Properties":{"Ordinal":"0"}},'
'   { "column" : "EventHubOffset", "Properties":{"Ordinal":"1"}}'
']'
```

JSON mapping example

Data is added by using the system properties mapping. Run these commands:

```
.create table TestTable ingestion json mapping "JsonMapping1"
'['
'    { "column" : "Timespan", "Properties":{"Path":"$.timestamp"}},'
'    { "column" : "Metric", "Properties":{"Path":"$.metric"}},'
'    { "column" : "Value", "Properties":{"Path":"$.value"}},'
'    { "column" : "EventHubEnqueuedTime", "Properties":{"Path":"$.x-opt-enqueued-time"}},'
'    { "column" : "EventHubOffset", "Properties":{"Path":"$.x-opt-offset"}}'
']'
```

Event Hub connection

> **NOTE**
> For best performance, create all resources in the same region as the Azure Synapse Data Explorer cluster.

Create an Event Hub

If you don't already have one, Create an Event Hub. Connecting to Event Hub can be managed through the Azure portal, programmatically with C# or Python, or with the Azure Resource Manager template.

> **NOTE**
> - The partition count isn't changeable, so you should consider long-term scale when setting partition count.
> - Consumer group *must* be unique per consumer. Create a consumer group dedicated to Azure Synapse Data Explorer connection.

Send events

See the sample app that generates data and sends it to an Event Hub.

For an example of how to generate sample data, see Ingest data from Event Hub into Azure Synapse Data Explorer

Set up Geo-disaster recovery solution

Event Hub offers a Geo-disaster recovery solution. Azure Synapse Data Explorer doesn't support `Alias` Event Hub namespaces. To implement the Geo-disaster recovery in your solution, create two Event Hub data connections: one for the primary namespace and one for the secondary namespace. Azure Synapse Data Explorer will listen to both Event Hub connections.

> **NOTE**
>
> It's the user's responsibility to implement a failover from the primary namespace to the secondary namespace.

Next steps

- Ingest data from Event Hub into Azure Synapse Data Explorer
- Create an Event Hub data connection for Azure Synapse Data Explorer using C#
- Create an Event Hub data connection for Azure Synapse Data Explorer using Python
- Create an Event Hub data connection for Azure Synapse Data Explorer using Azure Resource Manager template

Ingest data from Event Hub into Azure Synapse Data Explorer

8/19/2022 • 9 minutes to read • Edit Online

Azure Synapse Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Synapse Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers.

Azure Synapse Data Explorer offers ingestion (data loading) from Event Hubs, a big data streaming platform and event ingestion service. Event Hubs can process millions of events per second in near real time. In this article, you create an Event Hub, connect to it from Azure Synapse Data Explorer and see data flow through the system.

Prerequisites

- An Azure subscription. Create a free Azure account.
- Create a Data Explorer pool using Synapse Studio or the Azure portal
- Create a Data Explorer database.
  1. In Synapse Studio, on the left-side pane, select **Data**.
  2. Select **+** (Add new resource) > **Data Explorer pool**, and use the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Pool name	*contosodataexplorer*	The name of the Data Explorer pool to use
Name	*TestDatabase*	The database name must be unique within the cluster.
Default retention period	*365*	The time span (in days) for which it's guaranteed that the data is kept available to query. The time span is measured from the time that data is ingested.
Default cache period	*31*	The time span (in days) for which to keep frequently queried data available in SSD storage or RAM, rather than in longer-term storage.

  3. Select **Create** to create the database. Creation typically takes less than a minute.
- Create a target table to which Event Hubs will send data
  1. In Synapse Studio, on the left-side pane, select **Develop**.
  2. Under **KQL scripts**, Select **+** (Add new resource) > **KQL script**. On the right-side pane, you can name your script.
  3. In the **Connect to** menu, select *contosodataexplorer*.

4. In the **Use database** menu, select *TestDatabase*.

5. Paste in the following command, and select **Run** to create the table.

```
.create table TestTable (TimeStamp: datetime, Name: string, Metric: int, Source:string)
```

> **TIP**
> Verify that the table was successfully created. On the left-side pane, select **Data**, select the *contosodataexplorer* more menu, and then select **Refresh**. Under *contosodataexplorer*, expand **Tables** and make sure that the *TestTable* table appears in the list.

6. Copy the following command into the window and select **Run** to map the incoming JSON data to the column names and data types of the table (TestTable).

```
.create table TestTable ingestion json mapping 'TestMapping' '[{"column":"TimeStamp",
"Properties": {"Path": "$.timeStamp"}},{"column":"Name", "Properties": {"Path":"$.name"}} ,
{"column":"Metric", "Properties": {"Path":"$.metric"}}, {"column":"Source", "Properties":
{"Path":"$.source"}}]'
```

- We recommend using a user assigned managed identity or system assigned managed identity for the data connection (optional).
- A sample app that generates data and sends it to an Event Hub. Download the sample app to your system.
- Visual Studio 2019 to run the sample app.

Sign in to the Azure portal

Sign in to the Azure portal.

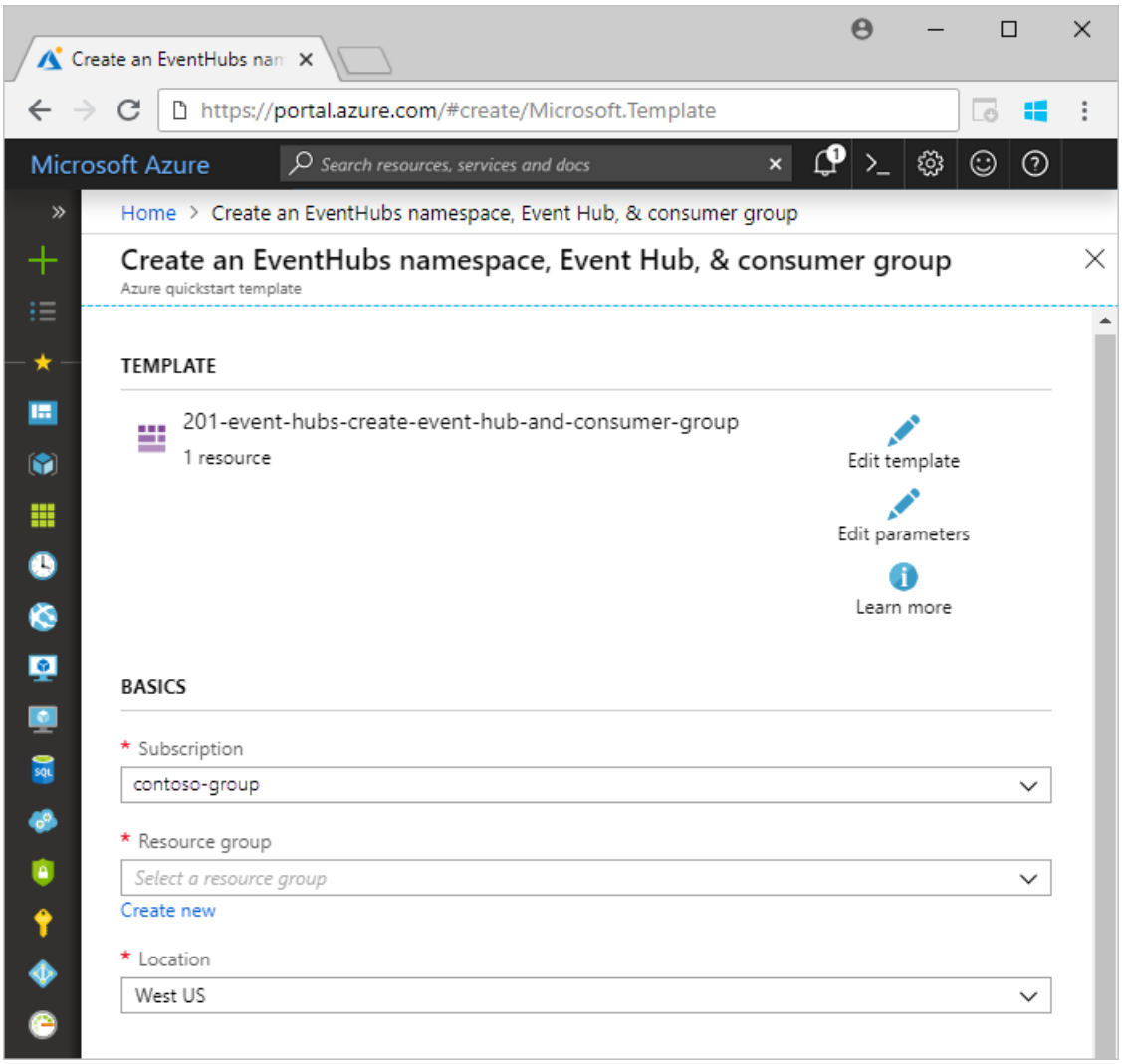
Create an Event Hub

Create an Event Hub by using an Azure Resource Manager template in the Azure portal.

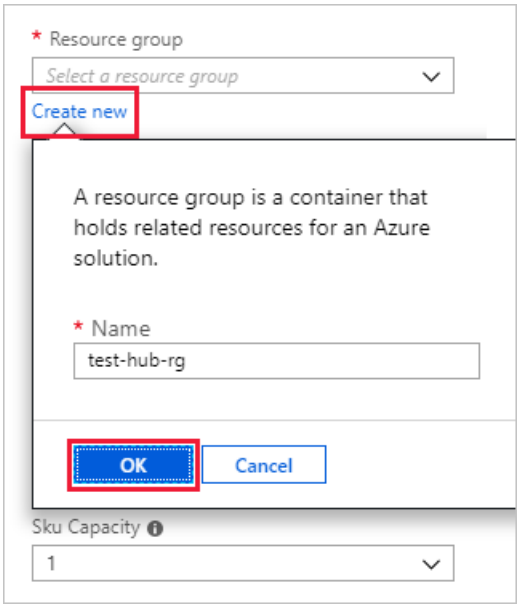
1. To create an Event Hub, use the following button to start the deployment. Right-click and select **Open in new window**, so you can follow the rest of the steps in this article.

   Deploy to Azure

   The **Deploy to Azure** button takes you to the Azure portal.



2. Select the subscription where you want to create the Event Hub, and create a resource group named *test-hub-rg*.

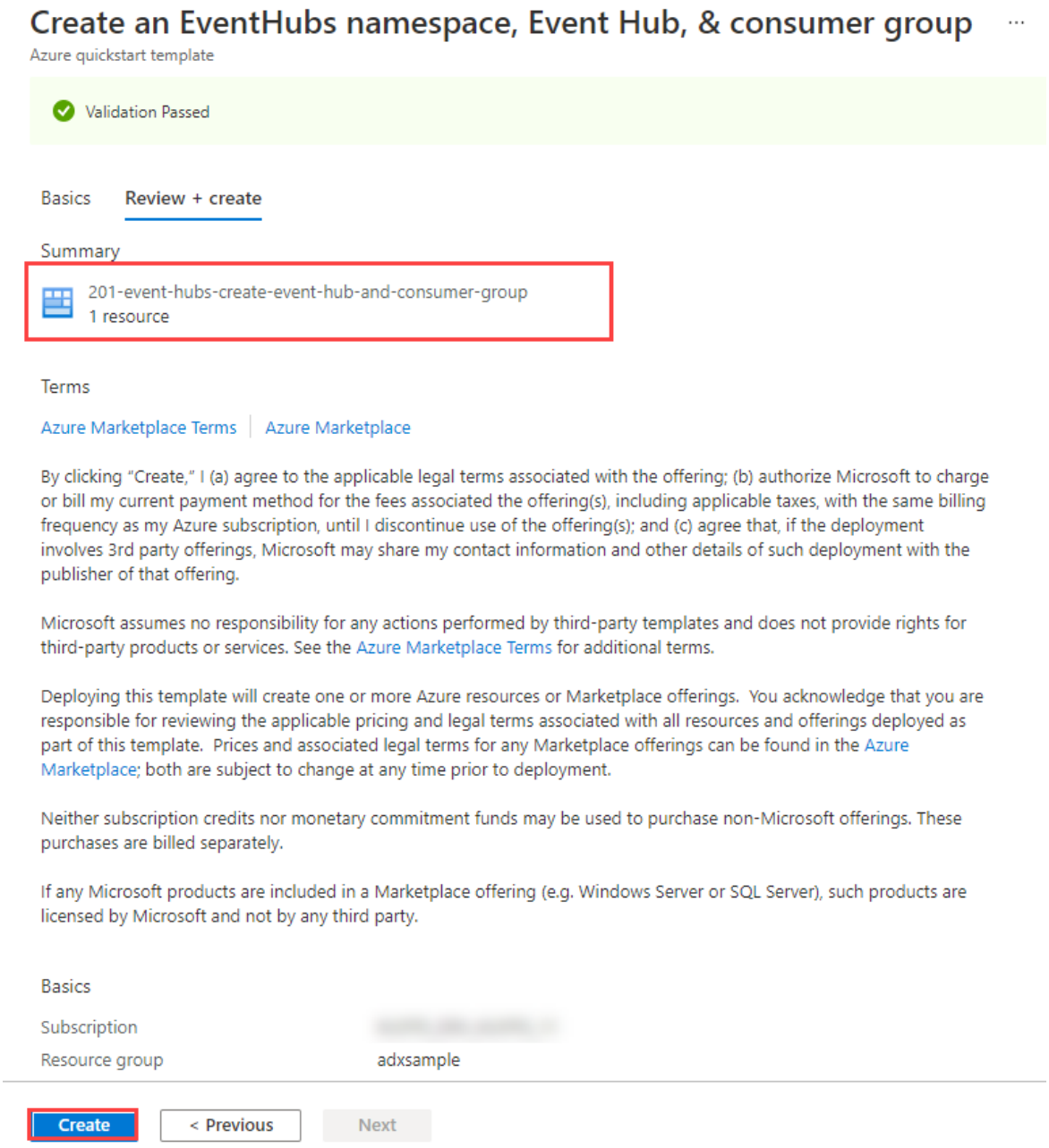


3. Fill out the form with the following information.

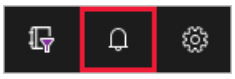
   Use defaults for any settings not listed in the following table.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Subscription	Your subscription	Select the Azure subscription that you want to use for your Event Hub.
Resource group	*test-hub-rg*	Create a new resource group.
Location	*West US*	Select *West US* for this article. For a production system, select the region that best meets your needs. Create the Event Hub namespace in the same Location as the Azure Synapse Data Explorer cluster for best performance (most important for Event Hub namespaces with high throughput).
Namespace name	A unique namespace name	Choose a unique name that identifies your namespace. For example, *mytestnamespace*. The domain name *servicebus.windows.net* is appended to the name you provide. The name can contain only letters, numbers, and hyphens. The name must start with a letter, and it must end with a letter or number. The value must be between 6 and 50 characters long.
Event Hub name	*test-hub*	The Event Hub sits under the namespace, which provides a unique scoping container. The Event Hub name must be unique within the namespace.
Consumer group name	*test-group*	Consumer groups enable multiple consuming applications to each have a separate view of the event stream.

4. Select **Review + create**.
5. Review the **Summary** of resources created. Select **Create**, which acknowledges that you're creating resources in your subscription.



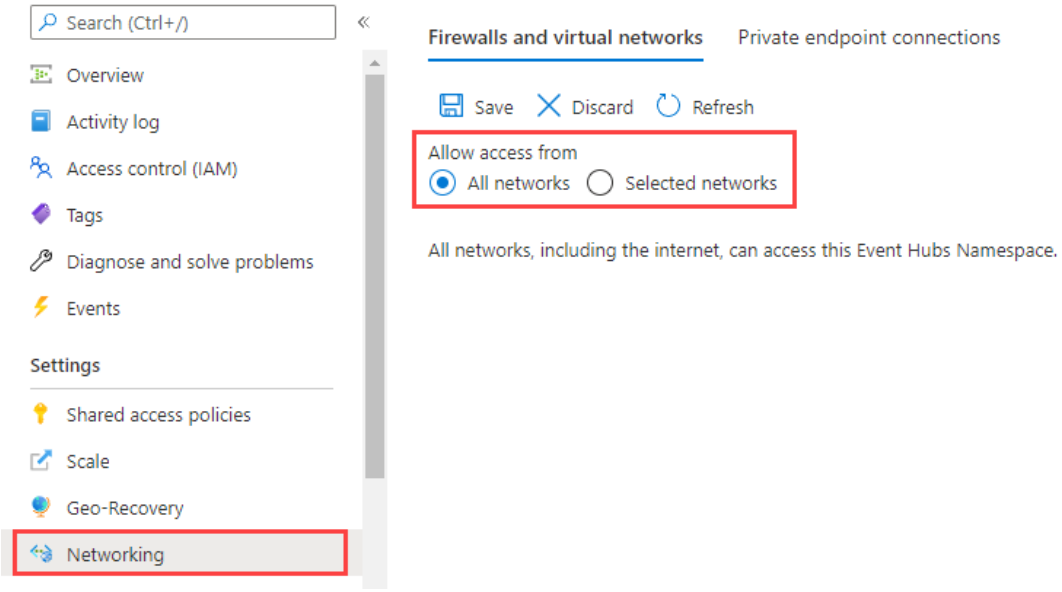
6. Select **Notifications** on the toolbar to monitor the provisioning process. It might take several minutes for the deployment to succeed, but you can move on to the next step now.



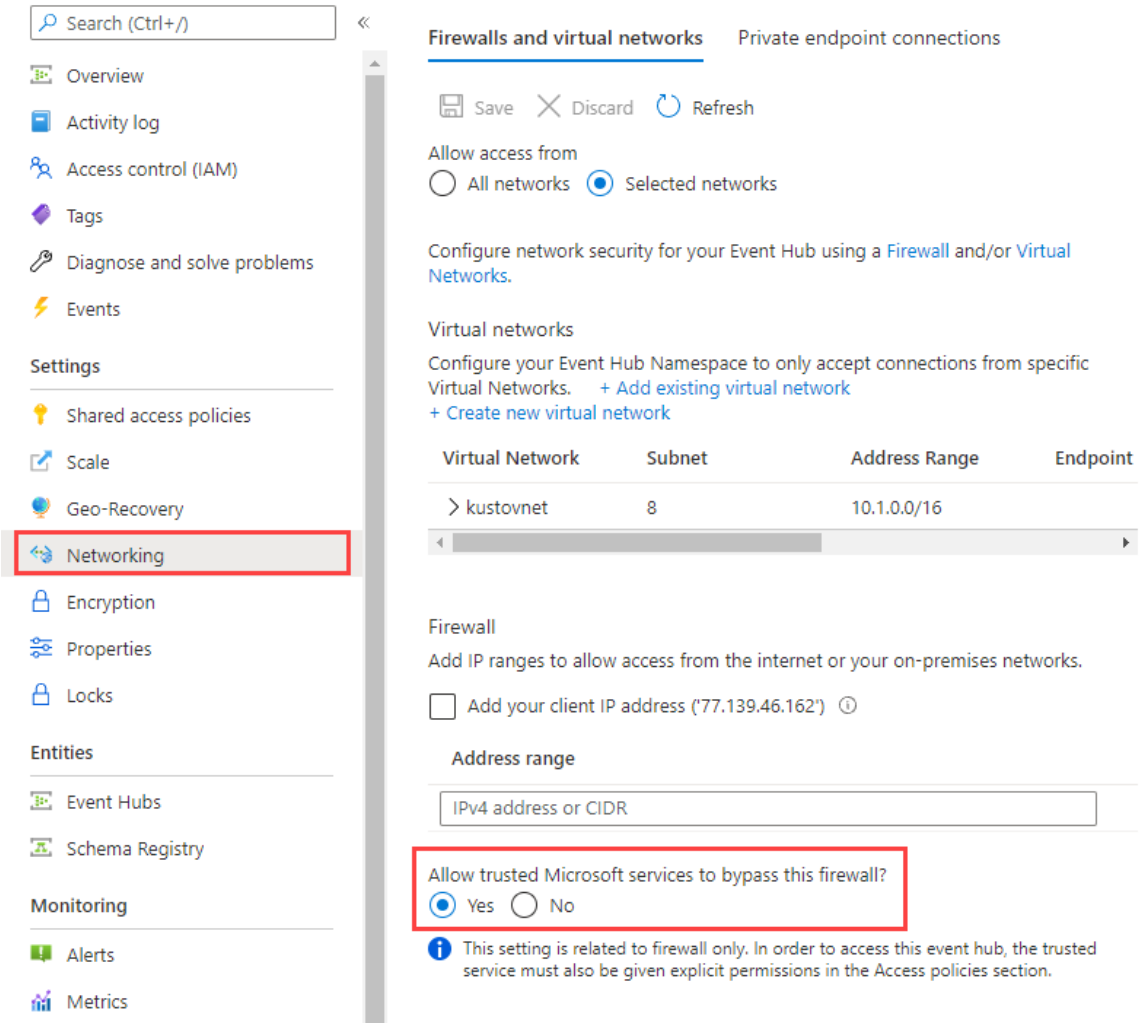
Authentication considerations

Depending on the type of identity, you are using to authenticate with the Event Hub, you may need some additional configurations.

- If you are authenticating with Event Hub using a user assigned managed identity, go to your Event Hub > **Networking**, and then under **Allow access from**, select **All networks** and save the changes.



- If you are authenticating with the Event Hub using a system assigned managed identity, go to your Event Hub > **Networking**, and then either allow access from all networks or under **Allow access from**, select **Selected networks**, select **Allow trusted Microsoft services to bypass this firewall** and save the changes.

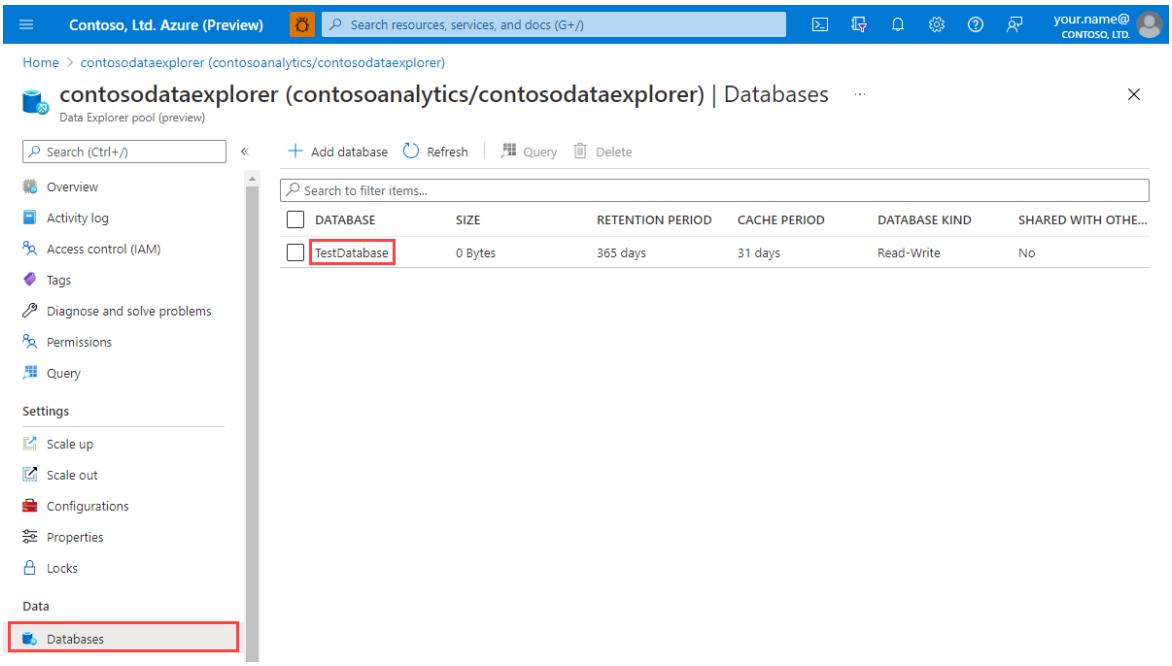


Connect to the Event Hub

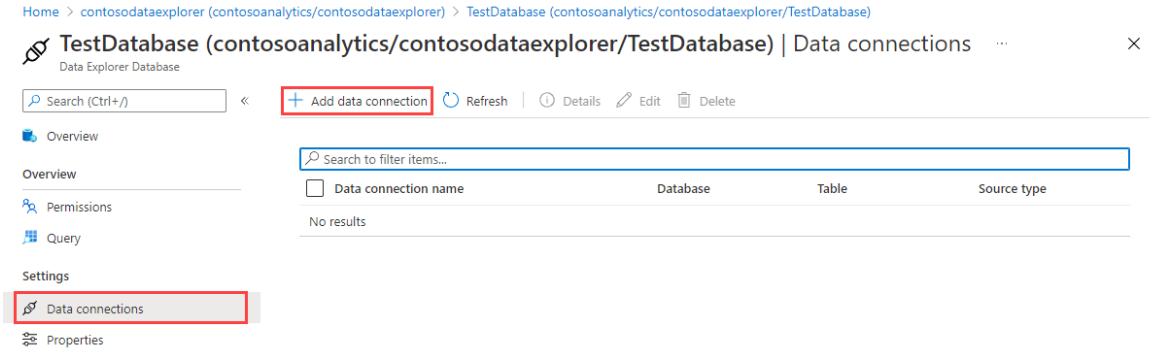
Now you connect to the Event Hub from Data Explorer pool. When this connection is in place, data that flows into the Event Hub streams to the test table you created earlier in this article.

1. Select **Notifications** on the toolbar to verify that the Event Hub deployment was successful.

2. Under the Data Explorer pool you created, select **Databases** > **TestDatabase**.

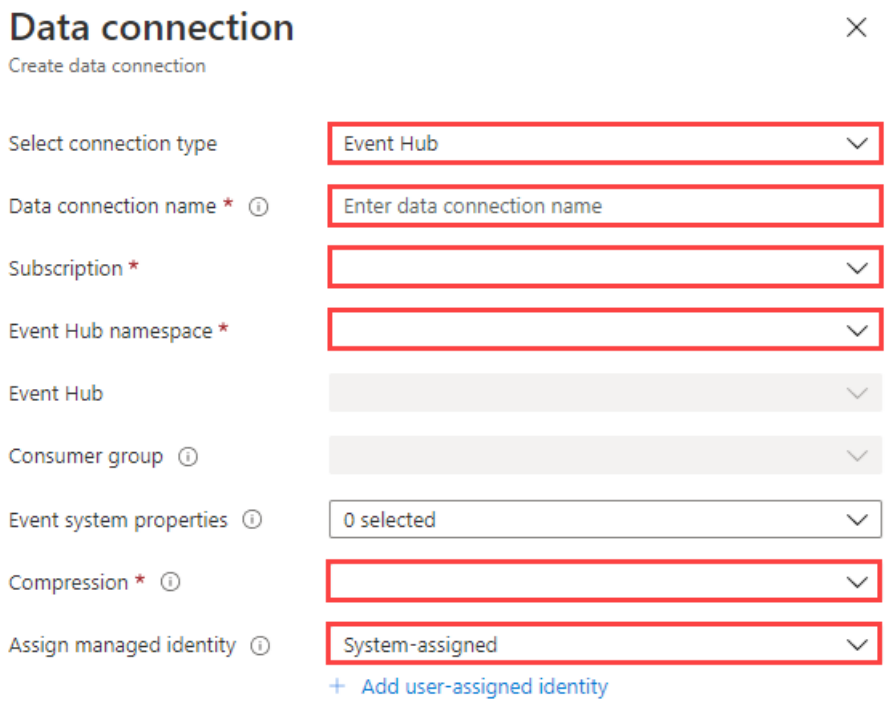


3. Select **Data connections** and **Add data connection**.



Create a data connection (Preview)

Fill out the form with the following information, and then select **Create**.

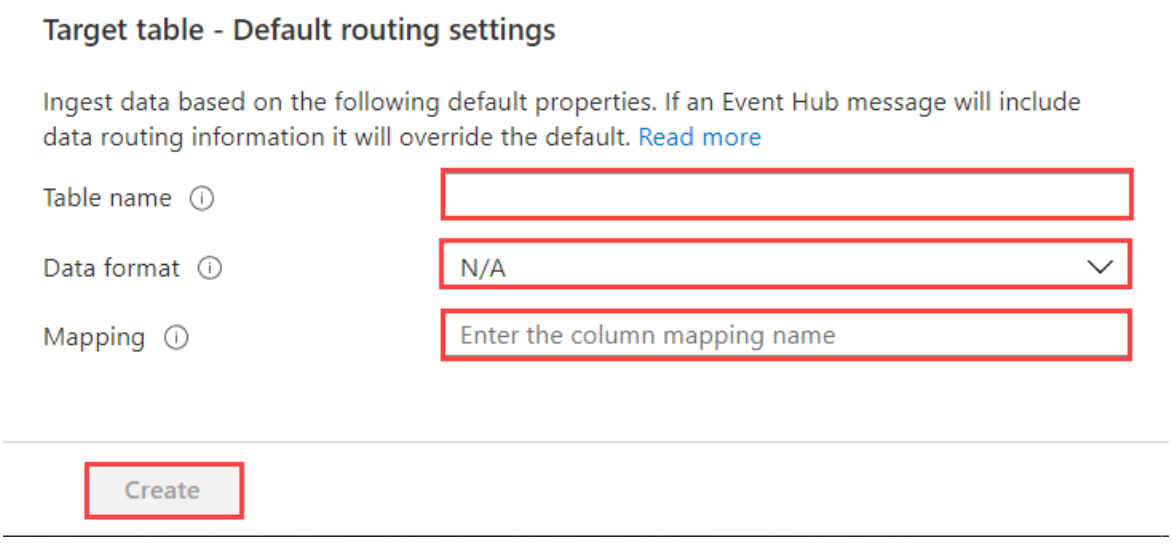


SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Data connection name	*test-hub-connection*	The name of the connection you want to create in Azure Synapse Data Explorer.
Subscription		The subscription ID where the Event Hub resource is located. This field is autopopulated.
Event Hub namespace	A unique namespace name	The name you chose earlier that identifies your namespace.
Event Hub	*test-hub*	The Event Hub you created.
Consumer group	*test-group*	The consumer group defined in the Event Hub you created.
Event system properties	Select relevant properties	The Event Hub system properties. If there are multiple records per event message, the system properties will be added to the first record. When adding system properties, create or update table schema and mapping to include the selected properties.
Compression	*None*	The compression type of the Event Hub messages payload. Supported compression types: *None, Gzip*.
Managed Identity	System-assigned	The managed identity used by the Data Explorer cluster for access to read from the Event Hub. **Note**: When the data connection is created: - *System-assigned* identities are automatically created if they don't exist - The managed identity is automatically assigned the *Azure Event Hubs Data Receiver* role and is added to your Data Explorer cluster. We recommend verifying that the role was assigned and that the identity was added to the cluster.

Target table

There are two options for routing the ingested data: *static* and *dynamic*. For this article, you use static routing, where you specify the table name, data format, and mapping as default values. If the Event Hub message includes data routing information, this routing information will override the default settings.

1. Fill out the following routing settings:



SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Table name	*TestTable*	The table you created in **TestDatabase**.
Data format	*JSON*	Supported formats are Avro, CSV, JSON, MULTILINE JSON, ORC, PARQUET, PSV, SCSV, SOHSV, TSV, TXT, TSVE, APACHEAVRO, and W3CLOG.
Mapping	*TestMapping*	The mapping you created in **TestDatabase**, which maps incoming data to the column names and data types of **TestTable**. Required for JSON, MULTILINE JSON and AVRO, and optional for other formats.

NOTE

- You don't have to specify all **Default routing settings**. Partial settings are also accepted.
- Only events enqueued after you create the data connection are ingested.

2. Select **Create**.

Event system properties mapping

NOTE

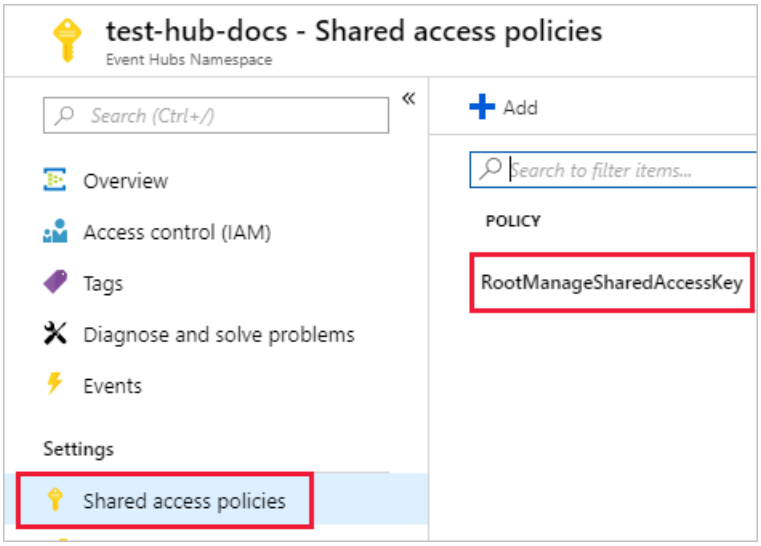
- System properties are supported for `json` and tabular formats (`csv`, `tsv` etc.) and aren't supported on compressed data. When using a non-supported format, the data will still be ingested, but the properties will be ignored.
- For tabular data, system properties are supported only for single-record event messages.
- For JSON data, system properties are also supported for multiple-record event messages. In such cases, the system properties are added only to the first record of the event message.
- For `csv` mapping, properties are added at the beginning of the record in the order listed in the System properties table.
- For `json` mapping, properties are added according to property names in the System properties table.

If you selected **Event system properties** in the **Data Source** section of the table, you must include system properties in the table schema and mapping.

Copy the connection string

When you run the sample app listed in Prerequisites, you need the connection string for the Event Hub namespace.

1. Under the Event Hub namespace you created, select **Shared access policies**, then **RootManageSharedAccessKey**.



2. Copy **Connection string - primary key**. You paste it in the next section.

Generate sample data

Use the sample app you downloaded to generate data.

1. Open the sample app solution in Visual Studio.
2. In the *program.cs* file, update the `eventHubName` constant to the name of your Event Hub and update the `connectionString` constant to the connection string you copied from the Event Hub namespace.

```
const string eventHubName = "test-hub";
// Copy the connection string ("Connection string-primary key") from your Event Hub namespace.
const string connectionString = @"<YourConnectionString>";
```

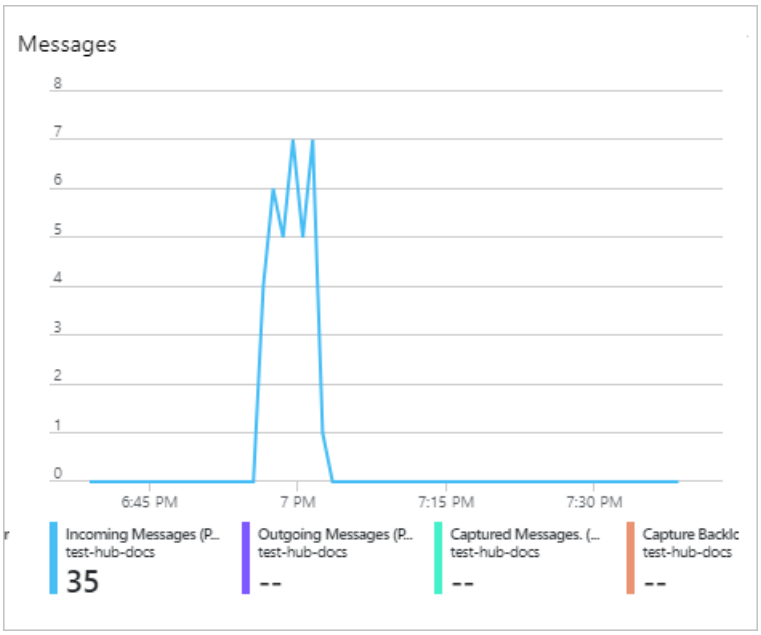
3. Build and run the app. The app sends messages to the Event Hub, and prints its status every 10 seconds.
4. After the app has sent a few messages, move on to the next step: reviewing the flow of data into your

Event Hub and test table.

Review the data flow

With the app generating data, you can now see the flow of that data from the Event Hub to the table in your cluster.

1. In the Azure portal, under your Event Hub, you see the spike in activity while the app is running.



2. To check how many messages have made it to the database so far, run the following query in your test database.

```
TestTable
| count
```

3. To see the content of the messages, run the following query:

```
TestTable
```

The result set should look like the following image:

TestTable | Stats

TimeStamp	Name	Metric	Source
2018-09-18T18:19:17Z	name 0	0	EventHubMessage
2018-09-18T18:19:17Z	name 0	1	EventHubMessage
2018-09-18T18:19:17Z	name 0	2	EventHubMessage
2018-09-18T18:24:29Z	name 3	3	EventHubMessage
2018-09-18T18:24:29Z	name 3	4	EventHubMessage
2018-09-18T18:24:29Z	name 3	5	EventHubMessage
2018-09-18T18:29:39Z	name 6	6	EventHubMessage
2018-09-18T18:29:39Z	name 6	7	EventHubMessage
2018-09-18T18:29:39Z	name 6	8	EventHubMessage

NOTE

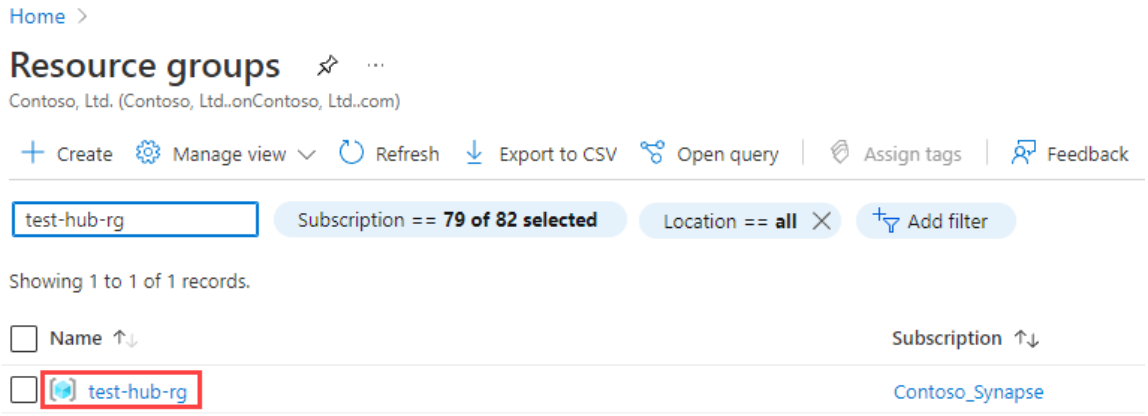
- Azure Synapse Data Explorer has an aggregation (batching) policy for data ingestion, designed to optimize the ingestion process. The default batching policy is configured to seal a batch once one of the following conditions is true for the batch: a maximum delay time of 5 minutes, total size of 1G, or 1000 blobs. Therefore, you may experience a latency. For more information, see batching policy.
- Event Hub ingestion includes Event Hub response time of 10 seconds or 1 MB.
- To reduce response time lag, configure your table to support streaming. See streaming policy.

Clean up resources

If you don't plan to use your Event Hub again, clean up **test-hub-rg**, to avoid incurring costs.

1. In the Azure portal, select **Resource groups** on the far left, and then select the resource group you created.

   If the left menu is collapsed, select » to expand it.



2. Under **test-resource-group**, select **Delete resource group**.
3. In the new window, type the name of the resource group to delete (*test-hub-rg*), and then select **Delete**.

Next steps

- Analyze with Data Explorer
- Monitor Data Explorer pools

Use one-click ingestion to create an Event Hub data connection for Azure Synapse Data Explorer (Preview)

8/19/2022 • 3 minutes to read • Edit Online

Azure Synapse Data Explorer offers ingestion (data loading) from Event Hubs, a big data streaming platform and event ingestion service. Event Hubs can process millions of events per second in near real-time. In this article, you connect an Event Hub to a table in Azure Synapse Data Explorer using the one-click ingestion experience.

Prerequisites

- An Azure subscription. Create a free Azure account.
- Create a Data Explorer pool using Synapse Studio or the Azure portal
- Create a Data Explorer database.
  1. In Synapse Studio, on the left-side pane, select **Data**.
  2. Select **+** (Add new resource) > **Data Explorer pool**, and use the following information:

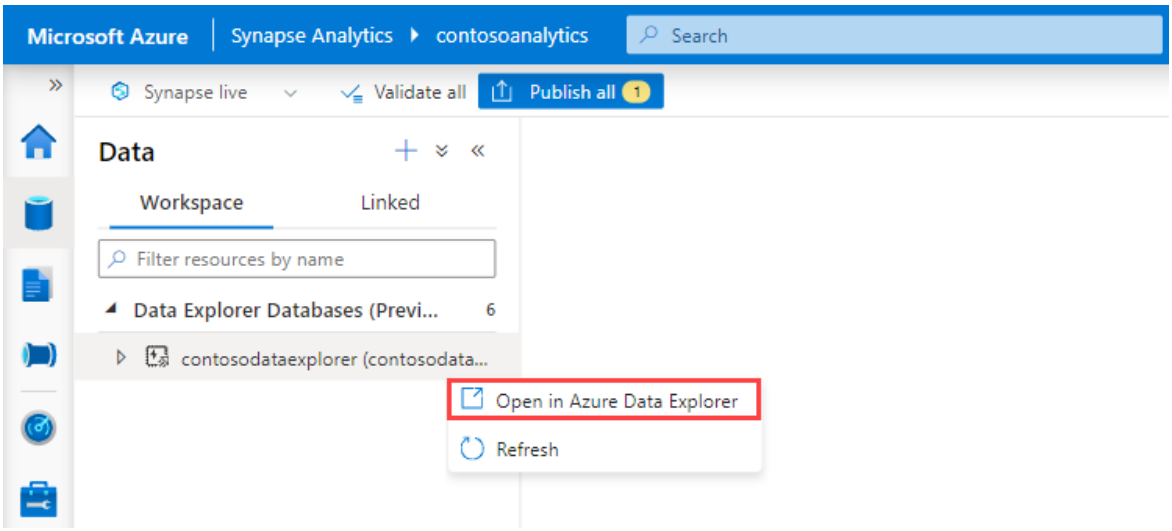
SETTING	SUGGESTED VALUE	DESCRIPTION
Pool name	*contosodataexplorer*	The name of the Data Explorer pool to use
Name	*TestDatabase*	The database name must be unique within the cluster.
Default retention period	*365*	The time span (in days) for which it's guaranteed that the data is kept available to query. The time span is measured from the time that data is ingested.
Default cache period	*31*	The time span (in days) for which to keep frequently queried data available in SSD storage or RAM, rather than in longer-term storage.

  3. Select **Create** to create the database. Creation typically takes less than a minute.
- Event Hub with data for ingestion.

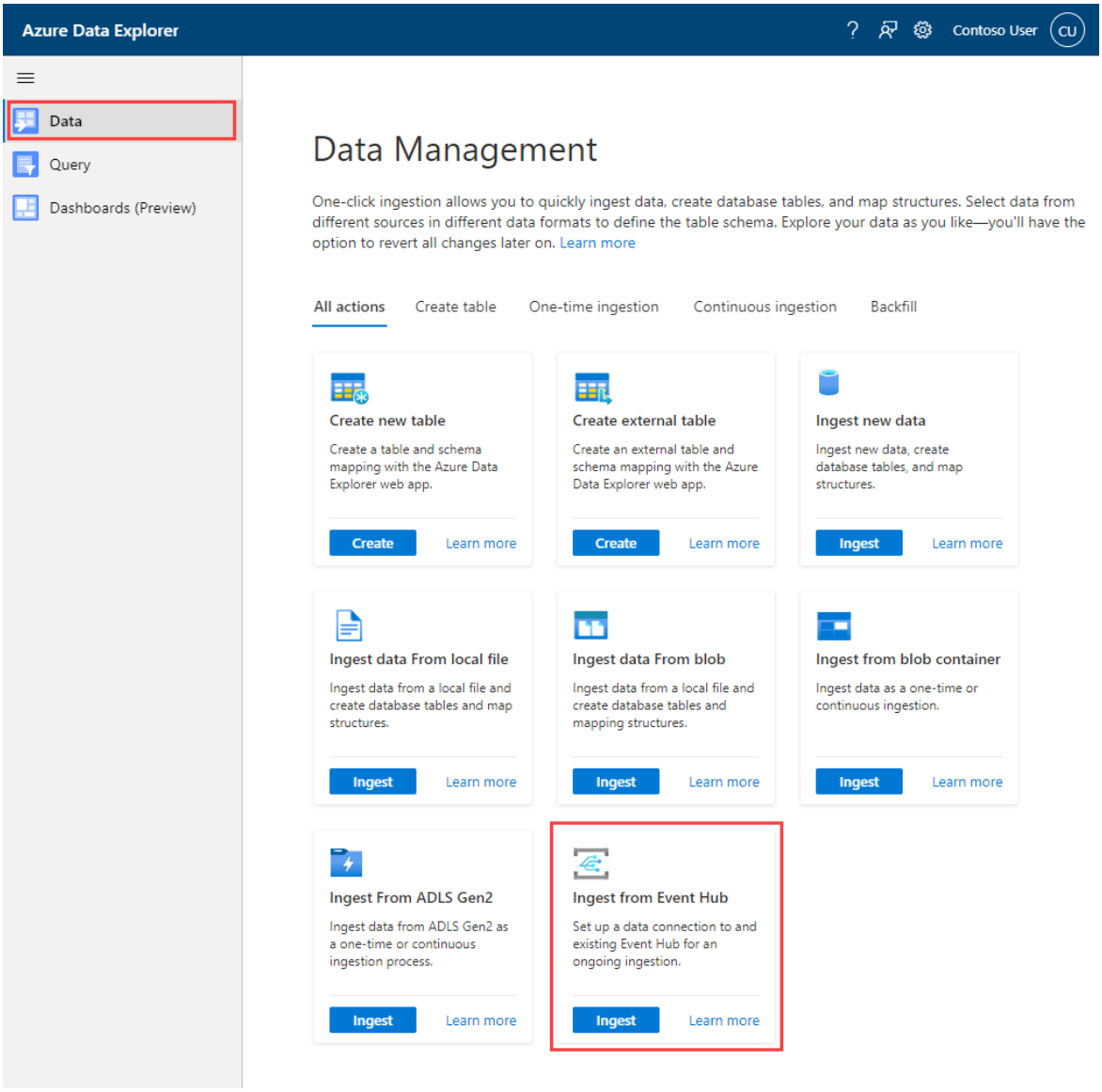
> **NOTE**
> Ingesting data from an Event Hub into Data Explorer pools will not work if your Synapse workspace uses a managed virtual network with data exfiltration protection enabled.

Ingest new data

1. In Synapse Studio, on the left-side pane, select **Data**.
2. Under **Data Explorer Databases**, right-click the relevant database, and then select **Open in Azure Data Explorer**.



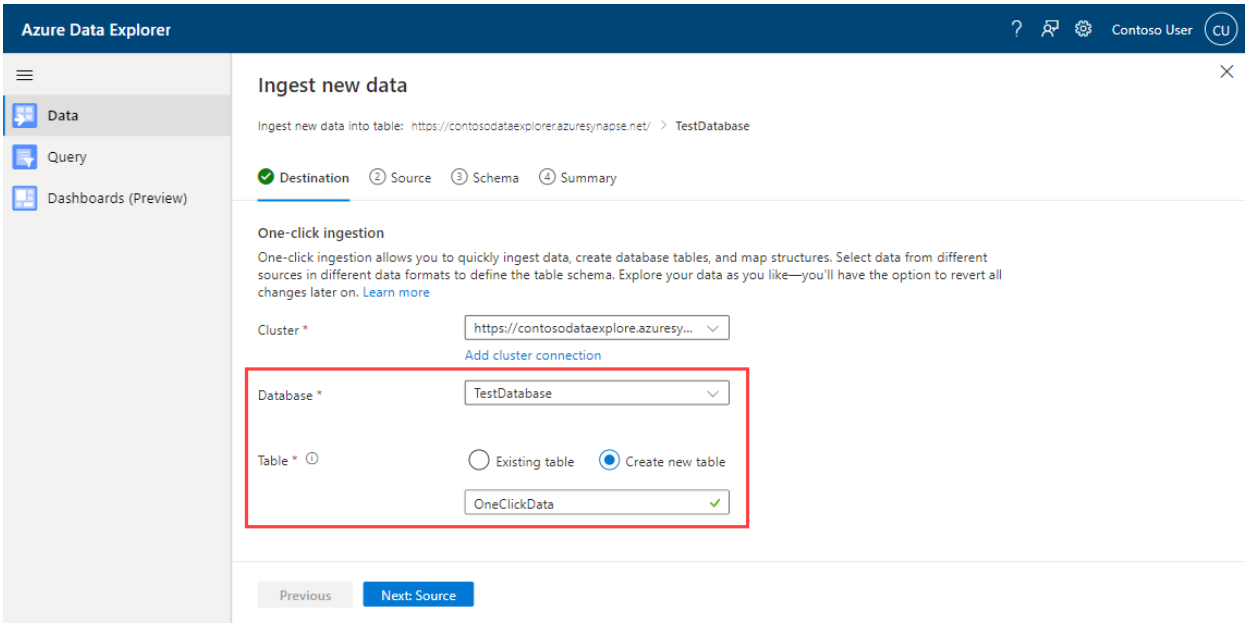
3. In the left menu of the Web UI, select the **Data** tab.



4. In the **Ingest data from Event Hub** card, select **Ingest**.

The **Ingest new data** window opens with the **Destination** tab selected.

Destination tab



1. The **Cluster** and **Database** fields are auto-populated. You may select a different cluster or database from the drop-down menus.

2. Under **Table**, select **Create new table** and enter a name for the new table. Alternatively, use an existing table.

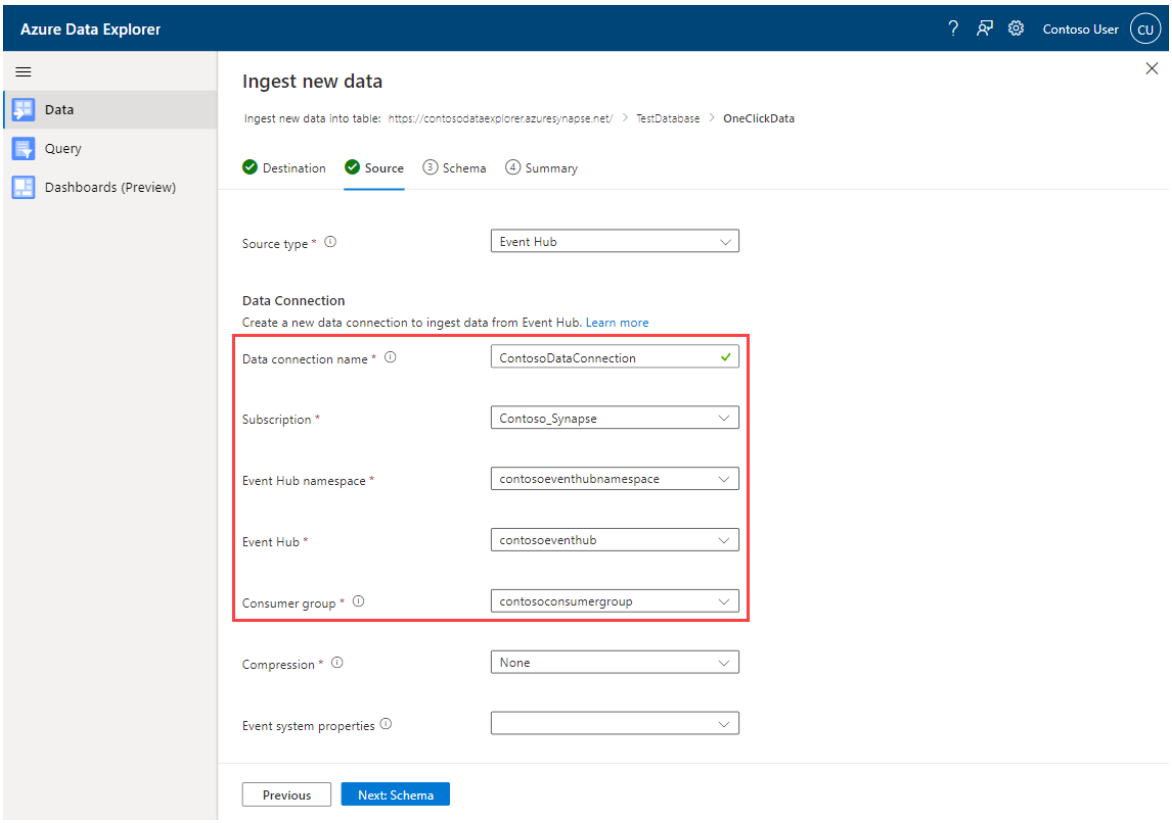
   > **NOTE**
   > Table names must be between 1 and 1024 characters. You can use alphanumeric, hyphens, and underscores. Special characters aren't supported.

3. Select **Next: Source**.

Source tab

1. Under **Source type**, select **Event Hub**.

2. Under **Data Connection**, fill in the following fields:

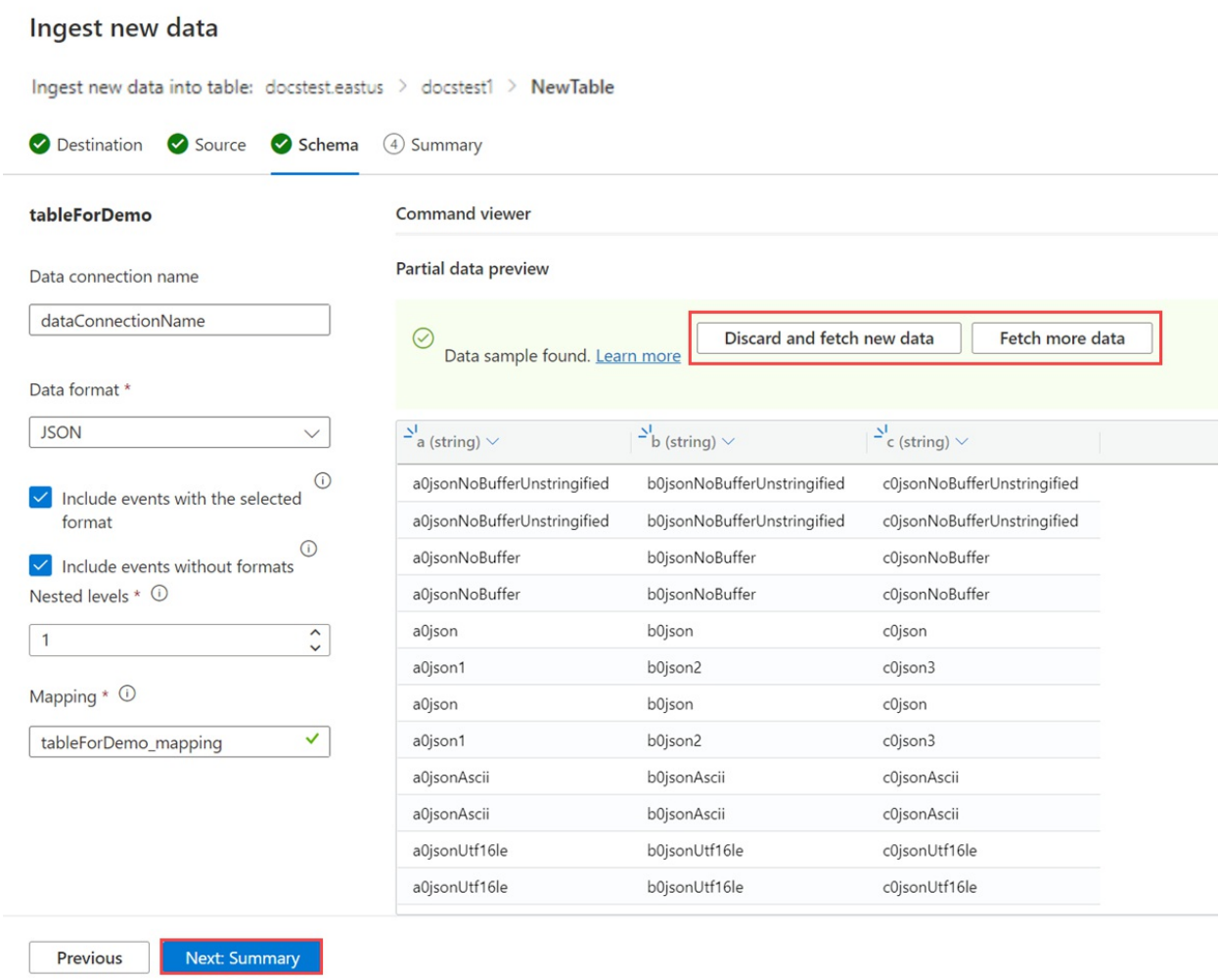


SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Data connection name	*ContosoDataConnection*	The name that identifies your data connection.
Subscription		The subscription ID where the Event Hub resource is located.
Event Hub namespace		The name that identifies your namespace.
Event Hub		The Event Hub you wish to use.
Consumer group		The consumer group defined in your Event Hub.
Event system properties	Select relevant properties	The Event Hub system properties. If there are multiple records per event message, the system properties will be added to the first one. When adding system properties, create or update table schema and mapping to include the selected properties.

3. Select **Next: Schema**.

Schema tab

Data is read from the Event Hub in form of EventData objects. Supported formats are CSV, JSON, PSV, SCsv, SOHsv TSV, TXT, and TSVE.



1. If the data you see in the preview window is not complete, you may need more data to create a table with all necessary data fields. Use the following commands to fetch new data from your Event Hub:

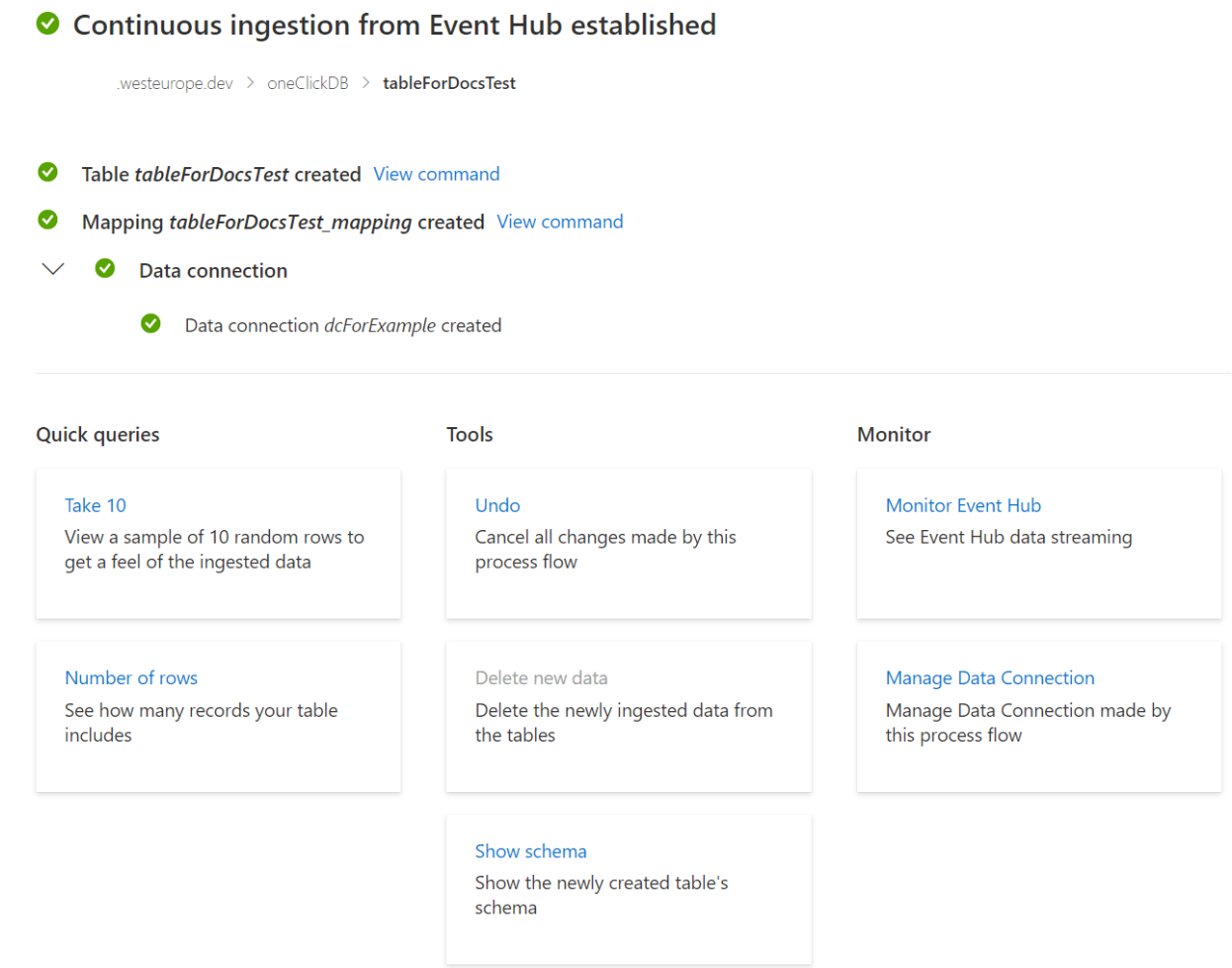
   - **Discard and fetch new data**: discards the data presented and searches for new events.
   - **Fetch more data**: Searches for more events in addition to the events already found.

   > **NOTE**
   > To see a preview of your data, your Event Hub must be sending events.

2. Select **Next: Summary**.

Continuous ingestion from Event Hub

In the **Continuous ingestion from Event Hub established** window, all steps will be marked with green check marks when establishment finishes successfully. The cards below these steps give you options to explore your data with **Quick queries**, undo changes made using **Tools**, or **Monitor** the Event Hub connections and data.



Next steps

- Analyze with Data Explorer
- Monitor Data Explorer pools

Create an Event Hub data connection for Azure Synapse Data Explorer by using C# (Preview)

8/19/2022 • 7 minutes to read • Edit Online

Azure Synapse Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Synapse Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers.

In this article, you create an Event Hub data connection for Azure Synapse Data Explorer by using C#.

Prerequisites

- An Azure subscription. Create a free Azure account.
- Create a Data Explorer pool using Synapse Studio or the Azure portal
- Create a Data Explorer database.
  1. In Synapse Studio, on the left-side pane, select **Data**.
  2. Select **+** (Add new resource) > **Data Explorer pool**, and use the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Pool name	*contosodataexplorer*	The name of the Data Explorer pool to use
Name	*TestDatabase*	The database name must be unique within the cluster.
Default retention period	*365*	The time span (in days) for which it's guaranteed that the data is kept available to query. The time span is measured from the time that data is ingested.
Default cache period	*31*	The time span (in days) for which to keep frequently queried data available in SSD storage or RAM, rather than in longer-term storage.

  3. Select **Create** to create the database. Creation typically takes less than a minute.
- Event Hub with data for ingestion.

> **NOTE**
> Ingesting data from an Event Hub into Data Explorer pools will not work if your Synapse workspace uses a managed virtual network with data exfiltration protection enabled.

- Visual Studio 2019, download and use the **free** Visual Studio 2019 Community Edition. Enable **Azure development** during the Visual Studio setup.

Create a table on your test cluster

Create a table named `StormEvents` that matches the schema of the data in the `StormEvents.csv` file.

> **TIP**
> The following code snippets create an instance of a client for almost every call. This is done to make each snippet individually runnable. In production, the client instances are reentrant, and should be kept as long as needed. A single client instance per URI is sufficient, even when working with multiple databases (database can be specified on a command level).

```
var databaseName = "<DatabaseName>";
var table = "StormEvents";
using (var kustoClient = KustoClientFactory.CreateCslAdminProvider(kustoConnectionStringBuilder))
{
    var command =
        CslCommandGenerator.GenerateTableCreateCommand(
            table,
            new[]
            {
                Tuple.Create("StartTime", "System.DateTime"),
                Tuple.Create("EndTime", "System.DateTime"),
                Tuple.Create("EpisodeId", "System.Int32"),
                Tuple.Create("EventId", "System.Int32"),
                Tuple.Create("State", "System.String"),
                Tuple.Create("EventType", "System.String"),
                Tuple.Create("InjuriesDirect", "System.Int32"),
                Tuple.Create("InjuriesIndirect", "System.Int32"),
                Tuple.Create("DeathsDirect", "System.Int32"),
                Tuple.Create("DeathsIndirect", "System.Int32"),
                Tuple.Create("DamageProperty", "System.Int32"),
                Tuple.Create("DamageCrops", "System.Int32"),
                Tuple.Create("Source", "System.String"),
                Tuple.Create("BeginLocation", "System.String"),
                Tuple.Create("EndLocation", "System.String"),
                Tuple.Create("BeginLat", "System.Double"),
                Tuple.Create("BeginLon", "System.Double"),
                Tuple.Create("EndLat", "System.Double"),
                Tuple.Create("EndLon", "System.Double"),
                Tuple.Create("EpisodeNarrative", "System.String"),
                Tuple.Create("EventNarrative", "System.String"),
                Tuple.Create("StormSummary", "System.Object"),
            });

    kustoClient.ExecuteControlCommand(databaseName, command);
}
```

Define ingestion mapping

Map the incoming CSV data to the column names used when creating the table. Provision a CSV column mapping object on that table.

```
var tableMapping = "StormEvents_CSV_Mapping";
using (var kustoClient = KustoClientFactory.CreateCslAdminProvider(kustoConnectionStringBuilder))
{
    var command =
        CslCommandGenerator.GenerateTableMappingCreateCommand(
            Data.Ingestion.IngestionMappingKind.Csv,
            table,
            tableMapping,
            new[] {
                new ColumnMapping() { ColumnName = "StartTime", Properties = new Dictionary<string, string>
() { { MappingConsts.Ordinal, "0" } } },
                new ColumnMapping() { ColumnName = "EndTime", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "1" } } },
                new ColumnMapping() { ColumnName = "EpisodeId", Properties = new Dictionary<string, string>
() { { MappingConsts.Ordinal, "2" } } },
                new ColumnMapping() { ColumnName = "EventId", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "3" } } },
                new ColumnMapping() { ColumnName = "State", Properties =  new Dictionary<string, string>() {
{ MappingConsts.Ordinal, "4" } } },
                new ColumnMapping() { ColumnName = "EventType", Properties =  new Dictionary<string, string>
() { { MappingConsts.Ordinal, "5" } } },
                new ColumnMapping() { ColumnName = "InjuriesDirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "6" } } },
                new ColumnMapping() { ColumnName = "InjuriesIndirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "7" } } },
                new ColumnMapping() { ColumnName = "DeathsDirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "8" } } },
                new ColumnMapping() { ColumnName = "DeathsIndirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "9" } } },
                new ColumnMapping() { ColumnName = "DamageProperty", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "10" } } },
                new ColumnMapping() { ColumnName = "DamageCrops", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "11" } } },
                new ColumnMapping() { ColumnName = "Source", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "12" } } },
                new ColumnMapping() { ColumnName = "BeginLocation", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "13" } } },
                new ColumnMapping() { ColumnName = "EndLocation", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "14" } } },
                new ColumnMapping() { ColumnName = "BeginLat", Properties =  new Dictionary<string, string>
() { { MappingConsts.Ordinal, "15" } } },
                new ColumnMapping() { ColumnName = "BeginLon", Properties =  new Dictionary<string, string>
() { { MappingConsts.Ordinal, "16" } } },
                new ColumnMapping() { ColumnName = "EndLat", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "17" } } },
                new ColumnMapping() { ColumnName = "EndLon", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "18" } } },
                new ColumnMapping() { ColumnName = "EpisodeNarrative", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "19" } } },
                new ColumnMapping() { ColumnName = "EventNarrative", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "20" } } },
                new ColumnMapping() { ColumnName = "StormSummary", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "21" } } }
        });

    kustoClient.ExecuteControlCommand(databaseName, command);
}
```

Install C# NuGet

- Install the Microsoft.Azure.Management.Kusto NuGet package.

Authentication

To run the following example, you need an Azure Active Directory (Azure AD) application and service principal that can access resources. To create a free Azure AD application and add role assignment at the subscription level, see Create an Azure AD application. You also need the directory (tenant) ID, application ID, and client secret.

Add an Event Hub data connection

The following example shows you how to add an Event Hub data connection programmatically. See connect to the Event Hub for information about adding an Event Hub data connection using the Azure portal.

```
var tenantId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx";//Directory (tenant) ID
var clientId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx";//Application ID
var clientSecret = "xxxxxxxxxxxxx";//Client Secret
var subscriptionId = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx";
var authenticationContext = new AuthenticationContext($"https://login.windows.net/{tenantId}");
var credential = new ClientCredential(clientId, clientSecret);
var result = await authenticationContext.AcquireTokenAsync(resource: "https://management.core.windows.net/",
clientCredential: credential);

var credentials = new TokenCredentials(result.AccessToken, result.AccessTokenType);

var kustoManagementClient = new KustoManagementClient(credentials)
{
    SubscriptionId = subscriptionId
};

var resourceGroupName = "testrg";
//The cluster and database that are created as part of the Prerequisites
var clusterName = "mykustocluster";
var databaseName = "mykustodatabase";
var dataConnectionName = "myeventhubconnect";
//The Event Hub that is created as part of the Prerequisites
var eventHubResourceId = "/subscriptions/xxxxxxxx-xxxxx-xxxx-xxxx-
xxxxxxxxx/resourceGroups/xxxxxx/providers/Microsoft.EventHub/namespaces/xxxxxx/eventhubs/xxxxxx";
var consumerGroup = "$Default";
var location = "Central US";
//The table and column mapping are created as part of the Prerequisites
var tableName = "StormEvents";
var mappingRuleName = "StormEvents_CSV_Mapping";
var dataFormat = DataFormat.CSV;
var compression = "None";
await kustoManagementClient.DataConnections.CreateOrUpdateAsync(resourceGroupName, clusterName,
databaseName, dataConnectionName,
    new EventHubDataConnection(eventHubResourceId, consumerGroup, location: location, tableName: tableName,
mappingRuleName: mappingRuleName, dataFormat: dataFormat, compression: compression));
```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
tenantId	*xxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx*	Your tenant ID. Also known as directory ID.
subscriptionId	*xxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx*	The subscription ID that you use for resource creation.
clientId	*xxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx*	The client ID of the application that can access resources in your tenant.
clientSecret	*xxxxxxxxxxxxx*	The client secret of the application that can access resources in your tenant.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
resourceGroupName	*testrg*	The name of the resource group containing your cluster.
clusterName	*mykustocluster*	The name of your cluster.
databaseName	*mykustodatabase*	The name of the target database in your cluster.
dataConnectionName	*myeventhubconnect*	The desired name of your data connection.
tableName	*StormEvents*	The name of the target table in the target database.
mappingRuleName	*StormEvents_CSV_Mapping*	The name of your column mapping related to the target table.
dataFormat	*csv*	The data format of the message.
eventHubResourceId	*Resource ID*	The resource ID of your Event Hub that holds the data for ingestion.
consumerGroup	*$Default*	The consumer group of your Event Hub.
location	*Central US*	The location of the data connection resource.
compression	*Gzip* or *None*	The type of data compression.

Generate data

See the sample app that generates data and sends it to an Event Hub.

An event can contain one or more records, up to its size limit. In the following sample we send two events, each has five records appended:

```
var events = new List<EventData>();
var data = string.Empty;
var recordsPerEvent = 5;
var rand = new Random();
var counter = 0;

for (var i = 0; i < 10; i++)
{
    // Create the data
    var metric = new Metric { Timestamp = DateTime.UtcNow, MetricName = "Temperature", Value = rand.Next(-
30, 50) };
    var data += JsonConvert.SerializeObject(metric) + Environment.NewLine;
    counter++;

    // Create the event
    if (counter == recordsPerEvent)
    {
        var eventData = new EventData(Encoding.UTF8.GetBytes(data));
        events.Add(eventData);

        counter = 0;
        data = string.Empty;
    }
}

// Send events
eventHubClient.SendAsync(events).Wait();
```

Clean up resources

To delete the data connection, use the following command:

```
kustoManagementClient.DataConnections.Delete(resourceGroupName, clusterName, databaseName,
dataConnectionName);
```

Next steps

- Analyze with Data Explorer
- Monitor Data Explorer pools

Create an Event Hub data connection for Azure Synapse Data Explorer by using Python (Preview)

8/19/2022 • 6 minutes to read • Edit Online

Azure Synapse Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Synapse Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers.

In this article, you create an Event Hub data connection for Azure Synapse Data Explorer by using Python.

Prerequisites

- An Azure subscription. Create a free Azure account.
- Create a Data Explorer pool using Synapse Studio or the Azure portal
- Create a Data Explorer database.

  1. In Synapse Studio, on the left-side pane, select **Data**.
  2. Select **+** (Add new resource) > **Data Explorer pool**, and use the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Pool name	*contosodataexplorer*	The name of the Data Explorer pool to use
Name	*TestDatabase*	The database name must be unique within the cluster.
Default retention period	*365*	The time span (in days) for which it's guaranteed that the data is kept available to query. The time span is measured from the time that data is ingested.
Default cache period	*31*	The time span (in days) for which to keep frequently queried data available in SSD storage or RAM, rather than in longer-term storage.

  3. Select **Create** to create the database. Creation typically takes less than a minute.

- Event Hub with data for ingestion.
- Python 3.4+.

Create a table on your test cluster

Create a table named `StormEvents` that matches the schema of the data in the `StormEvents.csv` file.

TIP

The following code snippets create an instance of a client for almost every call. This is done to make each snippet individually runnable. In production, the client instances are reentrant, and should be kept as long as needed. A single client instance per URI is sufficient, even when working with multiple databases (database can be specified on a command level).

```
var databaseName = "<DatabaseName>";
var table = "StormEvents";
using (var kustoClient = KustoClientFactory.CreateCslAdminProvider(kustoConnectionStringBuilder))
{
    var command =
        CslCommandGenerator.GenerateTableCreateCommand(
            table,
            new[]
            {
                Tuple.Create("StartTime", "System.DateTime"),
                Tuple.Create("EndTime", "System.DateTime"),
                Tuple.Create("EpisodeId", "System.Int32"),
                Tuple.Create("EventId", "System.Int32"),
                Tuple.Create("State", "System.String"),
                Tuple.Create("EventType", "System.String"),
                Tuple.Create("InjuriesDirect", "System.Int32"),
                Tuple.Create("InjuriesIndirect", "System.Int32"),
                Tuple.Create("DeathsDirect", "System.Int32"),
                Tuple.Create("DeathsIndirect", "System.Int32"),
                Tuple.Create("DamageProperty", "System.Int32"),
                Tuple.Create("DamageCrops", "System.Int32"),
                Tuple.Create("Source", "System.String"),
                Tuple.Create("BeginLocation", "System.String"),
                Tuple.Create("EndLocation", "System.String"),
                Tuple.Create("BeginLat", "System.Double"),
                Tuple.Create("BeginLon", "System.Double"),
                Tuple.Create("EndLat", "System.Double"),
                Tuple.Create("EndLon", "System.Double"),
                Tuple.Create("EpisodeNarrative", "System.String"),
                Tuple.Create("EventNarrative", "System.String"),
                Tuple.Create("StormSummary", "System.Object"),
            });

    kustoClient.ExecuteControlCommand(databaseName, command);
}
```

Define ingestion mapping

Map the incoming CSV data to the column names used when creating the table. Provision a CSV column mapping object on that table.

```
var tableMapping = "StormEvents_CSV_Mapping";
using (var kustoClient = KustoClientFactory.CreateCslAdminProvider(kustoConnectionStringBuilder))
{
    var command =
        CslCommandGenerator.GenerateTableMappingCreateCommand(
            Data.Ingestion.IngestionMappingKind.Csv,
            table,
            tableMapping,
            new[] {
                new ColumnMapping() { ColumnName = "StartTime", Properties = new Dictionary<string, string>
() { { MappingConsts.Ordinal, "0" } } },
                new ColumnMapping() { ColumnName = "EndTime", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "1" } } },
                new ColumnMapping() { ColumnName = "EpisodeId", Properties = new Dictionary<string, string>
() { { MappingConsts.Ordinal, "2" } } },
                new ColumnMapping() { ColumnName = "EventId", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "3" } } },
                new ColumnMapping() { ColumnName = "State", Properties =  new Dictionary<string, string>() {
{ MappingConsts.Ordinal, "4" } } },
                new ColumnMapping() { ColumnName = "EventType", Properties =  new Dictionary<string, string>
() { { MappingConsts.Ordinal, "5" } } },
                new ColumnMapping() { ColumnName = "InjuriesDirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "6" } } },
                new ColumnMapping() { ColumnName = "InjuriesIndirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "7" } } },
                new ColumnMapping() { ColumnName = "DeathsDirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "8" } } },
                new ColumnMapping() { ColumnName = "DeathsIndirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "9" } } },
                new ColumnMapping() { ColumnName = "DamageProperty", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "10" } } },
                new ColumnMapping() { ColumnName = "DamageCrops", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "11" } } },
                new ColumnMapping() { ColumnName = "Source", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "12" } } },
                new ColumnMapping() { ColumnName = "BeginLocation", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "13" } } },
                new ColumnMapping() { ColumnName = "EndLocation", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "14" } } },
                new ColumnMapping() { ColumnName = "BeginLat", Properties =  new Dictionary<string, string>
() { { MappingConsts.Ordinal, "15" } } },
                new ColumnMapping() { ColumnName = "BeginLon", Properties =  new Dictionary<string, string>
() { { MappingConsts.Ordinal, "16" } } },
                new ColumnMapping() { ColumnName = "EndLat", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "17" } } },
                new ColumnMapping() { ColumnName = "EndLon", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "18" } } },
                new ColumnMapping() { ColumnName = "EpisodeNarrative", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "19" } } },
                new ColumnMapping() { ColumnName = "EventNarrative", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "20" } } },
                new ColumnMapping() { ColumnName = "StormSummary", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "21" } } }
        });

    kustoClient.ExecuteControlCommand(databaseName, command);
}
```

Install Python package

To install the Python package for Azure Synapse Data Explorer, open a command prompt that has Python in its path. Run the following command:

```
pip install azure-common
pip install azure-mgmt-kusto
```

Authentication

To run the following example, you need an Azure Active Directory (Azure AD) application and service principal that can access resources. To create a free Azure AD application and add role assignment at the subscription level, see Create an Azure AD application. You also need the directory (tenant) ID, application ID, and client secret.

Add an Event Hub data connection

The following example shows you how to add an Event Hub data connection programmatically. See connect to the event hub for adding an Event Hub data connection using the Azure portal.

```
from azure.mgmt.kusto import KustoManagementClient
from azure.mgmt.kusto.models import EventHubDataConnection
from azure.common.credentials import ServicePrincipalCredentials

#Directory (tenant) ID
tenant_id = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx"
#Application ID
client_id = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx"
#Client Secret
client_secret = "xxxxxxxxxxxxx"
subscription_id = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx"
credentials = ServicePrincipalCredentials(
        client_id=client_id,
        secret=client_secret,
        tenant=tenant_id
    )
kusto_management_client = KustoManagementClient(credentials, subscription_id)

resource_group_name = "testrg"
#The cluster and database that are created as part of the Prerequisites
cluster_name = "mykustocluster"
database_name = "mykustodatabase"
data_connection_name = "myeventhubconnect"
#The event hub that is created as part of the Prerequisites
event_hub_resource_id = "/subscriptions/xxxxxxxx-xxxxx-xxxx-xxxx-
xxxxxxxxx/resourceGroups/xxxxxx/providers/Microsoft.EventHub/namespaces/xxxxxx/eventhubs/xxxxxx";
consumer_group = "$Default"
location = "Central US"
#The table and column mapping that are created as part of the Prerequisites
table_name = "StormEvents"
mapping_rule_name = "StormEvents_CSV_Mapping"
data_format = "csv"
#Returns an instance of LROPoller, check
https://docs.microsoft.com/python/api/msrest/msrest.polling.lropoller?view=azure-python
poller = kusto_management_client.data_connections.create_or_update(resource_group_name=resource_group_name,
cluster_name=cluster_name, database_name=database_name, data_connection_name=data_connection_name,

parameters=EventHubDataConnection(event_hub_resource_id=event_hub_resource_id,
consumer_group=consumer_group, location=location,
                                                                    table_name=table_name,
mapping_rule_name=mapping_rule_name, data_format=data_format))
```

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
tenant_id	*xxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxx*	Your tenant ID. Also known as directory ID.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
subscriptionId	*xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx*	The subscription ID that you use for resource creation.
client_id	*xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx*	The client ID of the application that can access resources in your tenant.
client_secret	*xxxxxxxxxxxx*	The client secret of the application that can access resources in your tenant.
resource_group_name	*testrg*	The name of the resource group containing your cluster.
cluster_name	*mykustocluster*	The name of your cluster.
database_name	*mykustodatabase*	The name of the target database in your cluster.
data_connection_name	*myeventhubconnect*	The desired name of your data connection.
table_name	*StormEvents*	The name of the target table in the target database.
mapping_rule_name	*StormEvents_CSV_Mapping*	The name of your column mapping related to the target table.
data_format	*csv*	The data format of the message.
event_hub_resource_id	*Resource ID*	The resource ID of your Event Hub that holds the data for ingestion.
consumer_group	*$Default*	The consumer group of your Event Hub.
location	*Central US*	The location of the data connection resource.

Clean up resources

To delete the data connection, use the following command:

```
kusto_management_client.data_connections.delete(resource_group_name=resource_group_name,
cluster_name=kusto_cluster_name, database_name=kusto_database_name,
data_connection_name=kusto_data_connection_name)
```

Next steps

- Analyze with Data Explorer
- Monitor Data Explorer pools

Create an Event Hub data connection for Azure Synapse Data Explorer by using Azure Resource Manager template (Preview)

8/19/2022 • 6 minutes to read • Edit Online

Azure Synapse Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Synapse Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers.

In this article, you create an Event Hub data connection for Azure Synapse Data Explorer by using Azure Resource Manager template.

Prerequisites

- An Azure subscription. Create a free Azure account.
- Create a Data Explorer pool using Synapse Studio or the Azure portal
- Create a Data Explorer database.
  1. In Synapse Studio, on the left-side pane, select **Data**.
  2. Select **+** (Add new resource) > **Data Explorer pool**, and use the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Pool name	*contosodataexplorer*	The name of the Data Explorer pool to use
Name	*TestDatabase*	The database name must be unique within the cluster.
Default retention period	*365*	The time span (in days) for which it's guaranteed that the data is kept available to query. The time span is measured from the time that data is ingested.
Default cache period	*31*	The time span (in days) for which to keep frequently queried data available in SSD storage or RAM, rather than in longer-term storage.

  3. Select **Create** to create the database. Creation typically takes less than a minute.
- Event Hub with data for ingestion.

Create a table on your test cluster

Create a table named `StormEvents` that matches the schema of the data in the `StormEvents.csv` file.

TIP

The following code snippets create an instance of a client for almost every call. This is done to make each snippet individually runnable. In production, the client instances are reentrant, and should be kept as long as needed. A single client instance per URI is sufficient, even when working with multiple databases (database can be specified on a command level).

```
var databaseName = "<DatabaseName>";
var table = "StormEvents";
using (var kustoClient = KustoClientFactory.CreateCslAdminProvider(kustoConnectionStringBuilder))
{
    var command =
        CslCommandGenerator.GenerateTableCreateCommand(
            table,
            new[]
            {
                Tuple.Create("StartTime", "System.DateTime"),
                Tuple.Create("EndTime", "System.DateTime"),
                Tuple.Create("EpisodeId", "System.Int32"),
                Tuple.Create("EventId", "System.Int32"),
                Tuple.Create("State", "System.String"),
                Tuple.Create("EventType", "System.String"),
                Tuple.Create("InjuriesDirect", "System.Int32"),
                Tuple.Create("InjuriesIndirect", "System.Int32"),
                Tuple.Create("DeathsDirect", "System.Int32"),
                Tuple.Create("DeathsIndirect", "System.Int32"),
                Tuple.Create("DamageProperty", "System.Int32"),
                Tuple.Create("DamageCrops", "System.Int32"),
                Tuple.Create("Source", "System.String"),
                Tuple.Create("BeginLocation", "System.String"),
                Tuple.Create("EndLocation", "System.String"),
                Tuple.Create("BeginLat", "System.Double"),
                Tuple.Create("BeginLon", "System.Double"),
                Tuple.Create("EndLat", "System.Double"),
                Tuple.Create("EndLon", "System.Double"),
                Tuple.Create("EpisodeNarrative", "System.String"),
                Tuple.Create("EventNarrative", "System.String"),
                Tuple.Create("StormSummary", "System.Object"),
            });

    kustoClient.ExecuteControlCommand(databaseName, command);
}
```

Define ingestion mapping

Map the incoming CSV data to the column names used when creating the table. Provision a CSV column mapping object on that table.

```
var tableMapping = "StormEvents_CSV_Mapping";
using (var kustoClient = KustoClientFactory.CreateCslAdminProvider(kustoConnectionStringBuilder))
{
    var command =
        CslCommandGenerator.GenerateTableMappingCreateCommand(
            Data.Ingestion.IngestionMappingKind.Csv,
            table,
            tableMapping,
            new[] {
                new ColumnMapping() { ColumnName = "StartTime", Properties = new Dictionary<string, string>
() { { MappingConsts.Ordinal, "0" } } },
                new ColumnMapping() { ColumnName = "EndTime", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "1" } } },
                new ColumnMapping() { ColumnName = "EpisodeId", Properties = new Dictionary<string, string>
() { { MappingConsts.Ordinal, "2" } } },
                new ColumnMapping() { ColumnName = "EventId", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "3" } } },
                new ColumnMapping() { ColumnName = "State", Properties =  new Dictionary<string, string>() {
{ MappingConsts.Ordinal, "4" } } },
                new ColumnMapping() { ColumnName = "EventType", Properties =  new Dictionary<string, string>
() { { MappingConsts.Ordinal, "5" } } },
                new ColumnMapping() { ColumnName = "InjuriesDirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "6" } } },
                new ColumnMapping() { ColumnName = "InjuriesIndirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "7" } } },
                new ColumnMapping() { ColumnName = "DeathsDirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "8" } } },
                new ColumnMapping() { ColumnName = "DeathsIndirect", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "9" } } },
                new ColumnMapping() { ColumnName = "DamageProperty", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "10" } } },
                new ColumnMapping() { ColumnName = "DamageCrops", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "11" } } },
                new ColumnMapping() { ColumnName = "Source", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "12" } } },
                new ColumnMapping() { ColumnName = "BeginLocation", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "13" } } },
                new ColumnMapping() { ColumnName = "EndLocation", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "14" } } },
                new ColumnMapping() { ColumnName = "BeginLat", Properties =  new Dictionary<string, string>
() { { MappingConsts.Ordinal, "15" } } },
                new ColumnMapping() { ColumnName = "BeginLon", Properties =  new Dictionary<string, string>
() { { MappingConsts.Ordinal, "16" } } },
                new ColumnMapping() { ColumnName = "EndLat", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "17" } } },
                new ColumnMapping() { ColumnName = "EndLon", Properties =  new Dictionary<string, string>()
{ { MappingConsts.Ordinal, "18" } } },
                new ColumnMapping() { ColumnName = "EpisodeNarrative", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "19" } } },
                new ColumnMapping() { ColumnName = "EventNarrative", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "20" } } },
                new ColumnMapping() { ColumnName = "StormSummary", Properties =  new Dictionary<string,
string>() { { MappingConsts.Ordinal, "21" } } }
        });

    kustoClient.ExecuteControlCommand(databaseName, command);
}
```

Azure Resource Manager template for adding an Event Hub data connection

The following example shows an Azure Resource Manager template for adding an Event Hub data connection. You can edit and deploy the template in the Azure portal by using the form.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "namespaces_eventhubns_name": {
            "type": "string",
            "defaultValue": "eventhubns",
            "metadata": {
                "description": "Specifies the Event Hub Namespace name."
            }
        },
        "EventHubs_eventhubdemo_name": {
            "type": "string",
            "defaultValue": "eventhubdemo",
            "metadata": {
                "description": "Specifies the Event Hub name."
            }
        },
        "consumergroup_default_name": {
            "type": "string",
            "defaultValue": "$Default",
            "metadata": {
                "description": "Specifies the consumer group of the Event Hub."
            }
        },
        "Clusters_kustocluster_name": {
            "type": "string",
            "defaultValue": "kustocluster",
            "metadata": {
                "description": "Specifies the name of the cluster"
            }
        },
        "databases_kustodb_name": {
            "type": "string",
            "defaultValue": "kustodb",
            "metadata": {
                "description": "Specifies the name of the database"
            }
        },
        "tables_kustotable_name": {
            "type": "string",
            "defaultValue": "kustotable",
            "metadata": {
                "description": "Specifies the name of the table"
            }
        },
        "mapping_kustomapping_name": {
            "type": "string",
            "defaultValue": "kustomapping",
            "metadata": {
                "description": "Specifies the name of the mapping rule"
            }
        },
        "dataformat_type": {
            "type": "string",
            "defaultValue": "csv",
            "metadata": {
                "description": "Specifies the data format"
            }
        },
        "dataconnections_kustodc_name": {
            "type": "string",
            "defaultValue": "kustodc",
            "metadata": {
                "description": "Name of the data connection to create"
            }
        },
        "subscriptionId": {
            "type": "string",
```

```
            "defaultValue": "[subscription().subscriptionId]",
            "metadata": {
                "description": "Specifies the subscriptionId of the Event Hub"
            }
        },
        "resourceGroup": {
            "type": "string",
            "defaultValue": "[resourceGroup().name]",
            "metadata": {
                "description": "Specifies the resourceGroup of the Event Hub"
            }
        },
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Location for all resources."
            }
        }
    },
    "variables": {
    },
    "resources": [{
            "type": "Microsoft.Kusto/Clusters/Databases/DataConnections",
            "apiVersion": "2019-09-07",
            "name": "[concat(parameters('Clusters_kustocluster_name'), '/',
parameters('databases_kustodb_name'), '/', parameters('dataconnections_kustodc_name'))]",
            "location": "[parameters('location')]",
            "kind": "EventHub",
            "properties": {
                "eventHubResourceId": "[resourceId(parameters('subscriptionId'),
parameters('resourceGroup'), 'Microsoft.EventHub/namespaces/eventhubs',
parameters('namespaces_eventhubns_name'), parameters('EventHubs_eventhubdemo_name'))]",
                "consumerGroup": "[parameters('consumergroup_default_name')]",
                "tableName": "[parameters('tables_kustotable_name')]",
                "mappingRuleName": "[parameters('mapping_kustomapping_name')]",
                "dataFormat": "[parameters('dataformat_type')]"
            }
        }
    ]
}
```

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

Clean up resources using the Azure portal

Delete the resources in the Azure portal by following the steps in clean up resources.

Clean up resources using PowerShell

If the Cloud Shell is still open, you don't need to copy/run the first line (Read-Host).

```
$projectName = Read-Host -Prompt "Enter the same project name that you used in the last procedure"
$resourceGroupName = "${projectName}rg"

Remove-AzResourceGroup -ResourceGroupName $resourceGroupName

Write-Host "Press [ENTER] to continue ..."
```

Next steps

- Analyze with Data Explorer
- Monitor Data Explorer pools

Event Grid data connection (Preview)

8/19/2022 • 4 minutes to read • Edit Online

Event Grid ingestion is a pipeline that listens to Azure storage, and updates Azure Data Explorer to pull information when subscribed events occur. Data Explorer offers continuous ingestion from Azure Storage (Blob storage and ADLSv2) with Azure Event Grid subscription for blob created or blob renamed notifications and streaming these notifications to Data Explorer via an Event Hub.

The Event Grid ingestion pipeline goes through several steps. You create a target table in Data Explorer into which the data in a particular format will be ingested. Then you create an Event Grid data connection in Data Explorer. The Event Grid data connection needs to know events routing information, such as what table to send the data to and the table mapping. You also specify ingestion properties, which describe the data to be ingested, the target table, and the mapping. You can generate sample data and upload blobs or rename blobs to test your connection. Delete blobs after ingestion. This process can be managed through the Azure portal.

Data format

- See supported formats.
- See supported compressions.
  - The original uncompressed data size should be part of the blob metadata, or else Data Explorer will estimate it. The ingestion uncompressed size limit per file is 4 GB.

> **NOTE**
> Event Grid notification subscription can be set on Azure Storage accounts for `BlobStorage`, `StorageV2`, or Data Lake Storage Gen2.

Ingestion properties

You can specify ingestion properties of the blob ingestion via the blob metadata. You can set the following properties:

PROPERTY	PROPERTY DESCRIPTION
`rawSizeBytes`	Size of the raw (uncompressed) data. For Avro/ORC/Parquet, that is the size before format-specific compression is applied. Provide the original data size by setting this property to the uncompressed data size in bytes.
`kustoTable`	Name of the existing target table. Overrides the `Table` set on the `Data Connection` blade.
`kustoDataFormat`	Data format. Overrides the `Data format` set on the `Data Connection` blade.
`kustoIngestionMappingReference`	Name of the existing ingestion mapping to be used. Overrides the `Column mapping` set on the `Data Connection` blade.

PROPERTY	PROPERTY DESCRIPTION
`kustoIgnoreFirstRecord`	If set to `true`, Kusto ignores the first row of the blob. Use in tabular format data (CSV, TSV, or similar) to ignore headers.
`kustoExtentTags`	String representing tags that will be attached to resulting extent.
`kustoCreationTime`	Overrides $IngestionTime for the blob, formatted as an ISO 8601 string. Use for backfilling.

Events routing

When setting up a blob storage connection to Data Explorer cluster, specify target table properties:

- table name
- data format
- mapping

This setup is the default routing for your data, sometimes referred to as `static routing`. You can also specify target table properties for each blob, using blob metadata. The data will dynamically route, as specified by ingestion properties.

The following example shows you how to set ingestion properties on the blob metadata before uploading it. Blobs are routed to different tables.

For more information, see upload blobs.

```
// Blob is dynamically routed to table `Events`, ingested using `EventsMapping` data mapping
blob = container.GetBlockBlobReference(blobName2);
blob.Metadata.Add("rawSizeBytes", "4096"); // the uncompressed size is 4096 bytes
blob.Metadata.Add("kustoTable", "Events");
blob.Metadata.Add("kustoDataFormat", "json");
blob.Metadata.Add("kustoIngestionMappingReference", "EventsMapping");
blob.UploadFromFile(jsonCompressedLocalFileName);
```

Upload blobs

You can create a blob from a local file, set ingestion properties to the blob metadata, and upload it. For examples, see Ingest blobs into Data Explorer by subscribing to Event Grid notifications

NOTE

- Use `BlockBlob` to generate data. `AppendBlob` is not supported.
- Using Azure Data Lake Gen2 storage SDK requires using `CreateFile` for uploading files and `Flush` at the end with the close parameter set to "true".

> - When the Event Hub endpoint doesn't acknowledge receipt of an event, Azure Event Grid activates a retry mechanism. If this retry delivery fails, Event Grid can deliver the undelivered events to a storage account using a process of *dead-lettering*. For more information, see Event Grid message delivery and retry.

Rename blobs

When using ADLSv2, you can rename a blob to trigger blob ingestion to Data Explorer. For example, see Ingest blobs into Data Explorer by subscribing to Event Grid notifications.

> **NOTE**
>
> - Directory renaming is possible in ADLSv2, but it doesn't trigger *blob renamed* events and ingestion of blobs inside the directory. To ingest blobs following renaming, directly rename the desired blobs.
> - If you defined filters to track specific subjects while creating the data connection.

Delete blobs using storage lifecycle

Data Explorer won't delete the blobs after ingestion. Use Azure Blob storage lifecycle to manage your blob deletion. It's recommended to keep the blobs for three to five days.

Known Event Grid issues

- When using Data Explorer to export the files used for event grid ingestion, note:
  - Event Grid notifications aren't triggered if the connection string provided to the export command or the connection string provided to an external table is a connecting string in ADLS Gen2 format (for example, `abfss://filesystem@accountname.dfs.core.windows.net`) but the storage account isn't enabled for hierarchical namespace.
  - If the account isn't enabled for hierarchical namespace, connection string must use the Blob Storage format (for example, `https://accountname.blob.core.windows.net`). The export works as expected even when using the ADLS Gen2 connection string, but notifications won't be triggered and Event Grid ingestion won't work.

Next steps

- Ingest blobs into Data Explorer by subscribing to Event Grid notifications

Ingest blobs into Azure Synapse Data Explorer by subscribing to Event Grid notifications (Preview)

8/19/2022 • 7 minutes to read • Edit Online

Azure Synapse Data Explorer is a fast and highly scalable data exploration service for log and telemetry data. Azure Synapse Data Explorer offers ingestion (data loading) from Event Hubs, IoT Hubs, and blobs written to blob containers.

In this article, you learn how to ingest blobs from your storage account into Azure Synapse Data Explorer using an Event Grid data connection. You'll create an Event Grid data connection that sets an Azure Event Grid subscription. The Event Grid subscription routes events from your storage account to Data Explorer via an Azure Event Hub. Then you'll see an example of the data flow throughout the system.

For general information about ingesting into Data Explorer from Event Grid, see Connect to Event Grid.

Prerequisites

- An Azure subscription. Create a free Azure account.
- Create a Data Explorer pool using Synapse Studio or the Azure portal
- Create a Data Explorer database.
  1. In Synapse Studio, on the left-side pane, select **Data**.
  2. Select **+** (Add new resource) > **Data Explorer pool**, and use the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Pool name	*contosodataexplorer*	The name of the Data Explorer pool to use
Name	*TestDatabase*	The database name must be unique within the cluster.
Default retention period	*365*	The time span (in days) for which it's guaranteed that the data is kept available to query. The time span is measured from the time that data is ingested.
Default cache period	*31*	The time span (in days) for which to keep frequently queried data available in SSD storage or RAM, rather than in longer-term storage.

  3. Select **Create** to create the database. Creation typically takes less than a minute.
- Create a target table to which Event Hubs will send data
  1. In Synapse Studio, on the left-side pane, select **Develop**.
  2. Under **KQL scripts**, Select **+** (Add new resource) > **KQL script**. On the right-side pane, you can

name your script.

3. In the **Connect to** menu, select *contosodataexplorer*.
4. In the **Use database** menu, select *TestDatabase*.
5. Paste in the following command, and select **Run** to create the table.

```
.create table TestTable (TimeStamp: datetime, Value: string, Source:string)
```

> **TIP**
> Verify that the table was successfully created. On the left-side pane, select **Data**, select the *contosodataexplorer* more menu, and then select **Refresh**. Under *contosodataexplorer*, expand **Tables** and make sure that the *TestTable* table appears in the list.

6. Copy the following command into the window and select **Run** to map the incoming JSON data to the column names and data types of the table (TestTable).

```
.create table TestTable ingestion json mapping 'TestMapping'
'[{"column":"TimeStamp","path":"$.TimeStamp"},{"column":"Value","path":"$.Value"},
{"column":"Source","path":"$.Source"}]'
```

- Create a storage account.
- Event Grid notification subscription can be set on Azure Storage accounts for `BlobStorage`, `StorageV2`, or Data Lake Storage Gen2.

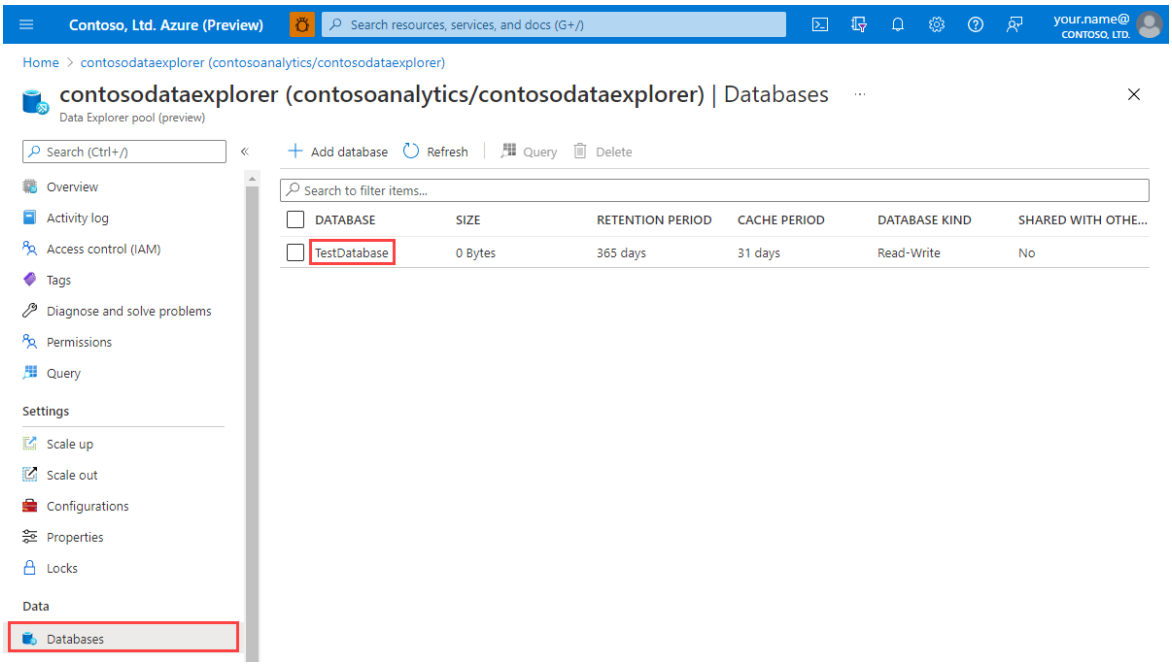
Sign in to the Azure portal

Sign in to the Azure portal.

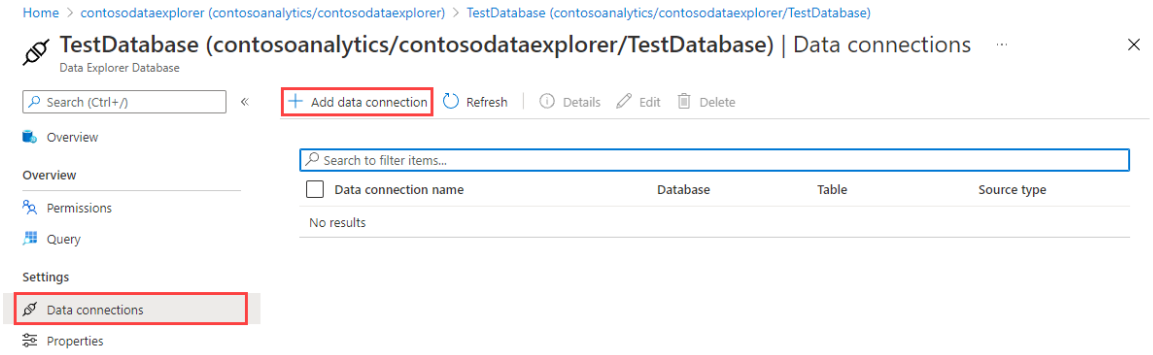
Create an Event Grid data connection

Now connect the storage account to Data Explorer, so that data flowing into the storage is streamed to the test table. This connection can be created in the Azure portal under Data Explorer.

1. Under the Data Explorer pool you created, select **Databases** > **TestDatabase**.



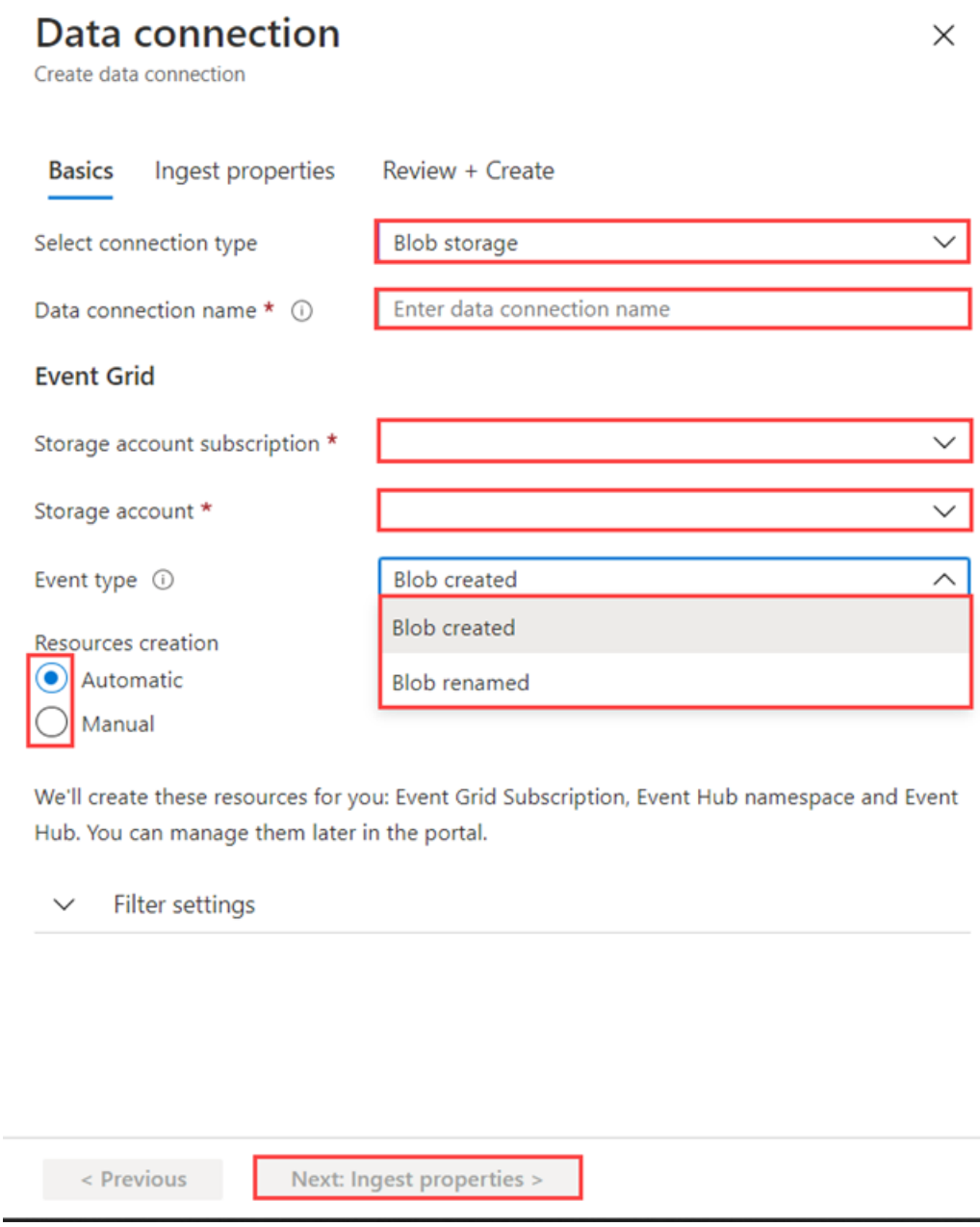
2. Select **Data connections** and **Add data connection**.



Data connection - Basics tab

1. Select the connection type: **Blob storage**.

2. Fill out the form with the following information:

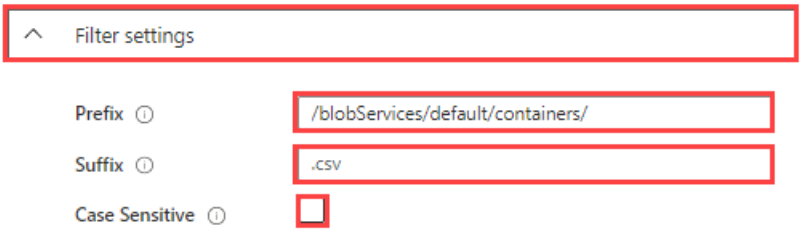


SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Data connection name	*test-grid-connection*	The name of the connection that you want to create in Data Explorer.
Storage account subscription	Your subscription ID	The subscription ID where your storage account is.
Storage account	*gridteststorage1*	The name of the storage account that you created previously.
Event type	*Blob created* or *Blob renamed*	The type of event that triggers ingestion. *Blob renamed* is supported only for ADLSv2 storage. Supported types are: Microsoft.Storage.BlobCreated or Microsoft.Storage.BlobRenamed.

SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Resources creation	*Automatic*	Define whether you want Data Explorer to create an Event Grid Subscription, an Event Hub namespace, and an Event Hub for you.

3. Select **Filter settings** if you want to track specific subjects. Set the filters for the notifications as follows:

   - **Prefix** field is the *literal* prefix of the subject. As the pattern applied is *startswith*, it can span multiple containers, folders, or blobs. No wildcards are allowed.
     - To define a filter on the blob container, the field *must* be set as follows: `/blobServices/default/containers/[container prefix]`.
     - To define a filter on a blob prefix (or a folder in Azure Data Lake Gen2), the field *must* be set as follows: `/blobServices/default/containers/[container name]/blobs/[folder/blob prefix]`.
   - **Suffix** field is the *literal* suffix of the blob. No wildcards are allowed.
   - **Case-Sensitive** field indicates whether the prefix and suffix filters are case-sensitive
   - For more information about filtering events, see Blob storage events.

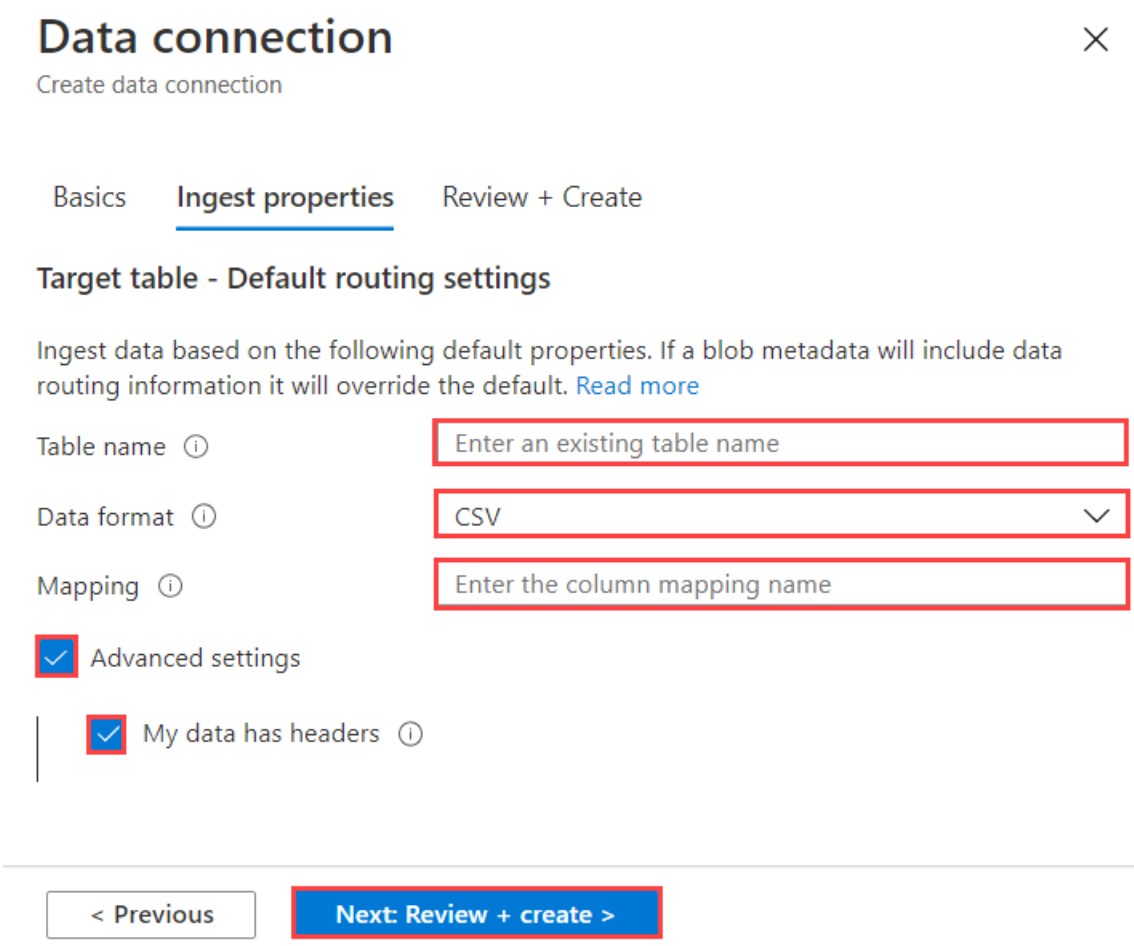


4. Select **Next: Ingest properties**.

> **NOTE**
> We recommend updating the data connection to use managed identities to access the storage account as soon as soon as the option becomes available for your cluster.

Data connection - Ingest properties tab

1. Fill out the form with the following information. Table and mapping names are case-sensitive:



Ingest properties:

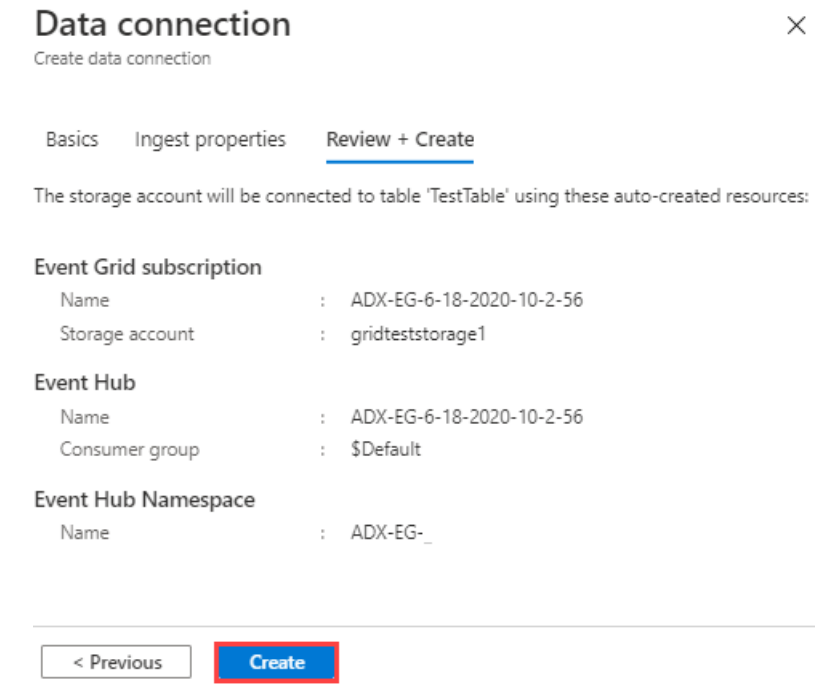
SETTING	SUGGESTED VALUE	FIELD DESCRIPTION
Table name	*TestTable*	The table you created in **TestDatabase**.
Data format	*JSON*	Supported formats are Avro, CSV, JSON, MULTILINE JSON, ORC, PARQUET, PSV, SCSV, SOHSV, TSV, TXT, TSVE, APACHEAVRO, RAW, and W3CLOG. Supported compression options are Zip and Gzip.
Mapping	*TestMapping*	The mapping you created in **TestDatabase**, which maps incoming JSON data to the column names and data types of **TestTable**.
Advanced settings	*My data has headers*	Ignores headers. Supported for *SV type files.

> **NOTE**
> You don't have to specify all **Default routing settings**. Partial settings are also accepted.

2. Select **Next: Review + Create**

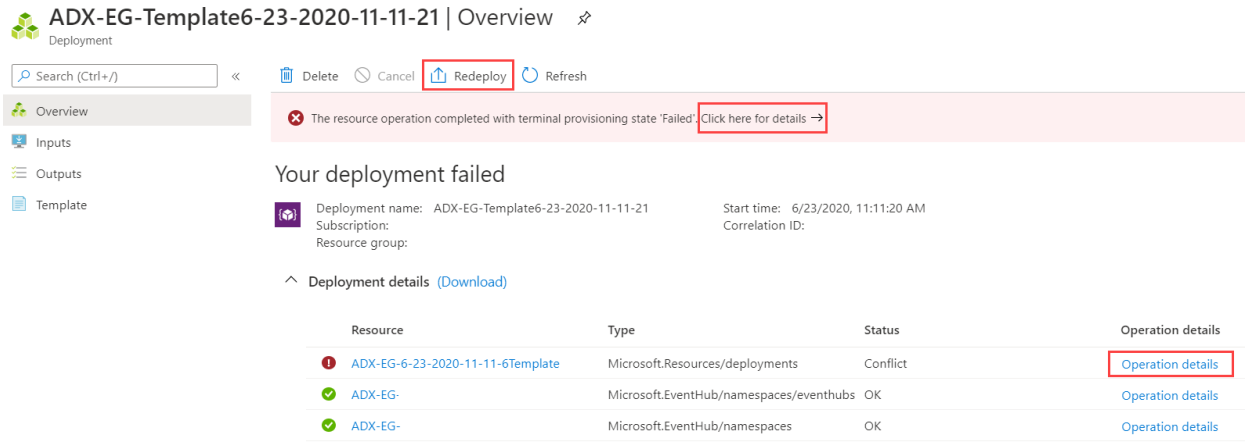
Data connection - Review + Create tab

1. Review the resources that were auto created for you and select **Create**.



Deployment

Wait until the deployment is completed. If your deployment failed, select **Operation details** next to the failed stage to get more information for the failure reason. Select **Redeploy** to try to deploy the resources again. You can alter the parameters before deployment.



Generate sample data

Now that Data Explorer and the storage account are connected, you can create sample data.

Upload blob to the storage container

We'll work with a small shell script that issues a few basic Azure CLI commands to interact with Azure Storage resources. This script does the following actions:

1. Creates a new container in your storage account.
2. Uploads an existing file (as a blob) to that container.
3. Lists the blobs in the container.

You can use Azure Cloud Shell to execute the script directly in the portal.

Save the data into a file and upload it with this script:

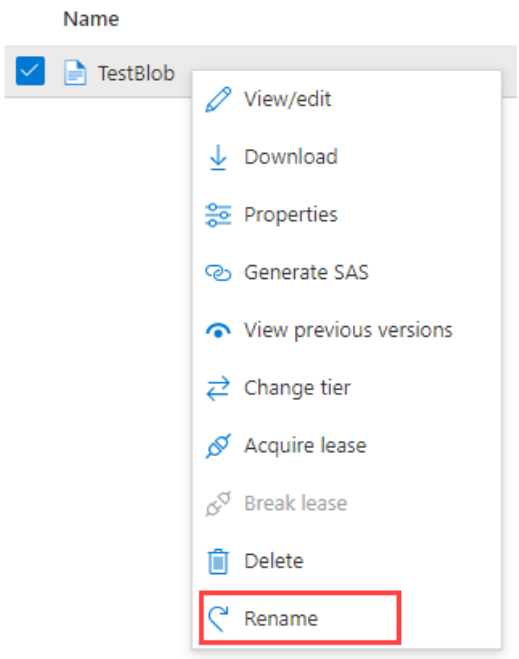
```
{"TimeStamp": "1987-11-16 12:00","Value": "Hello World","Source": "TestSource"}
```

```
    #!/bin/bash
    ### A simple Azure Storage example script

    export AZURE_STORAGE_ACCOUNT=<storage_account_name>
    export AZURE_STORAGE_KEY=<storage_account_key>

    export container_name=<container_name>
    export blob_name=<blob_name>
    export file_to_upload=<file_to_upload>
    export destination_file=<destination_file>

    echo "Creating the container..."
    az storage container create --name $container_name

    echo "Uploading the file..."
    az storage blob upload --container-name $container_name --file $file_to_upload --name $blob_name --
metadata "rawSizeBytes=1024"

    echo "Listing the blobs..."
    az storage blob list --container-name $container_name --output table

    echo "Done"
```

> **NOTE**
>
> To achieve the best ingestion performance, the *uncompressed* size of the compressed blobs submitted for ingestion must be communicated. Because Event Grid notifications contain only basic details, the size information must be explicitly communicated. The uncompressed size information can be provided by setting the `rawSizeBytes` property on the blob metadata with the *uncompressed* data size in bytes.

Rename blob

If you're ingesting data from ADLSv2 storage and have defined *Blob renamed* as the event type for the data connection, the trigger for blob ingestion is blob renaming. To rename a blob, navigate to the blob in Azure portal, right-click on the blob and select **Rename**:



Ingestion properties

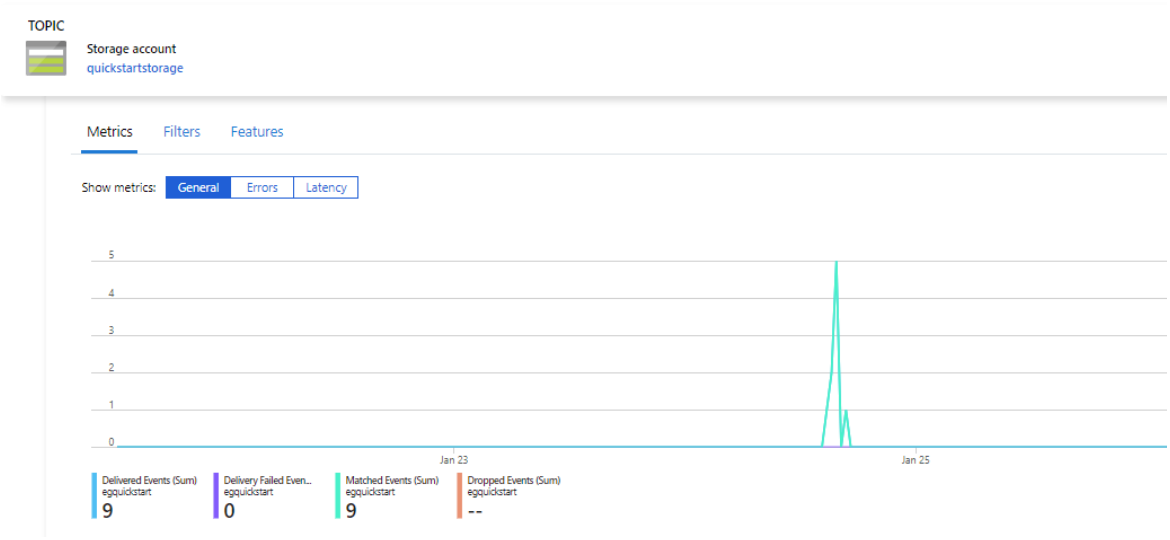
You can specify the ingestion properties of the blob ingestion via the blob metadata.

> **NOTE**
>
> Data Explorer won't delete the blobs post ingestion. Retain the blobs for three to five days. Use Azure Blob storage lifecycle to manage blob deletion.

Review the data flow

> **NOTE**
>
> Data Explorer has an aggregation (batching) policy for data ingestion designed to optimize the ingestion process. By default, the policy is configured to 5 minutes. You'll be able to alter the policy at a later time if needed. In this article you can expect a latency of a few minutes.

1. In the Azure portal, under your Event Grid, you see the spike in activity while the app is running.



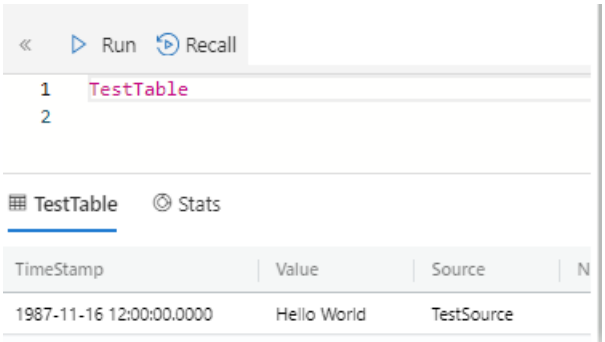
2. To check how many messages have made it to the database so far, run the following query in your test database.

```
TestTable
| count
```

3. To see the content of the messages, run the following query in your test database.

```
TestTable
```

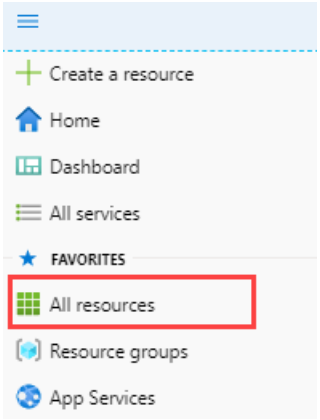
   The result set should look like the following image:



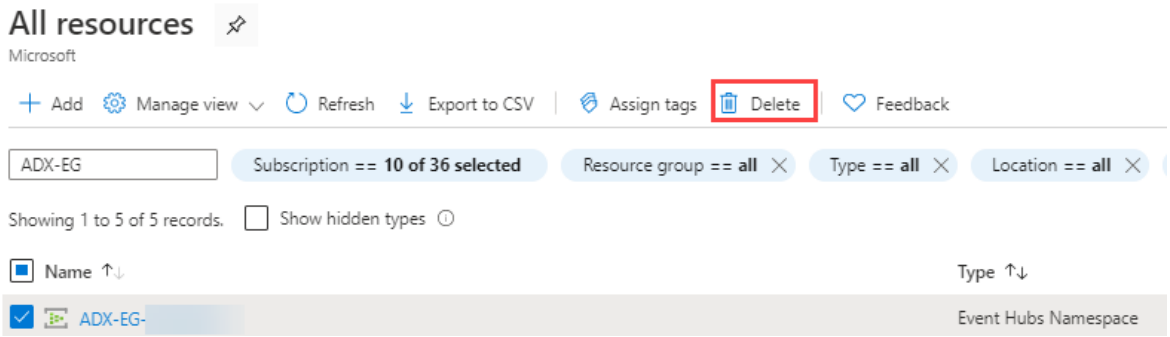
Clean up resources

If you don't plan to use your Event Grid again, clean up the Event Grid Subscription, Event Hub namespace, and Event Hub that were autocreated for you, to avoid incurring costs.

1. In Azure portal, go to the left menu and select **All resources**.

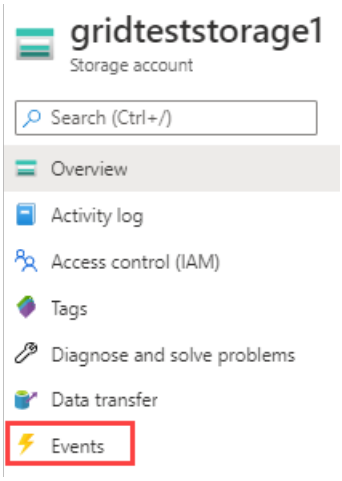


2. Search for your Event Hub Namespace and select **Delete** to delete it:

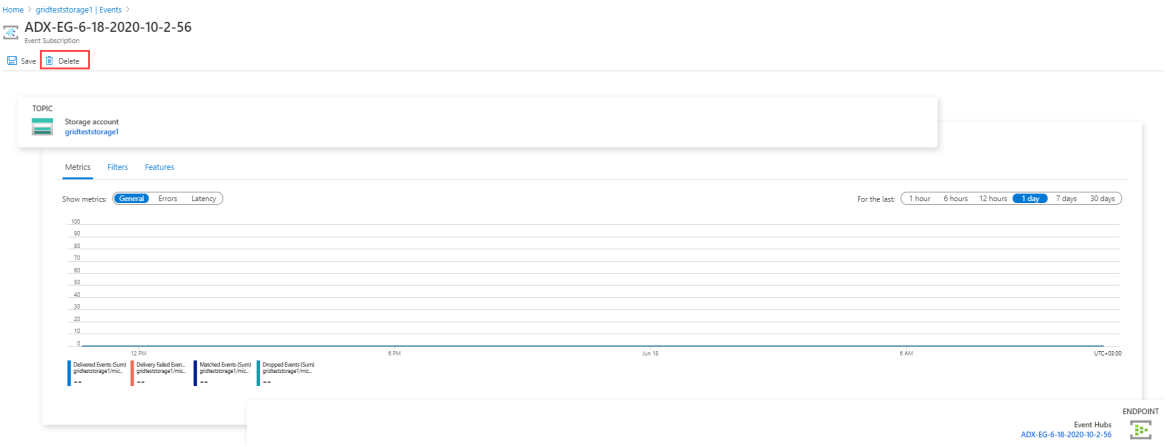


3. In the Delete resources form, confirm the deletion to delete the Event Hub Namespace and Event Hub resources.

4. Go to your storage account. In the left menu, select **Events**:

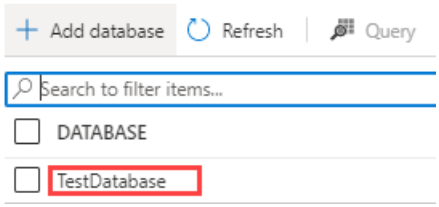


5. Below the graph, Select your Event Grid Subscription and then select **Delete** to delete it:

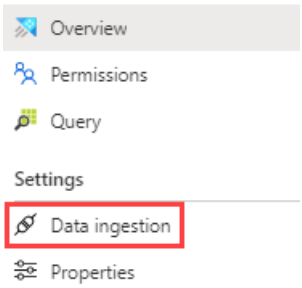


6. To delete your Event Grid data connection, go to your Data Explorer cluster. On the left menu, select **Databases**.

7. Select your database **TestDatabase**:



8. On the left menu, select **Data ingestion**:



9. Select your data connection *test-grid-connection* and then select **Delete** to delete it.

Next steps

- Analyze with Data Explorer
- Monitor Data Explorer pools

Quickstart: Ingest data using Azure Synapse Pipelines (Preview)

8/19/2022 • 5 minutes to read • Edit Online

In this quickstart, you learn how to load data from a data source into Azure Synapse Data Explorer pool.

Prerequisites

- An Azure subscription. Create a free Azure account.
- Create a Data Explorer pool using Synapse Studio or the Azure portal
- Create a Data Explorer database.
  1. In Synapse Studio, on the left-side pane, select **Data**.
  2. Select **+** (Add new resource) > **Data Explorer pool**, and use the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Pool name	*contosodataexplorer*	The name of the Data Explorer pool to use
Name	*TestDatabase*	The database name must be unique within the cluster.
Default retention period	*365*	The time span (in days) for which it's guaranteed that the data is kept available to query. The time span is measured from the time that data is ingested.
Default cache period	*31*	The time span (in days) for which to keep frequently queried data available in SSD storage or RAM, rather than in longer-term storage.

  3. Select **Create** to create the database. Creation typically takes less than a minute.
- Create a table
  1. In Synapse Studio, on the left-side pane, select **Develop**.
  2. Under **KQL scripts**, Select **+** (Add new resource) > **KQL script**. On the right-side pane, you can name your script.
  3. In the **Connect to** menu, select *contosodataexplorer*.
  4. In the **Use database** menu, select *TestDatabase*.
  5. Paste in the following command, and select **Run** to create the table.

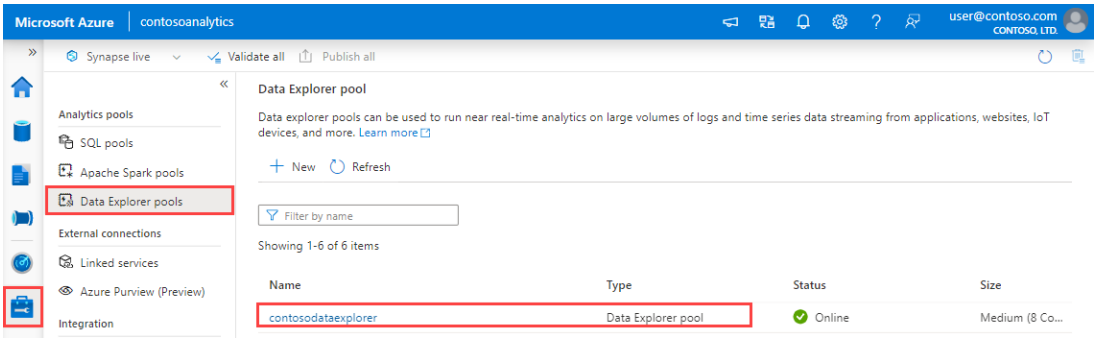
```
.create table StormEvents (StartTime: datetime, EndTime: datetime, EpisodeId: int, EventId: int,
State: string, EventType: string, InjuriesDirect: int, InjuriesIndirect: int, DeathsDirect: int,
DeathsIndirect: int, DamageProperty: int, DamageCrops: int, Source: string, BeginLocation: string,
EndLocation: string, BeginLat: real, BeginLon: real, EndLat: real, EndLon: real, EpisodeNarrative:
string, EventNarrative: string, StormSummary: dynamic)
```

TIP

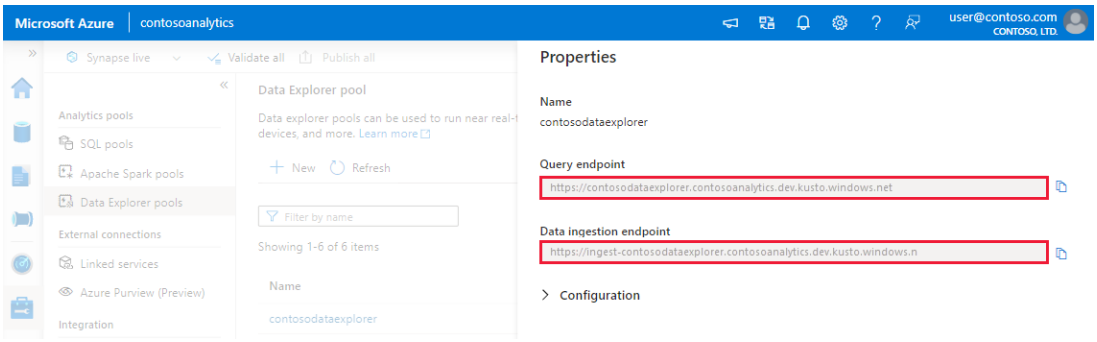
Verify that the table was successfully created. On the left-side pane, select **Data**, select the *contosodataexplorer* more menu, and then select **Refresh**. Under *contosodataexplorer*, expand **Tables** and make sure that the *StormEvents* table appears in the list.

- Get the Query and Data Ingestion endpoints. You'll need the query endpoint to configure your linked service.

  1. In Synapse Studio, on the left-side pane, select **Manage** > **Data Explorer pools**.
  2. Select the Data Explorer pool you want to use to view its details.



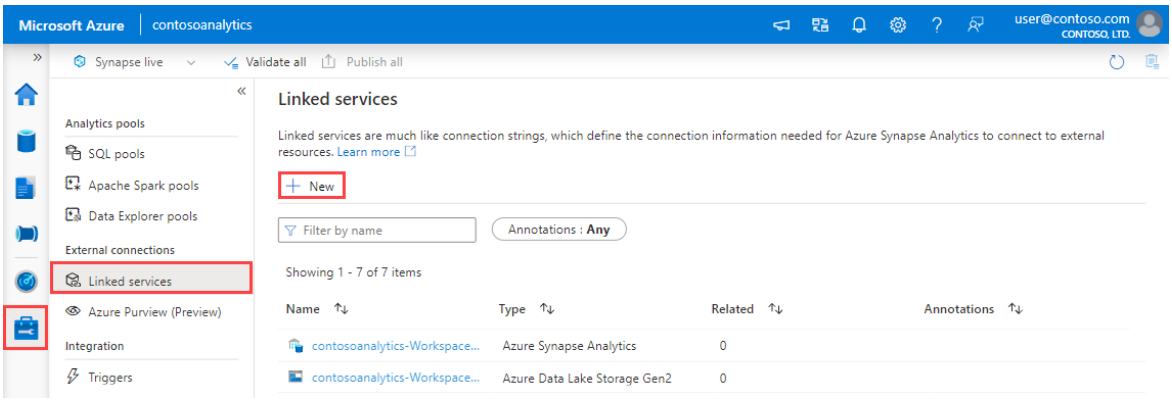
  3. Make a note of the Query and Data Ingestion endpoints. Use the Query endpoint as the cluster when configuring connections to your Data Explorer pool. When configuring SDKs for data ingestion, use the data ingestion endpoint.



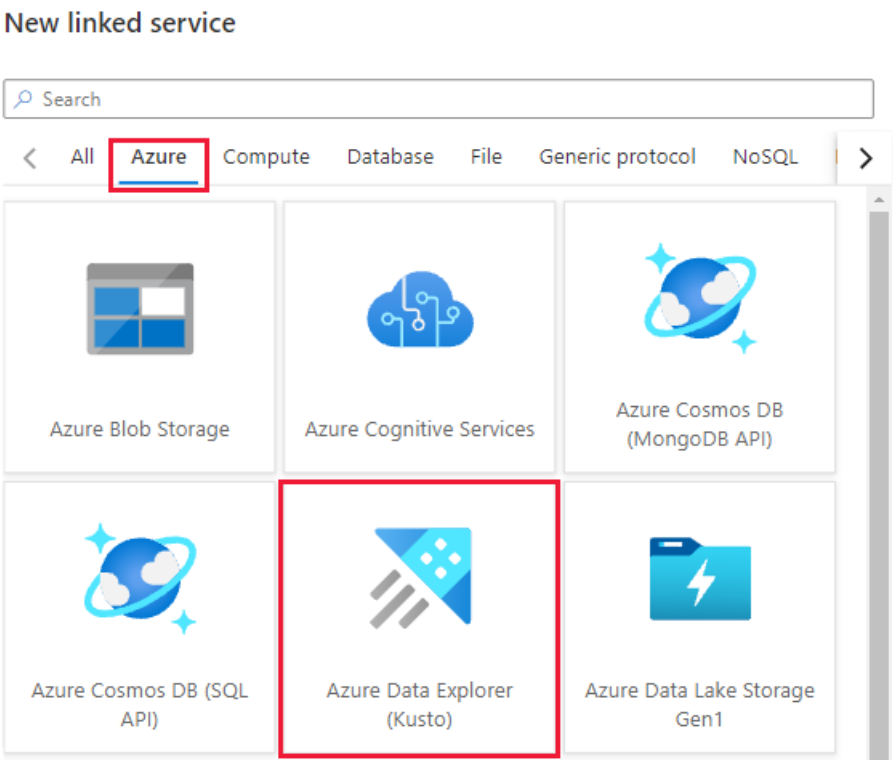
Create a linked service

In Azure Synapse Analytics, a linked service is where you define your connection information to other services. In this section, you'll create a linked service for Azure Data Explorer.

1. In Synapse Studio, on the left-side pane, select **Manage** > **Linked services**.
2. Select **+ New**.

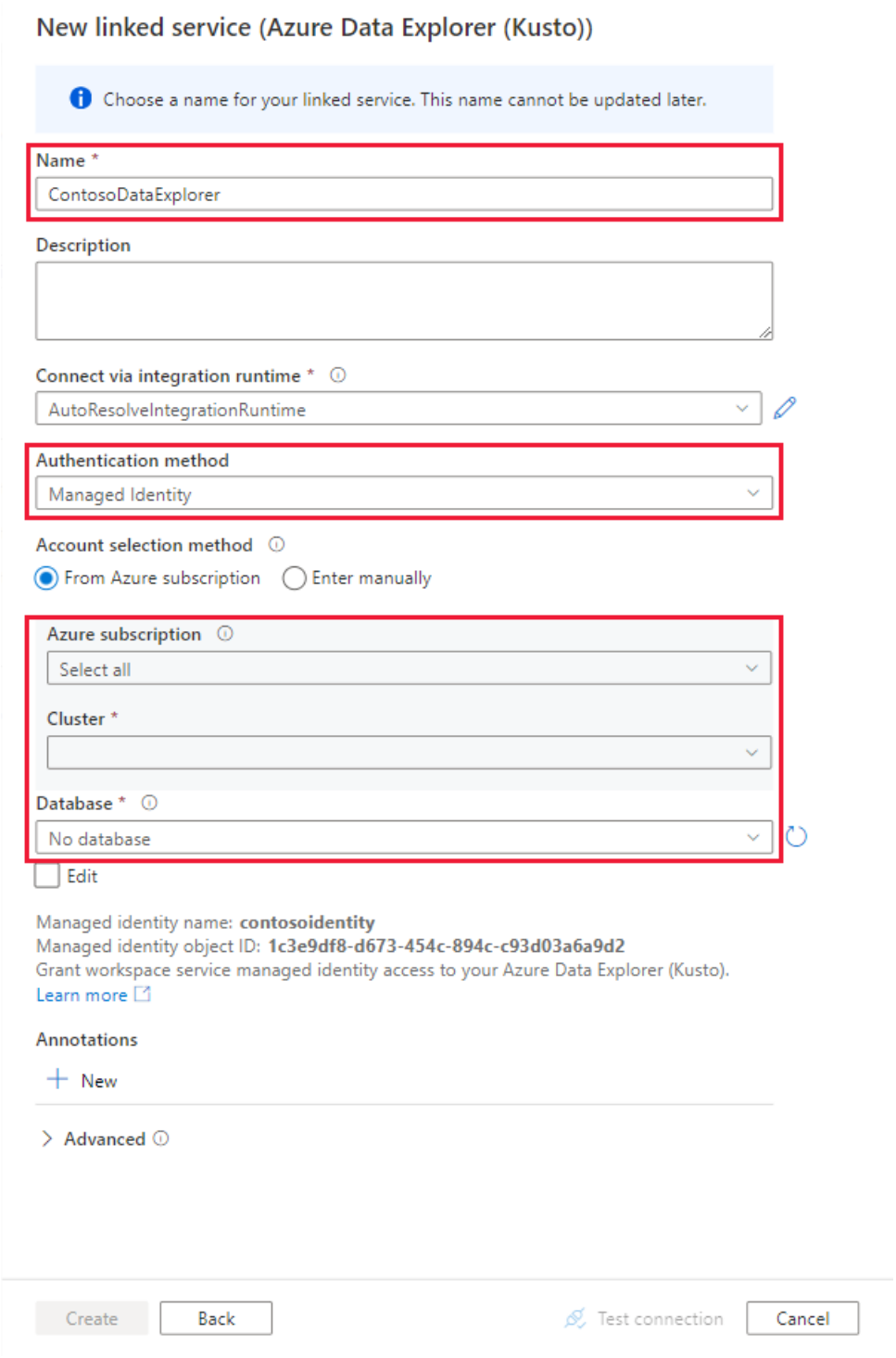


3. Select the **Azure Data Explorer** service from the gallery, and then select **Continue**.



4. In the New Linked Services page, use the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	*contosodataexplorerlinkedservice*	The name for the new Azure Data Explorer linked service.
Authentication method	*Managed Identity*	The authentication method for the new service.
Account selection method	*Enter manually*	The method for specifying the Query endpoint.
Endpoint	*https://contosodataexplorer.contosoanalytics.dev.kusto.windows.net*	The Query endpoint you made a note of earlier.
Database	*TestDatabase*	The database where you want to ingest data.



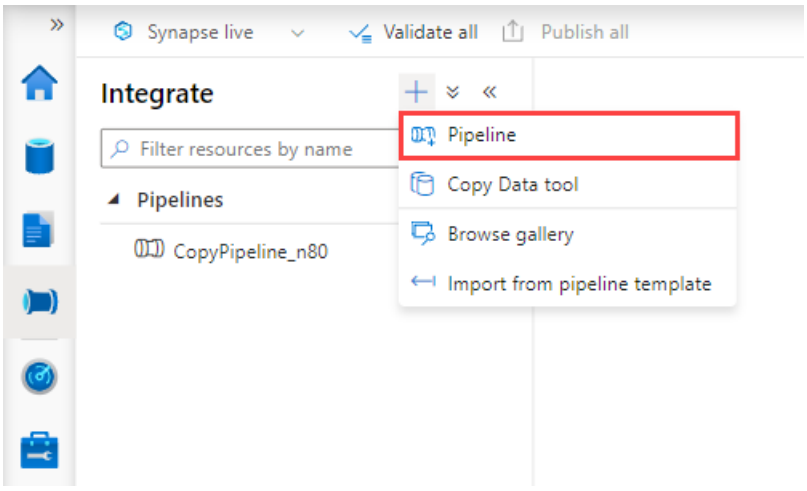
5. Select **Test connection** to validate the settings, and then select **Create**.

Create a pipeline to ingest data

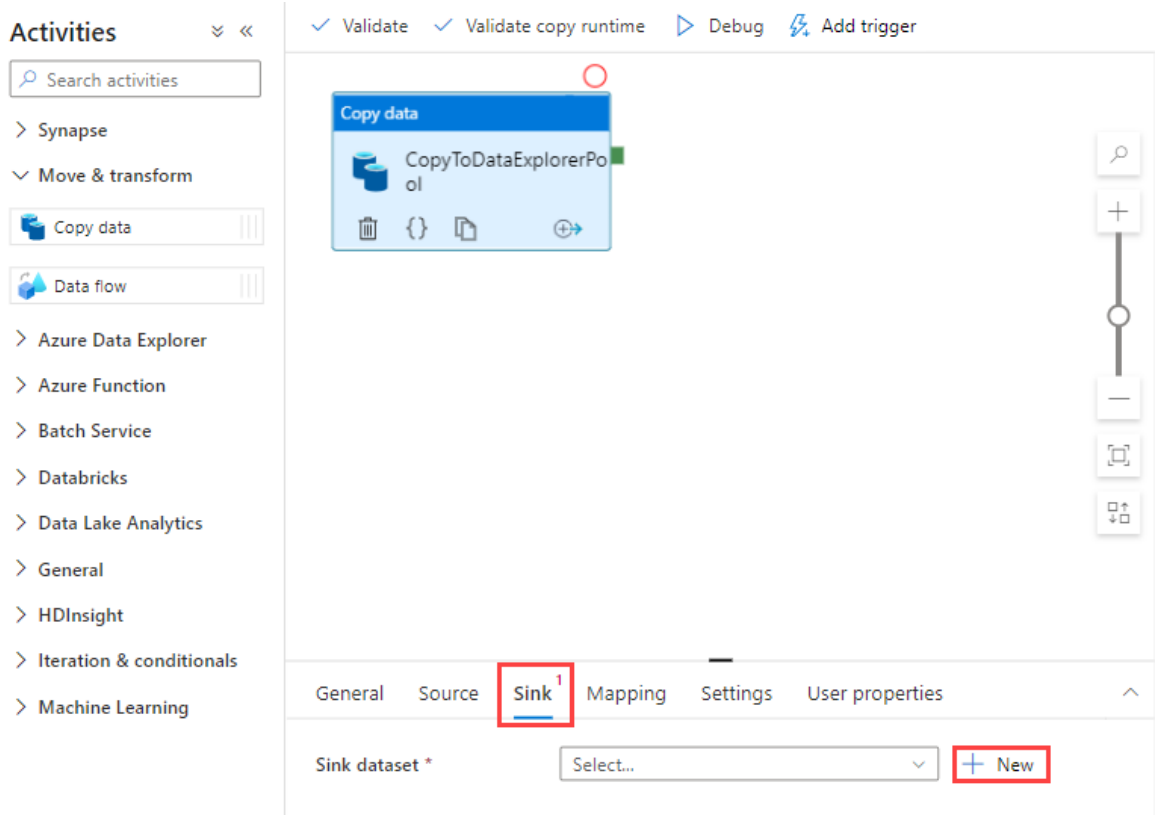
A pipeline contains the logical flow for an execution of a set of activities. In this section, you'll create a pipeline containing a copy activity that ingests data from your preferred source into a Data Explorer pool.

1. In Synapse Studio, on the left-side pane, select **Integrate**.

2. Select **+** > **Pipeline**. On the right-side pane, you can name your pipeline.

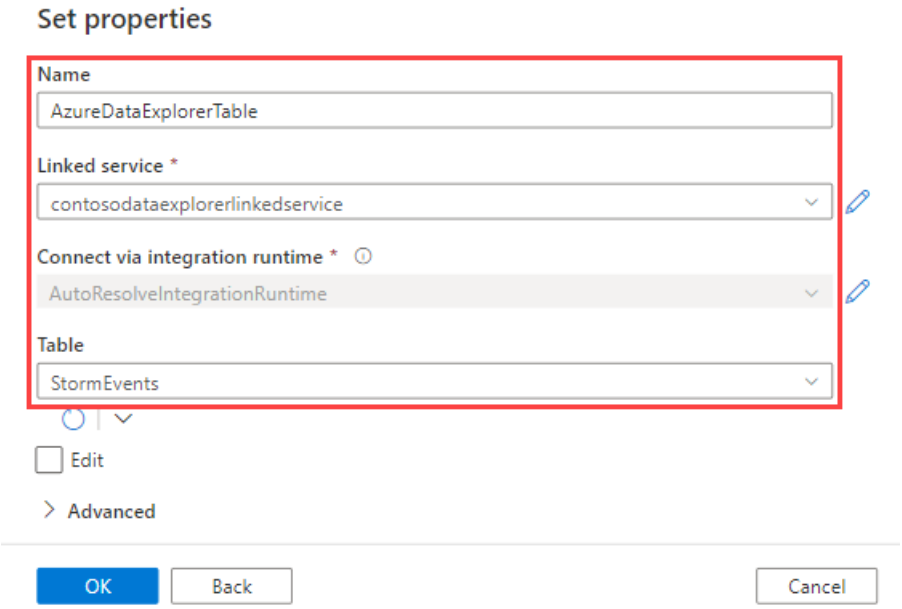


3. Under **Activities** > **Move & transform**, drag **Copy data** onto the pipeline canvas.
4. Select the copy activity and go to the **Source** tab. Select or create a new source dataset as the source to copy data from.
5. Go to the **Sink** tab. Select **New** to create a new sink dataset.



6. Select the **Azure Data Explorer** dataset from the gallery, and then select **Continue**.
7. In the **Set properties** pane, use the following information, and then select **OK**.

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	*AzureDataExplorerTable*	The name for the new pipeline.
Linked service	*contosodataexplorerlinkedservice*	The linked service you created earlier.
Table	*StormEvents*	The table you created earlier.



8. To validate the pipeline, select **Validate** on the toolbar. You see the result of the Pipeline validation output on the right side of the page.

Debug and publish the pipeline

Once you've finished configuring your pipeline, you can execute a debug run before you publish your artifacts to verify everything is correct.

1. Select **Debug** on the toolbar. You see the status of the pipeline run in the **Output** tab at the bottom of the window.
2. Once the pipeline run succeeds, in the top toolbar, select **Publish all**. This action publishes entities (datasets and pipelines) you created to the Synapse Analytics service.
3. Wait until you see the **Successfully published** message. To see notification messages, select the bell button in the top-right.

Trigger and monitor the pipeline

In this section, you manually trigger the pipeline published in the previous step.

1. Select **Add Trigger** on the toolbar, and then select **Trigger Now**. On the **Pipeline Run** page, select **OK**.
2. Go to the **Monitor** tab located in the left sidebar. You see a pipeline run that is triggered by a manual trigger.
3. When the pipeline run completes successfully, select the link under the **Pipeline name** column to view activity run details or to rerun the pipeline. In this example, there's only one activity, so you see only one entry in the list.
4. For details about the copy operation, select the **Details** link (eyeglasses icon) under the **Activity name** column. You can monitor details like the volume of data copied from the source to the sink, data throughput, execution steps with corresponding duration, and used configurations.
5. To switch back to the pipeline runs view, select the **All pipeline runs** link at the top. Select **Refresh** to refresh the list.
6. Verify your data is correctly written in the Data Explorer pool.

Next steps

- Analyze with Data Explorer
- Monitor Data Explorer pools

Quickstart: Use Synapse Studio to monitor your Data Explorer pools (Preview)

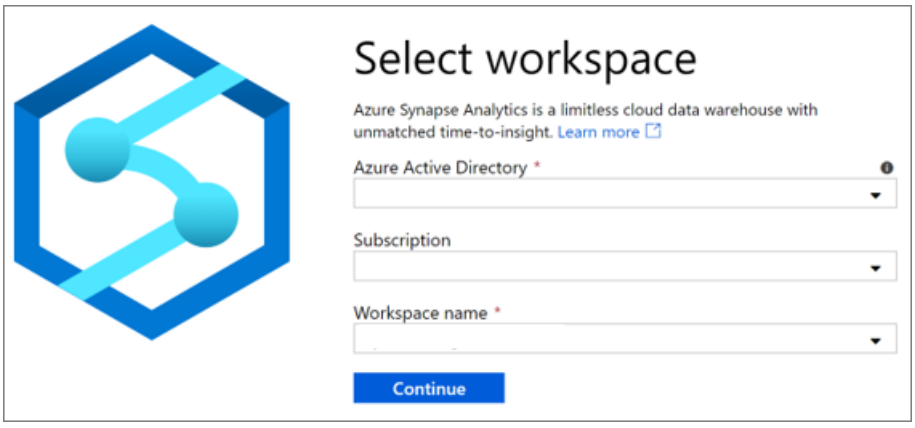
8/19/2022 • 2 minutes to read • Edit Online

With Azure Synapse Analytics, you can use Data Explorer to run queries, dashboards and more on Data Explorer pools in your workspace.

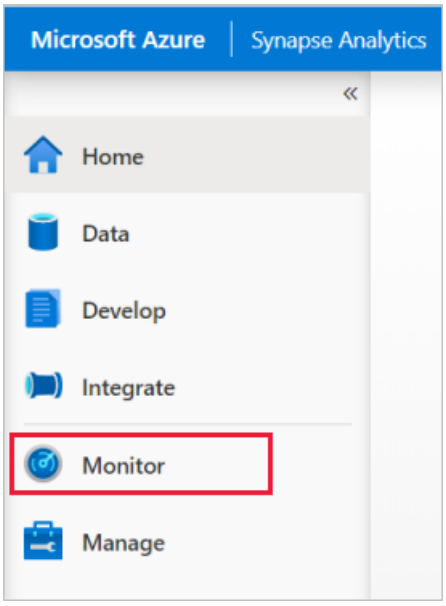
This article explains how to monitor your Data Explorer pools, allowing you to keep an eye on the status of your pools, including how many vCores are in use by different workspace users.

Access Data Explorer pools list

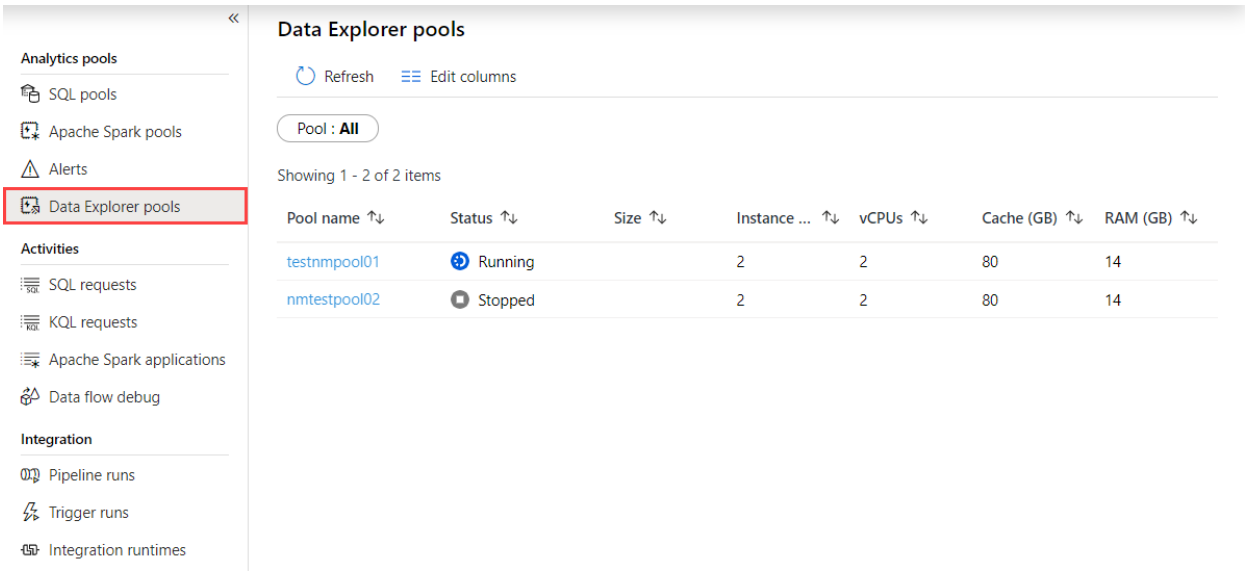
To see the list of Data Explorer pools in your workspace, first open the Synapse Studio and select your workspace.



Once you've opened your workspace, select the **Monitor** section on the left.



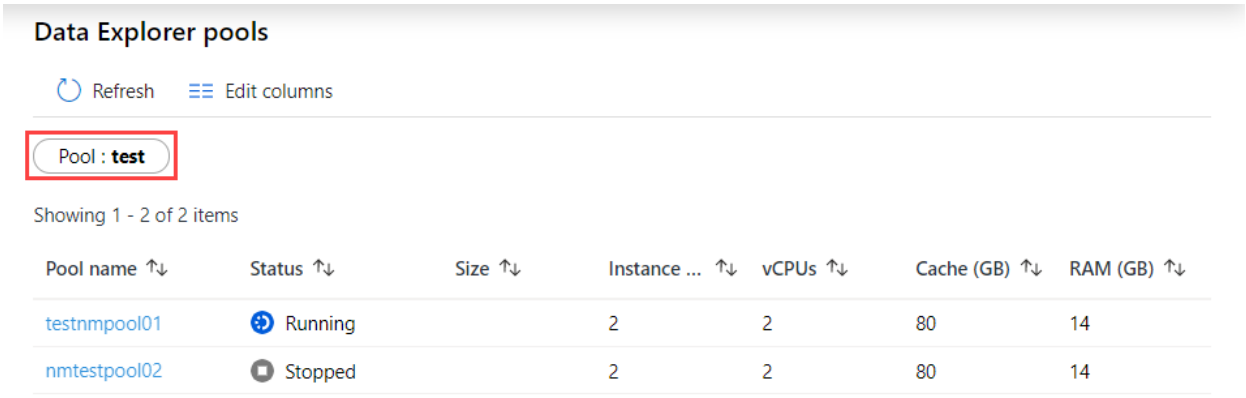
Select **Data Explorer pools** to view the list of Data Explorer pools.



Filter your Data Explorer pools

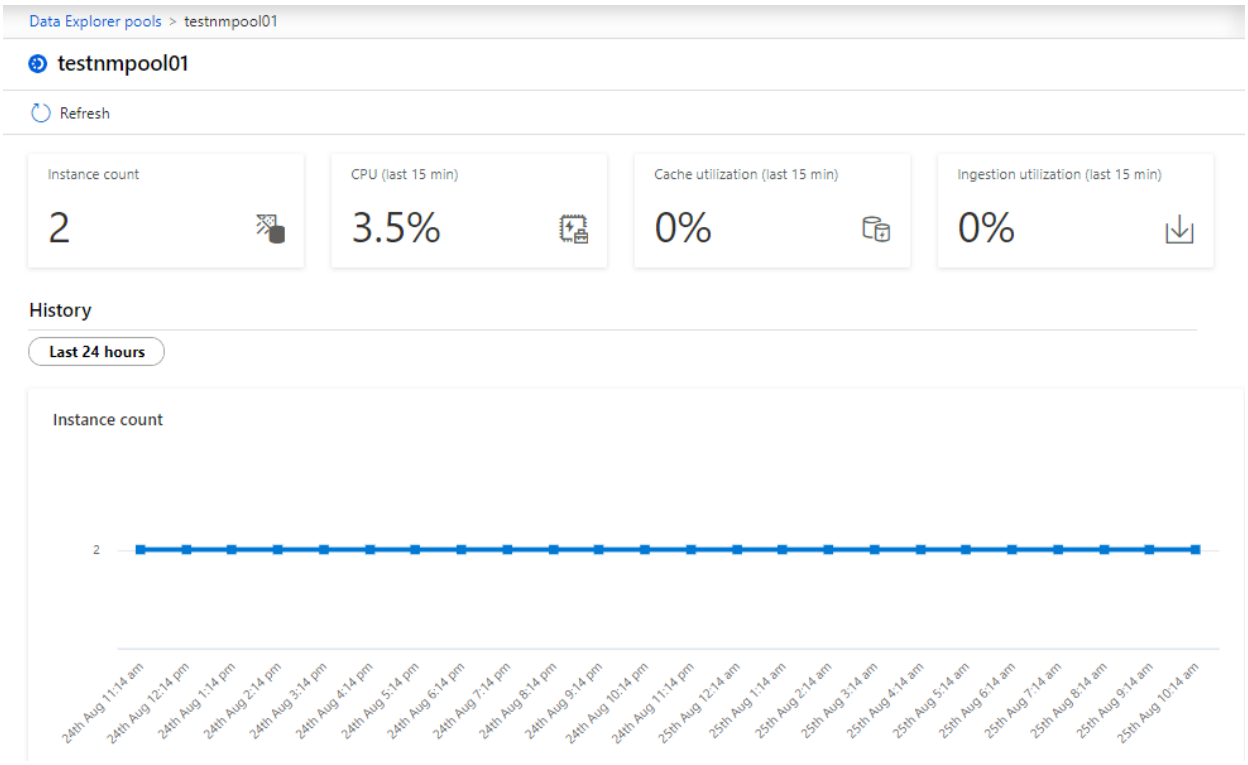
You can filter the list of Data Explorer pools to the ones that interest you. The filters at the top of the screen allow you to specify a field on which you'd like to filter.

For example, you can filter the view to see only the Data Explorer pools containing the name "test":



View details about a specific Data Explorer pool

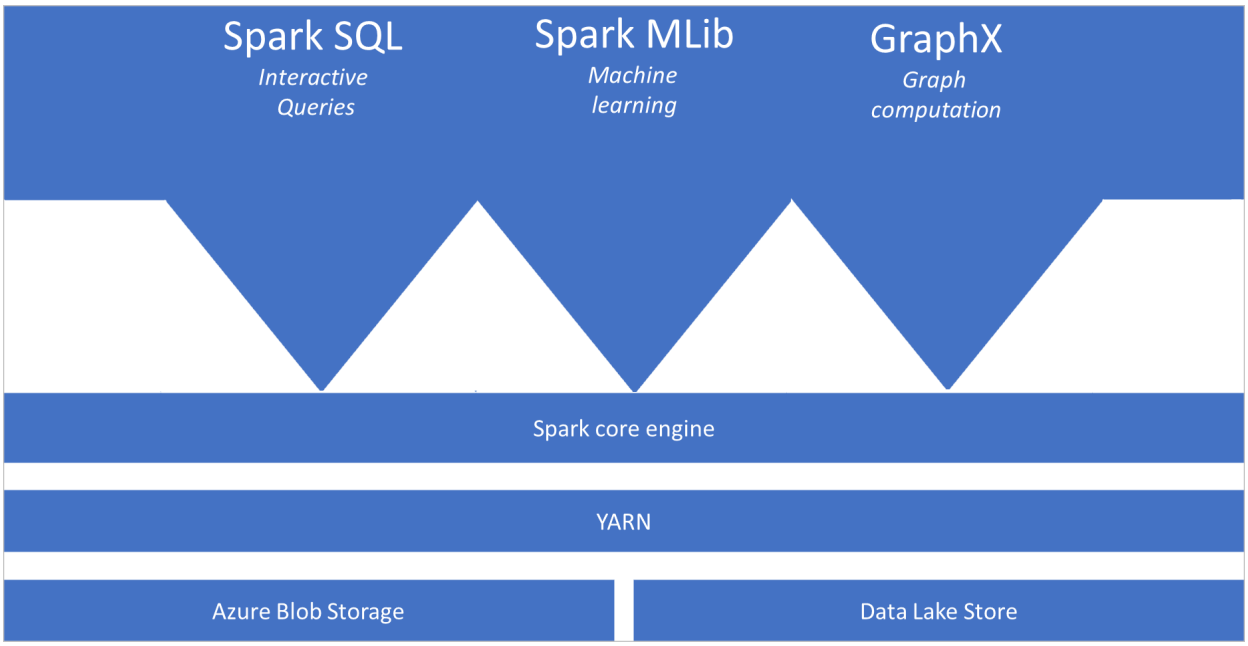
To view the details about one of your Data Explorer pools, select the Data Explorer pool to view the details.


Data Explorer pools > testnmpool01
testnmpool01
Refresh
Instance count
2
CPU (last 15 min)
3.5%
Cache utilization (last 15 min)
0%
Ingestion utilization (last 15 min)
0%
History
Last 24 hours
Instance count
2
24th Aug 11:14 am
24th Aug 12:14 pm
24th Aug 1:14 pm
24th Aug 2:14 pm
24th Aug 3:14 pm
24th Aug 4:14 pm
24th Aug 5:14 pm
24th Aug 6:14 pm
24th Aug 7:14 pm
24th Aug 8:14 pm
24th Aug 9:14 pm
24th Aug 10:14 pm
24th Aug 11:14 pm
25th Aug 12:14 am
25th Aug 1:14 am
25th Aug 2:14 am
25th Aug 3:14 am
25th Aug 4:14 am
25th Aug 5:14 am
25th Aug 6:14 am
25th Aug 7:14 am
25th Aug 8:14 am
25th Aug 9:14 am
25th Aug 10:14 am

Apache Spark in Azure Synapse Analytics

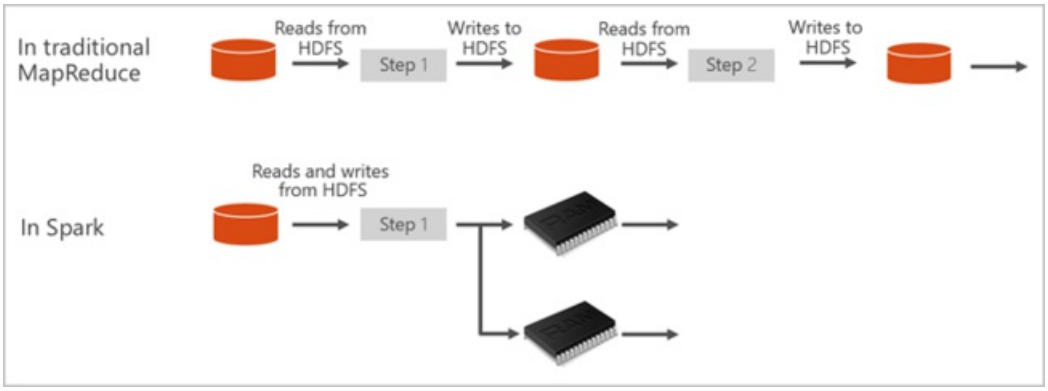
8/19/2022 • 4 minutes to read • Edit Online

Apache Spark is a parallel processing framework that supports in-memory processing to boost the performance of big data analytic applications. Apache Spark in Azure Synapse Analytics is one of Microsoft's implementations of Apache Spark in the cloud. Azure Synapse makes it easy to create and configure a serverless Apache Spark pool in Azure. Spark pools in Azure Synapse are compatible with Azure Storage and Azure Data Lake Generation 2 Storage. So you can use Spark pools to process your data stored in Azure.



What is Apache Spark

Apache Spark provides primitives for in-memory cluster computing. A Spark job can load and cache data into memory and query it repeatedly. In-memory computing is much faster than disk-based applications. Spark also integrates with multiple programming languages to let you manipulate distributed data sets like local collections. There's no need to structure everything as map and reduce operations.



Spark pools in Azure Synapse offer a fully managed Spark service. The benefits of creating a Spark pool in Azure Synapse Analytics are listed here.

FEATURE	DESCRIPTION

FEATURE	DESCRIPTION
Speed and efficiency	Spark instances start in approximately 2 minutes for fewer than 60 nodes and approximately 5 minutes for more than 60 nodes. The instance shuts down, by default, 5 minutes after the last job runs unless it's kept alive by a notebook connection.
Ease of creation	You can create a new Spark pool in Azure Synapse in minutes using the Azure portal, Azure PowerShell, or the Synapse Analytics .NET SDK. See Get started with Spark pools in Azure Synapse Analytics.
Ease of use	Synapse Analytics includes a custom notebook derived from nteract. You can use these notebooks for interactive data processing and visualization.
REST APIs	Spark in Azure Synapse Analytics includes Apache Livy, a REST API-based Spark job server to remotely submit and monitor jobs.
Support for Azure Data Lake Storage Generation 2	Spark pools in Azure Synapse can use Azure Data Lake Storage Generation 2 and BLOB storage. For more information on Data Lake Storage, see Overview of Azure Data Lake Storage.
Integration with third-party IDEs	Azure Synapse provides an IDE plugin for JetBrains' IntelliJ IDEA that is useful to create and submit applications to a Spark pool.
Preloaded Anaconda libraries	Spark pools in Azure Synapse come with Anaconda libraries preinstalled. Anaconda provides close to 200 libraries for machine learning, data analysis, visualization, and other technologies.
Scalability	Apache Spark in Azure Synapse pools can have Auto-Scale enabled, so that pools scale by adding or removing nodes as needed. Also, Spark pools can be shut down with no loss of data since all the data is stored in Azure Storage or Data Lake Storage.

Spark pools in Azure Synapse include the following components that are available on the pools by default:

- Spark Core. Includes Spark Core, Spark SQL, GraphX, and MLlib.
- Anaconda
- Apache Livy
- nteract notebook

Spark pool architecture

Spark applications run as independent sets of processes on a pool, coordinated by the `SparkContext` object in your main program, called the *driver program*.

The `SparkContext` can connect to the cluster manager, which allocates resources across applications. The cluster manager is Apache Hadoop YARN. Once connected, Spark acquires executors on nodes in the pool, which are processes that run computations and store data for your application. Next, it sends your application code, defined by JAR or Python files passed to `SparkContext`, to the executors. Finally, `SparkContext` sends tasks to the

executors to run.

The `SparkContext` runs the user's main function and executes the various parallel operations on the nodes. Then, the `SparkContext` collects the results of the operations. The nodes read and write data from and to the file system. The nodes also cache transformed data in-memory as Resilient Distributed Datasets (RDDs).

The `SparkContext` connects to the Spark pool and is responsible for converting an application to a directed acyclic graph (DAG). The graph consists of individual tasks that run within an executor process on the nodes. Each application gets its own executor processes, which stay up during the whole application and run tasks in multiple threads.

Apache Spark in Azure Synapse Analytics use cases

Spark pools in Azure Synapse Analytics enable the following key scenarios:

- Data Engineering/Data Preparation

  Apache Spark includes language features to support preparation and processing of large volumes of data so that it can be made more valuable and then consumed by other services within Azure Synapse Analytics. This approach is enabled through multiple languages, including C#, Scala, PySpark, and Spark SQL, and supplied libraries for processing and connectivity.

- Machine Learning

  Apache Spark comes with MLlib, a machine learning library built on top of Spark that you can use from a Spark pool in Azure Synapse Analytics. Spark pools in Azure Synapse Analytics also include Anaconda, a Python distribution with various packages for data science including machine learning. When combined with built-in support for notebooks, you have an environment for creating machine learning applications.

Where do I start

Use the following articles to learn more about Apache Spark in Azure Synapse Analytics:

- Quickstart: Create a Spark pool in Azure Synapse
- Quickstart: Create an Apache Spark notebook
- Tutorial: Machine learning using Apache Spark
- Apache Spark official documentation

> **NOTE**
> Some of the official Apache Spark documentation relies on using the Spark console, which is not available on Azure Synapse Spark. Use the notebook or IntelliJ experiences instead.

Next steps

This overview provided a basic understanding of Apache Spark in Azure Synapse Analytics. Advance to the next article to learn how to create a Spark pool in Azure Synapse Analytics:

- Create a Spark pool in Azure Synapse

Quickstart: Create a serverless Apache Spark pool in Azure Synapse Analytics using web tools

8/19/2022 • 5 minutes to read • Edit Online

In this quickstart, you learn how to create a serverless Apache Spark pool in Azure Synapse using web tools. You then learn to connect to the Apache Spark pool and run Spark SQL queries against files and tables. Apache Spark enables fast data analytics and cluster computing using in-memory processing. For information on Spark in Azure Synapse, see Overview: Apache Spark on Azure Synapse.

> **IMPORTANT**
>
> Billing for Spark instances is prorated per minute, whether you are using them or not. Be sure to shutdown your Spark instance after you have finished using it, or set a short timeout. For more information, see the **Clean up resources** section of this article.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure subscription - create one for free
- Synapse Analytics workspace
- Serverless Apache Spark pool

Sign in to the Azure portal

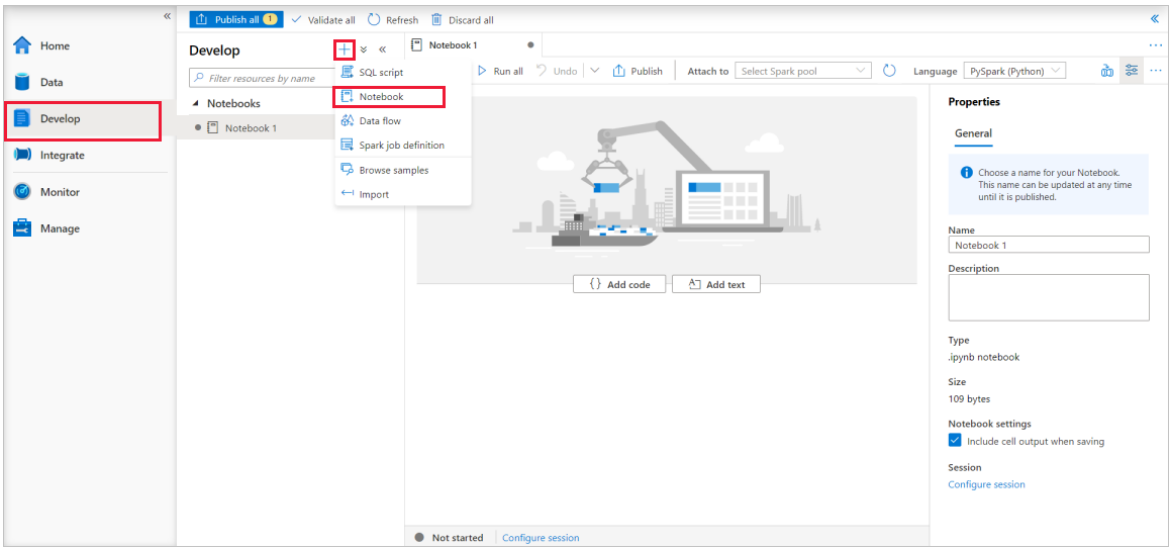
Sign in to the Azure portal.

If you don't have an Azure subscription, create a free account before you begin.

Create a notebook

A notebook is an interactive environment that supports various programming languages. The notebook allows you to interact with your data, combine code with markdown, text, and perform simple visualizations.

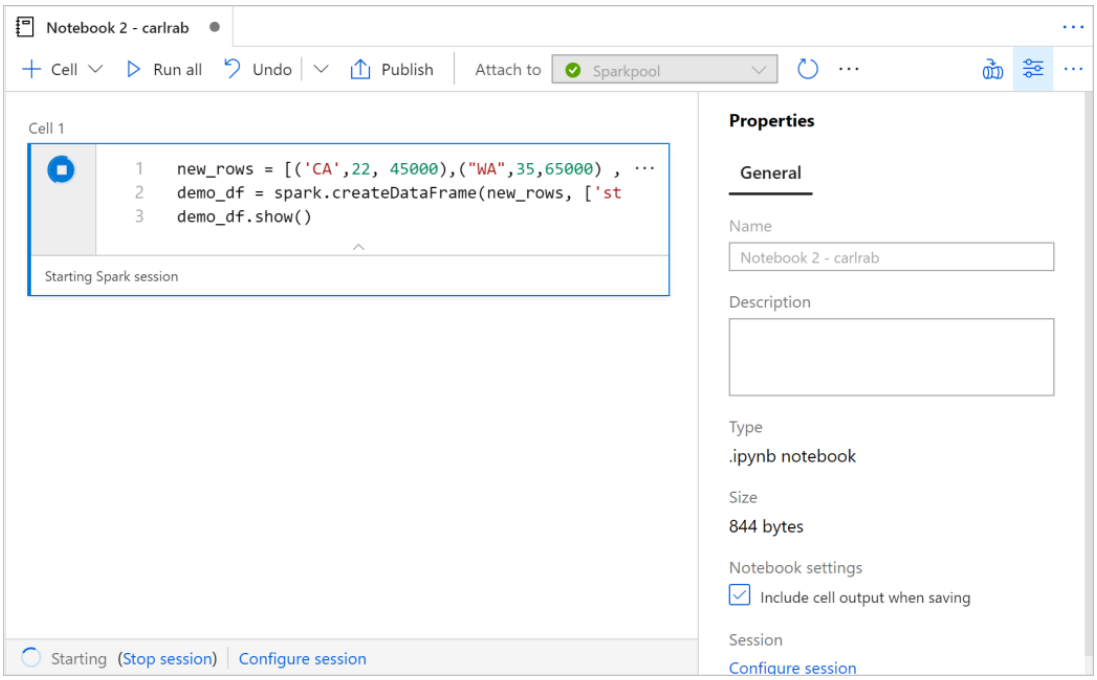
1. From the Azure portal view for the Azure Synapse workspace you want to use, select **Launch Synapse Studio**.
2. Once Synapse Studio has launched, select **Develop**. Then, select the "**+**" icon to add a new resource.
3. From there, select **Notebook**. A new notebook is created and opened with an automatically generated name.



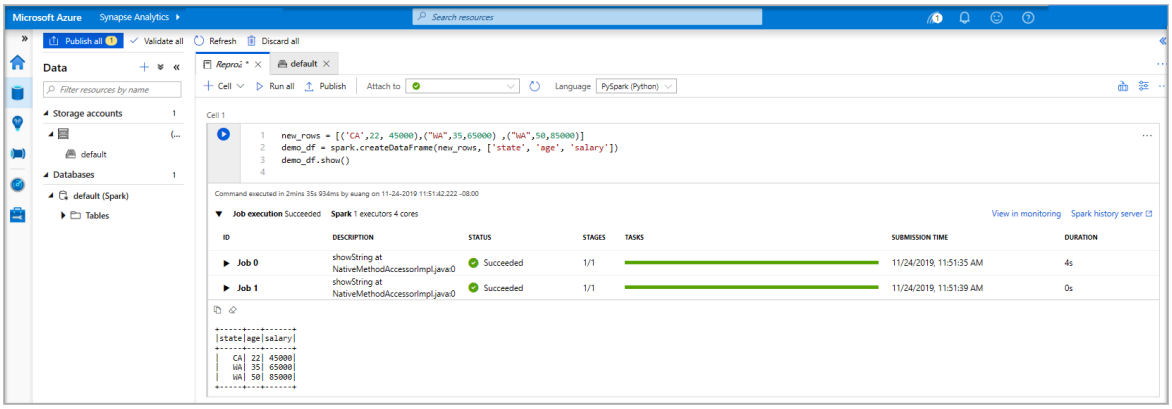
4. In the **Properties** window, provide a name for the notebook.
5. On the toolbar, click **Publish**.
6. If there is only one Apache Spark pool in your workspace, then it's selected by default. Use the drop-down to select the correct Apache Spark pool if none is selected.
7. Click **Add code**. The default language is `Pyspark`. You are going to use a mix of Pyspark and Spark SQL, so the default choice is fine. Other supported languages are Scala and .NET for Spark.
8. Next you create a simple Spark DataFrame object to manipulate. In this case, you create it from code. There are three rows and three columns:

```
new_rows = [('CA',22, 45000),("WA",35,65000) ,("WA",50,85000)]
demo_df = spark.createDataFrame(new_rows, ['state', 'age', 'salary'])
demo_df.show()
```

9. Now run the cell using one of the following methods:
   - Press **SHIFT + ENTER**.
   - Select the blue play icon to the left of the cell.
   - Select the **Run all** button on the toolbar.



10. If the Apache Spark pool instance isn't already running, it is automatically started. You can see the Apache Spark pool instance status below the cell you are running and also on the status panel at the bottom of the notebook. Depending on the size of pool, starting should take 2-5 minutes. Once the code has finished running, information below the cell displays showing how long it took to run and its execution. In the output cell, you see the output.



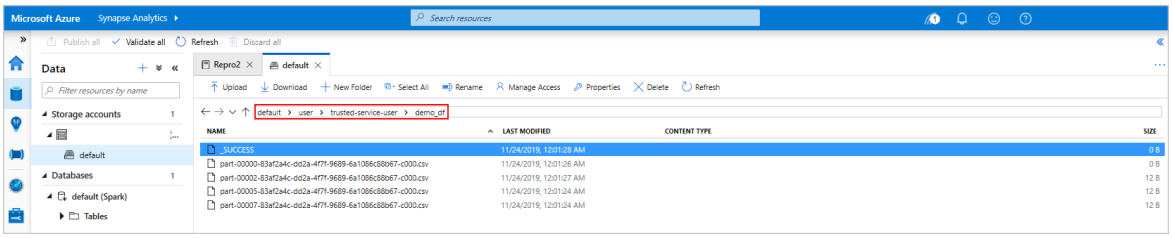
11. The data now exists in a DataFrame from there you can use the data in many different ways. You are going to need it in different formats for the rest of this quickstart.

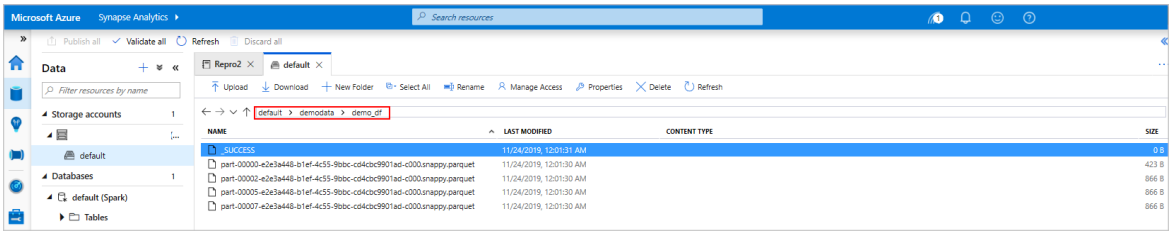
12. Enter the code below in another cell and run it, this creates a Spark table, a CSV, and a Parquet file all with copies of the data:

```
demo_df.createOrReplaceTempView('demo_df')
demo_df.write.csv('demo_df', mode='overwrite')
demo_df.write.parquet('abfss://<<TheNameOfAStorageAccountFileSystem>>@<<TheNameOfAStorageAccount>>.dfs.core.windows.net/demodata/demo_df', mode='overwrite')
```

If you use the storage explorer, it's possible to see the impact of the two different ways of writing a file used above. When no file system is specified then the default is used, in this case `default>user>trusted-service-user>demo_df`. The data is saved to the location of the specified file system.

Notice in both the "csv" and "parquet" formats, write operations a directory is created with many partitioned files.





Run Spark SQL statements

Structured Query Language (SQL) is the most common and widely used language for querying and defining data. Spark SQL functions as an extension to Apache Spark for processing structured data, using the familiar SQL syntax.

1. Paste the following code in an empty cell, and then run the code. The command lists the tables on the pool.

   ```
   %%sql
   SHOW TABLES
   ```

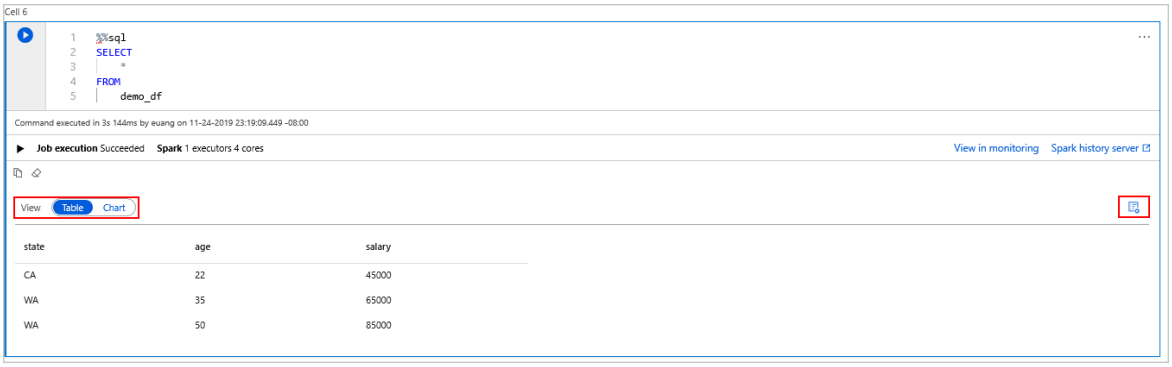
   When you use a Notebook with your Azure Synapse Apache Spark pool, you get a preset `sqlContext` that you can use to run queries using Spark SQL. `%%sql` tells the notebook to use the preset `sqlContext` to run the query. The query retrieves the top 10 rows from a system table that comes with all Azure Synapse Apache Spark pools by default.

2. Run another query to see the data in `demo_df`.

   ```
   %%sql
   SELECT * FROM demo_df
   ```

   The code produces two output cells, one that contains data results the other, which shows the job view.

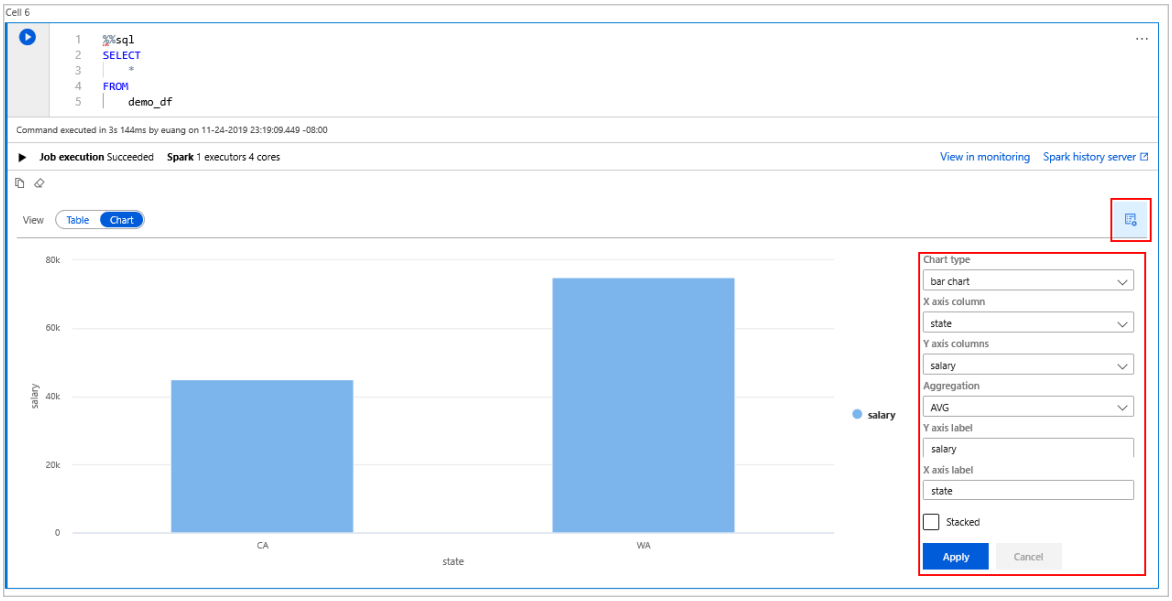
   By default the results view shows a grid. But, there is a view switcher underneath the grid that allows the view to switch between grid and graph views.



3. In the **View** switcher, select **Chart**.

4. Select the **View options** icon from the far right-hand side.

5. In the **Chart type** field, select "bar chart".
6. In the X-axis column field, select "state".
7. In the Y-axis column field, select "salary".
8. In the **Aggregation** field, select to "AVG".
9. Select **Apply**.



10. It is possible to get the same experience of running SQL but without having to switch languages. You can do this by replacing the SQL cell above with this PySpark cell, the output experience is the same because the **display** command is used:

```
display(spark.sql('SELECT * FROM demo_df'))
```

11. Each of the cells that previously executed had the option to go to **History Server** and **Monitoring**. Clicking the links takes you to different parts of the User Experience.

> **NOTE**
> Some of the Apache Spark official documentation relies on using the Spark console, which is not available on Synapse Spark. Use the notebook or IntelliJ experiences instead.

Clean up resources

Azure Synapse saves your data in Azure Data Lake Storage. You can safely let a Spark instance shut down when it is not in use. You are charged for a serverless Apache Spark pool as long as it is running, even when it is not in use.

Since the charges for the pool are many times more than the charges for storage, it makes economic sense to let Spark instances shut down when they are not in use.

To ensure the Spark instance is shut down, end any connected sessions(notebooks). The pool shuts down when the **idle time** specified in the Apache Spark pool is reached. You can also select **end session** from the status bar at the bottom of the notebook.

Next steps

In this quickstart, you learned how to create a serverless Apache Spark pool and run a basic Spark SQL query.

- Azure Synapse Analytics
- .NET for Apache Spark documentation

Quickstart: Create a new serverless Apache Spark pool using the Azure portal

8/19/2022 • 2 minutes to read • Edit Online

Azure Synapse Analytics offers various analytics engines to help you ingest, transform, model, analyze, and distribute your data. An Apache Spark pool provides open-source big data compute capabilities. After you've created an Apache Spark pool in your Synapse workspace, data can be loaded, modeled, processed, and distributed for faster analytic insight.

In this quickstart, you learn how to use the Azure portal to create an Apache Spark pool in a Synapse workspace.

> **IMPORTANT**
> Billing for Spark instances is prorated per minute, whether you are using them or not. Be sure to shutdown your Spark instance after you have finished using it, or set a short timeout. For more information, see the **Clean up resources** section of this article.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

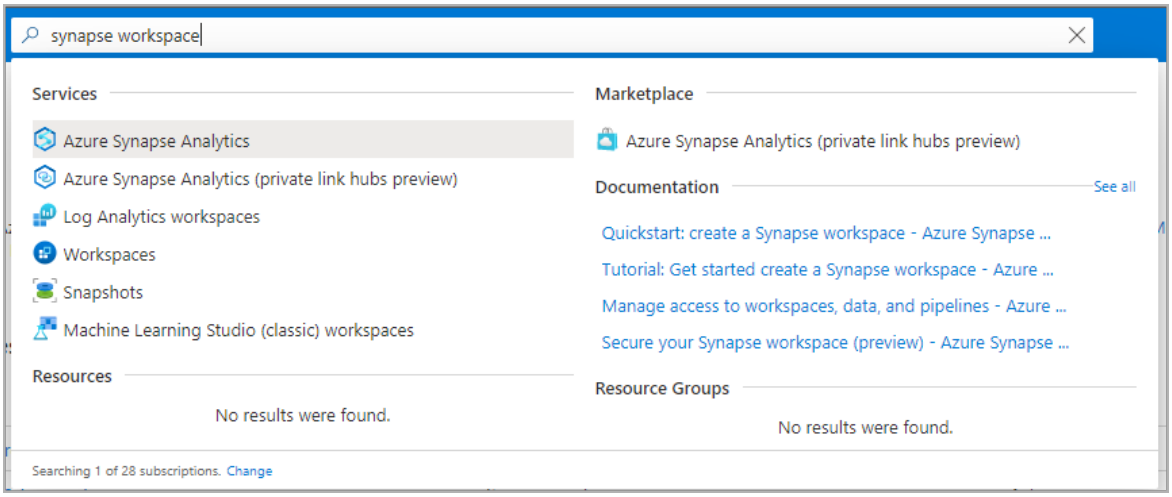
- Azure subscription - create one for free
- Synapse Analytics workspace

Sign in to the Azure portal

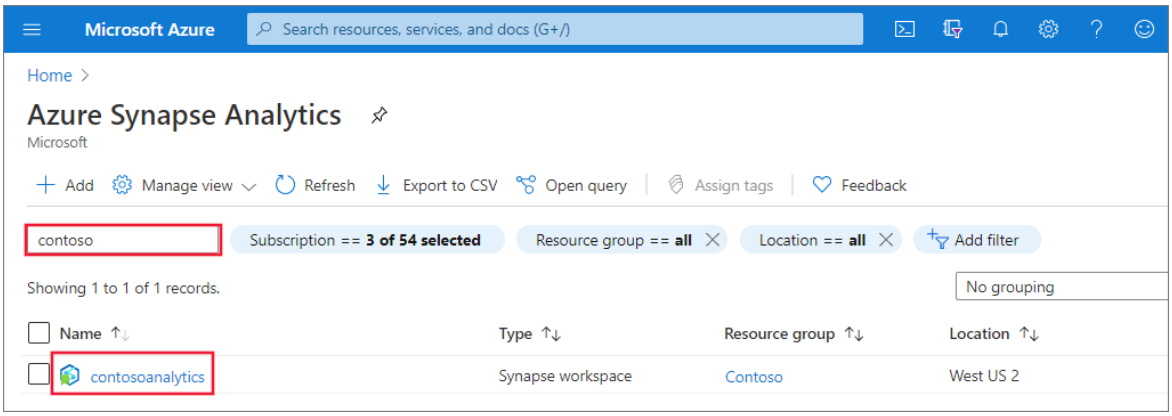
Sign in to the Azure portal

Navigate to the Synapse workspace

1. Navigate to the Synapse workspace where the Apache Spark pool will be created by typing the service name (or resource name directly) into the search bar.

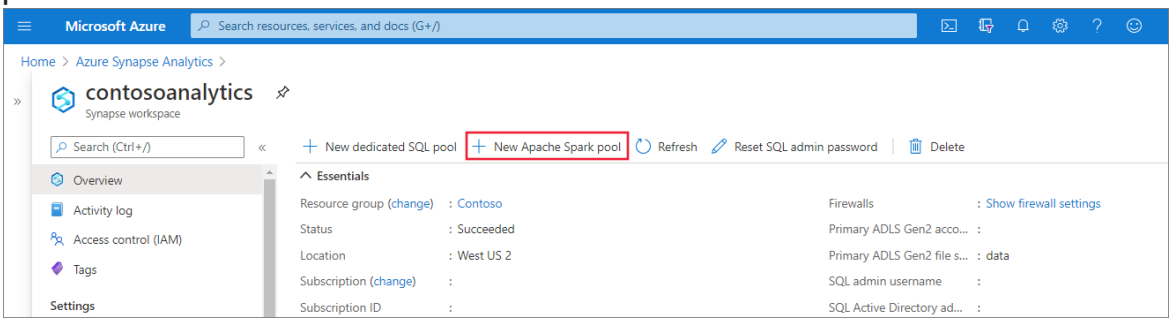


2. From the list of workspaces, type the name (or part of the name) of the workspace to open. For this example, we'll use a workspace named **contosoanalytics**.



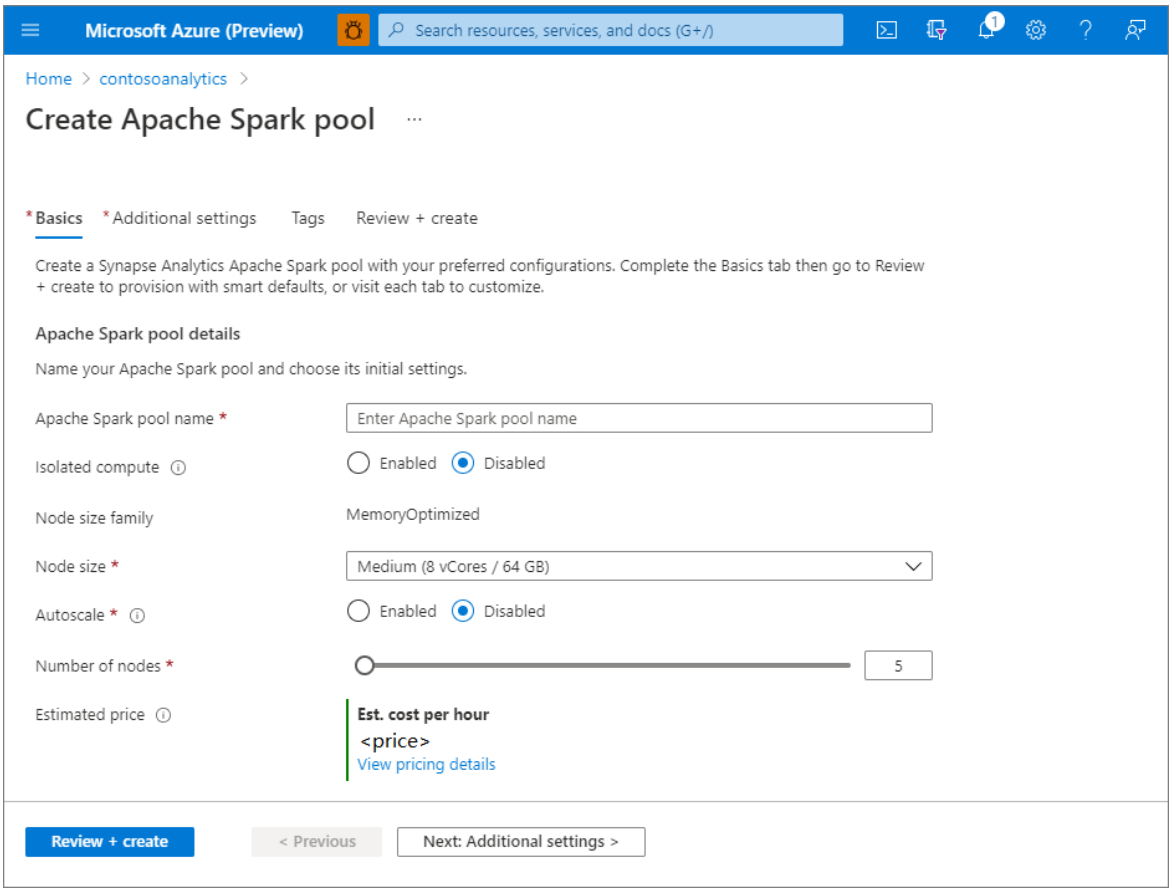
Create new Apache Spark pool

1. In the Synapse workspace where you want to create the Apache Spark pool, select **New Apache Spark pool**.



2. Enter the following details in the **Basics** tab:

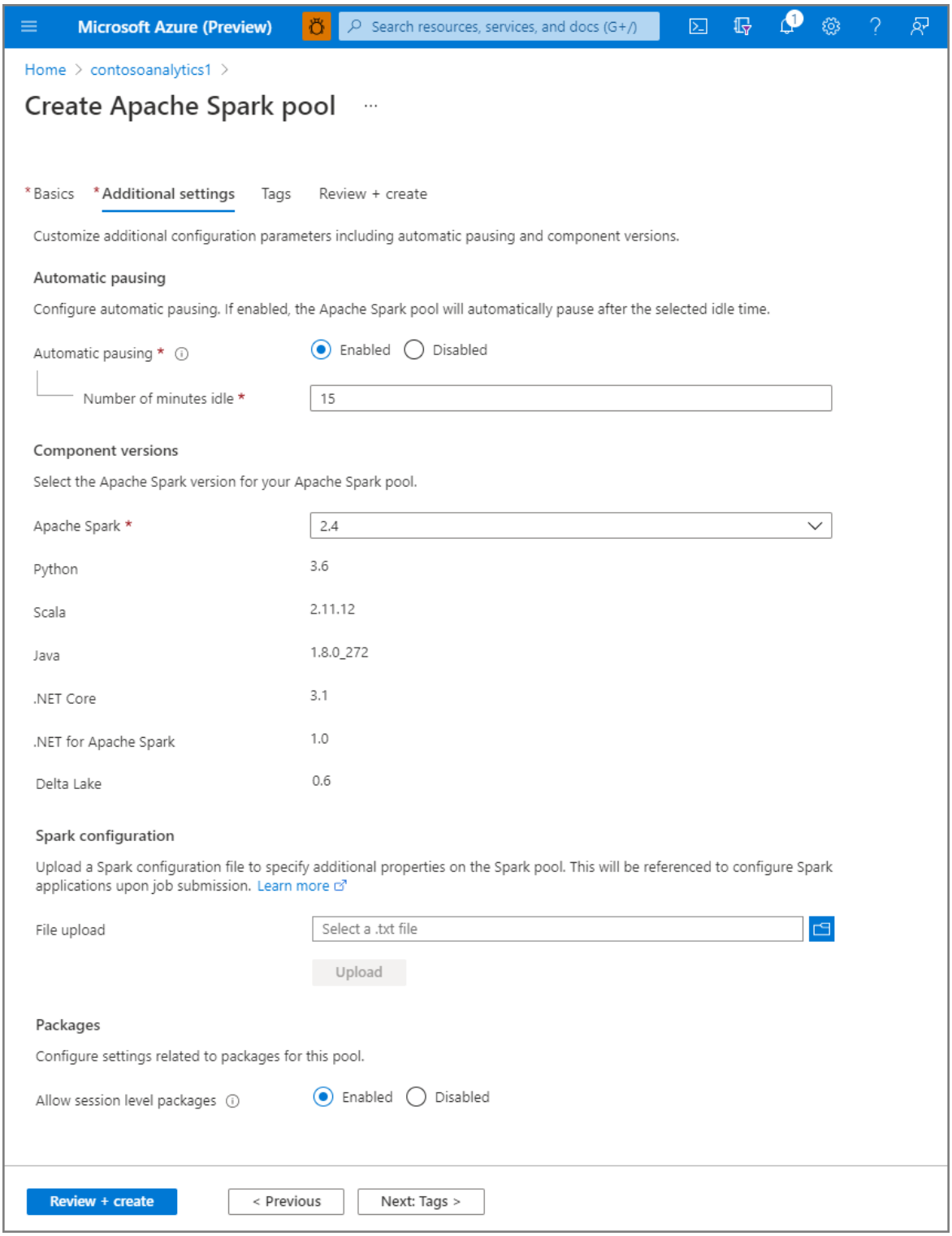
SETTING	SUGGESTED VALUE	DESCRIPTION
Apache Spark pool name	A valid pool name	This is the name that the Apache Spark pool will have.
Node size	Small (4 vCPU / 32 GB)	Set this to the smallest size to reduce costs for this quickstart
Autoscale	Disabled	We don't need autoscale for this quickstart
Number of nodes	5	Use a small size to limit costs for this quickstart



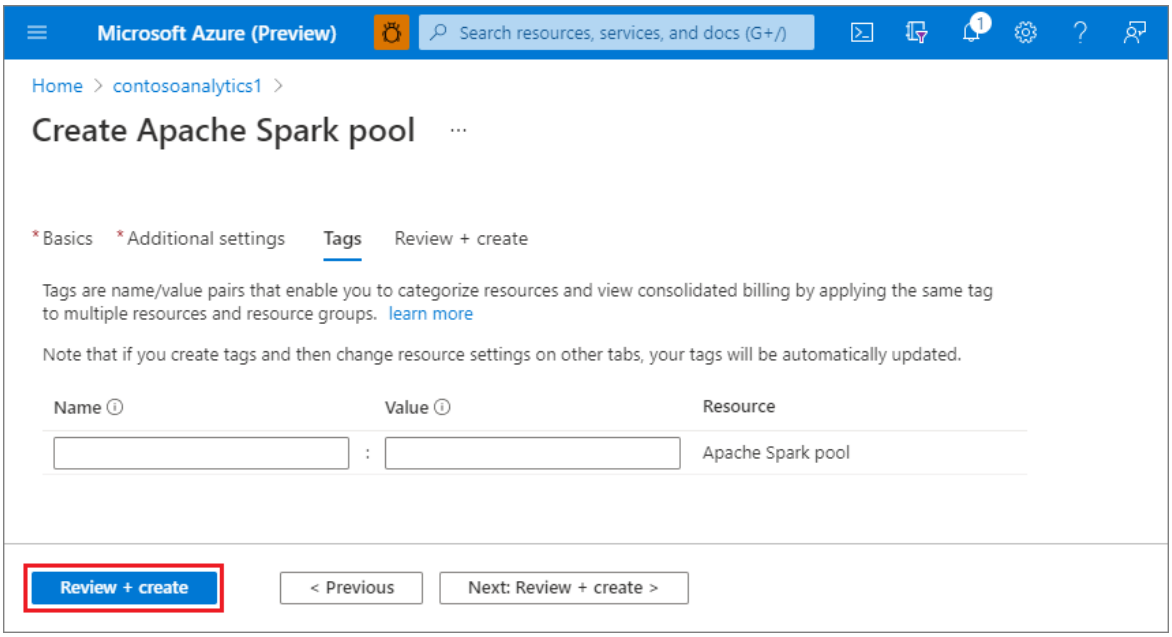
IMPORTANT

Note that there are specific limitations for the names that Apache Spark pools can use. Names must contain letters or numbers only, must be 15 or less characters, must start with a letter, not contain reserved words, and be unique in the workspace.

3. Select **Next: additional settings** and review the default settings. Do not modify any default settings.

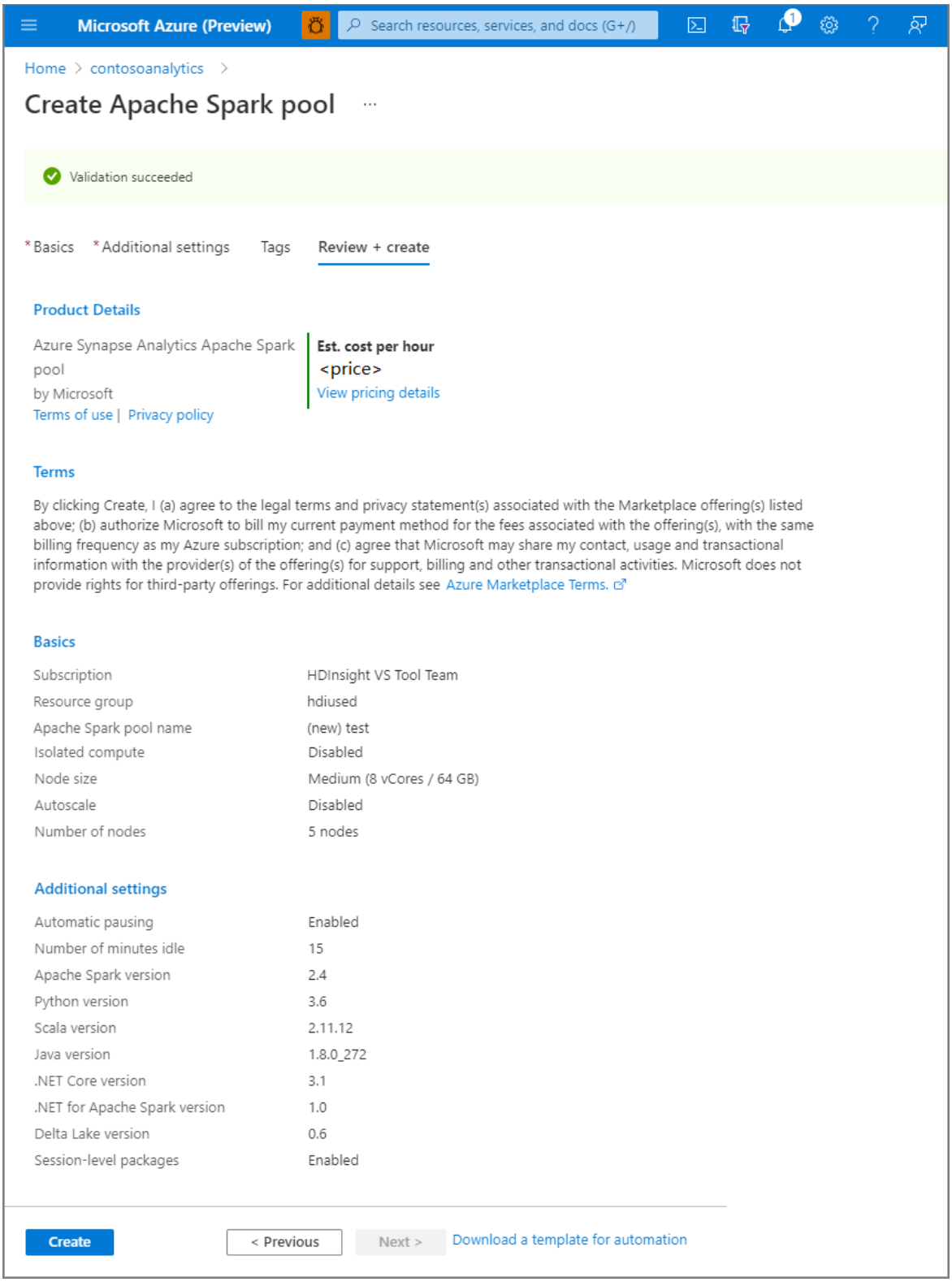


4. Select **Next: tags**. Don't add any tags.

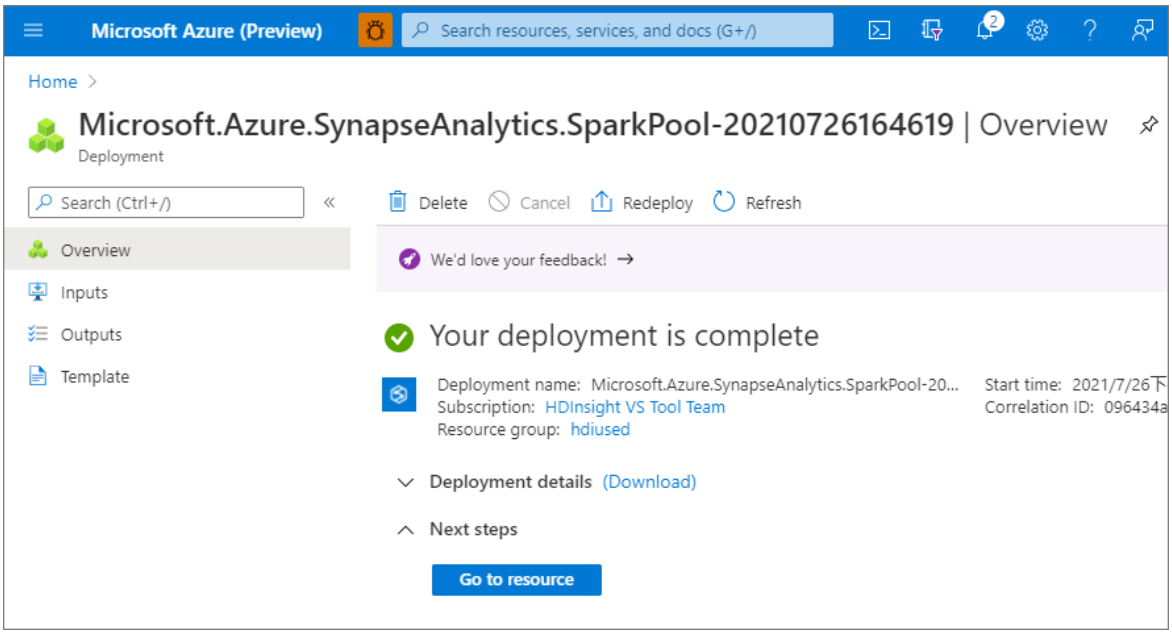


5. Select **Review + create**.

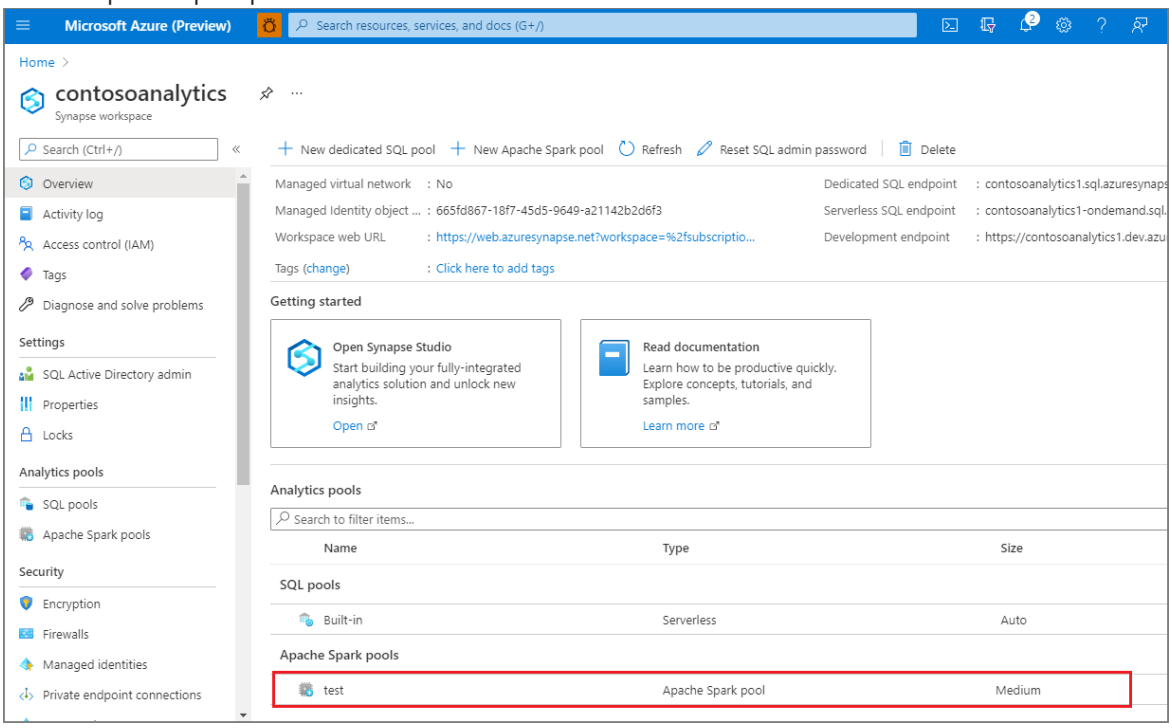
6. Make sure that the details look correct based on what was previously entered, and select **Create**.



7. At this point, the resource provisioning flow will start, indicating once it's complete.



8. After the provisioning completes, navigating back to the workspace will show a new entry for the newly created Apache Spark pool.



9. At this point, there are no resources running, no charges for Spark, you have created metadata about the Spark instances you want to create.

Clean up resources

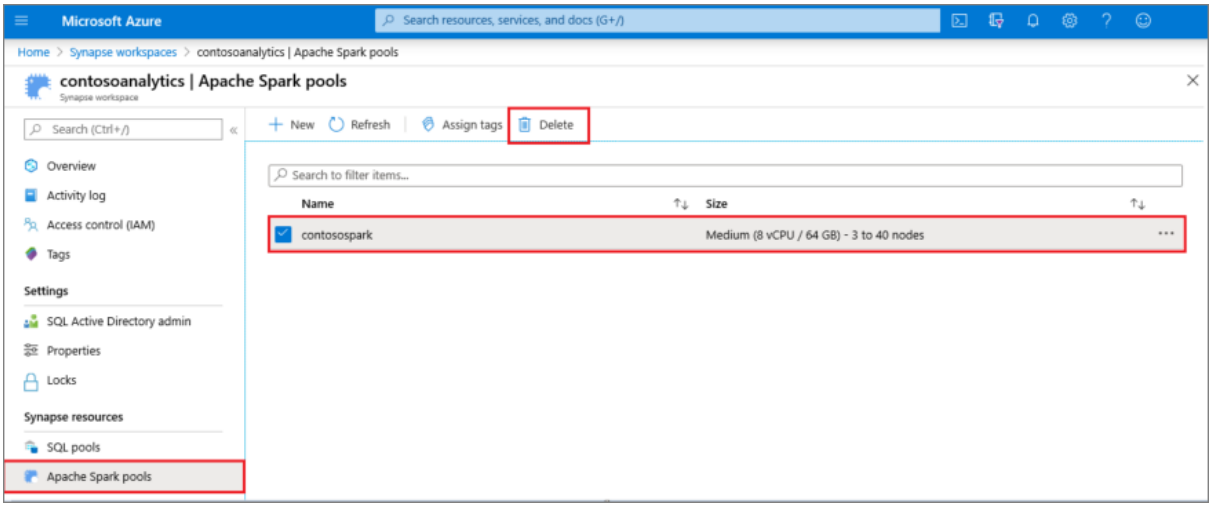
Follow the steps below to delete the Apache Spark pool from the workspace.

> **WARNING**
> Deleting an Apache Spark pool will remove the analytics engine from the workspace. It will no longer be possible to connect to the pool, and all queries, pipelines, and notebooks that use this Apache Spark pool will no longer work.

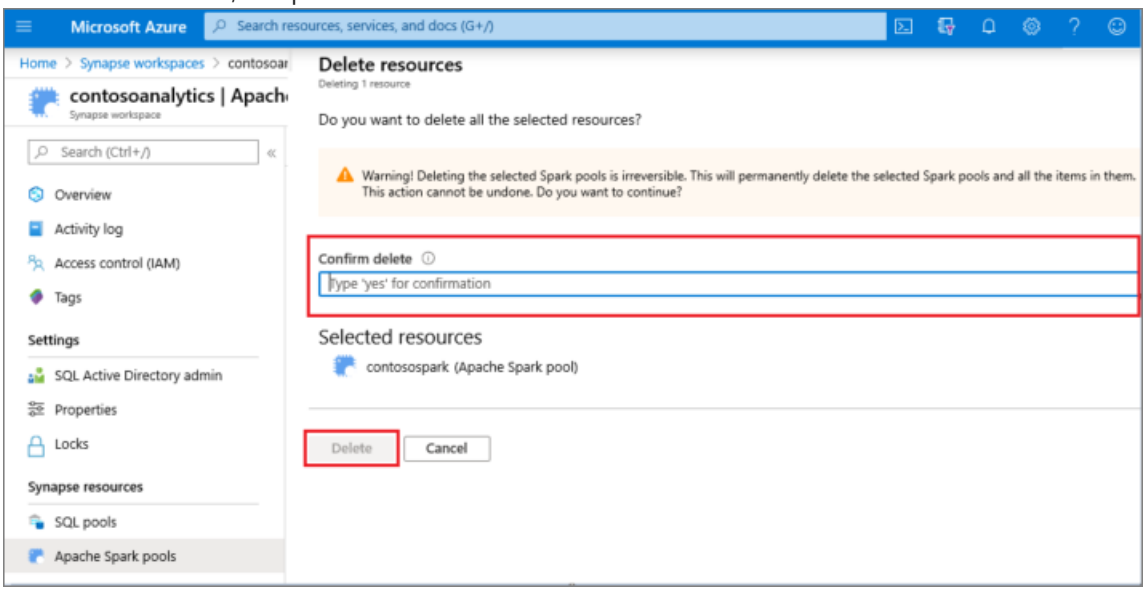
If you want to delete the Apache Spark pool, do the following:

1. Navigate to the Apache Spark pools blade in the workspace.

2. Select the Apache Spark pool to be deleted (in this case, **contosospark**).
3. Press **delete**.



4. Confirm the deletion, and press **Delete** button.



5. When the process completes successfully, the Apache Spark pool will no longer be listed in the workspace resources.

Next steps

- See Quickstart: Create an Apache Spark notebook.
- See Quickstart: Create a dedicated SQL pool using the Azure portal.

Quickstart: Create a serverless Apache Spark pool using Synapse Studio

8/19/2022 • 3 minutes to read • Edit Online

Azure Synapse Analytics offers various analytics engines to help you ingest, transform, model, analyze, and serve your data. Apache Spark pool offers open-source big data compute capabilities. After creating an Apache Spark pool in your Synapse workspace, data can be loaded, modeled, processed, and served to obtain insights.

This quickstart describes the steps to create an Apache Spark pool in a Synapse workspace by using Synapse Studio.

> **IMPORTANT**
> Billing for Spark instances is prorated per minute, whether you are using them or not. Be sure to shutdown your Spark instance after you have finished using it, or set a short timeout. For more information, see the **Clean up resources** section of this article.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

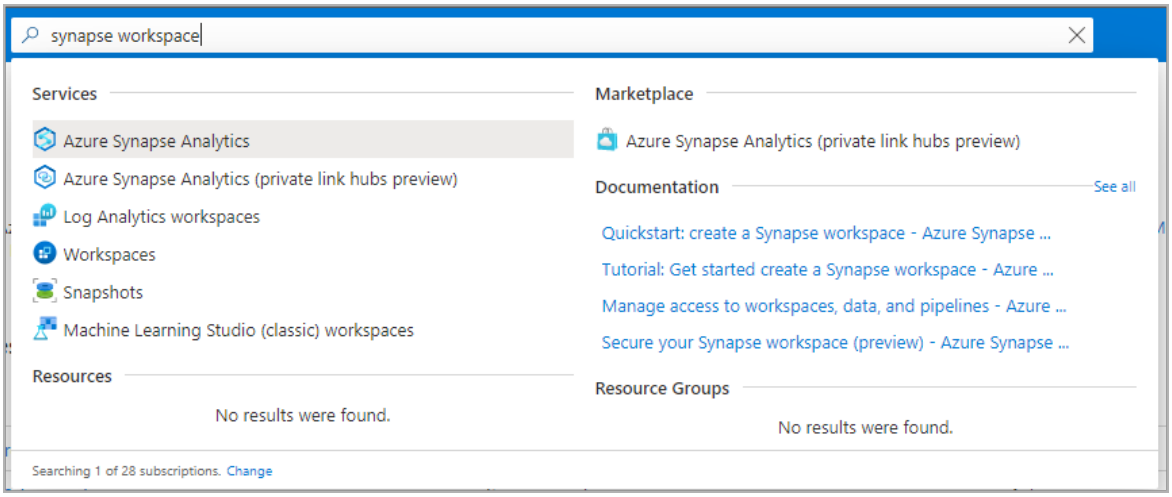
- Azure subscription - create one for free
- Synapse workspace

Sign in to the Azure portal

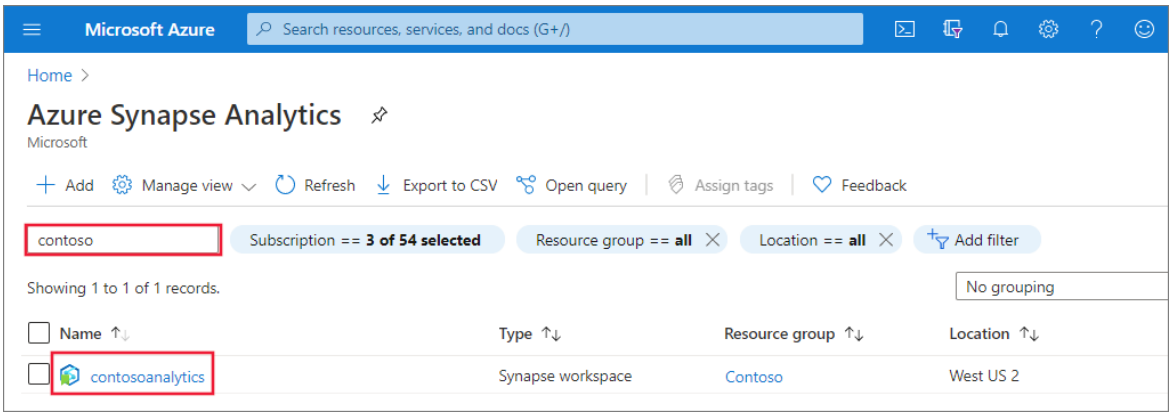
Sign in to the Azure portal

Navigate to the Synapse workspace

1. Navigate to the Synapse workspace where the Apache Spark pool will be created by typing the service name (or resource name directly) into the search bar.

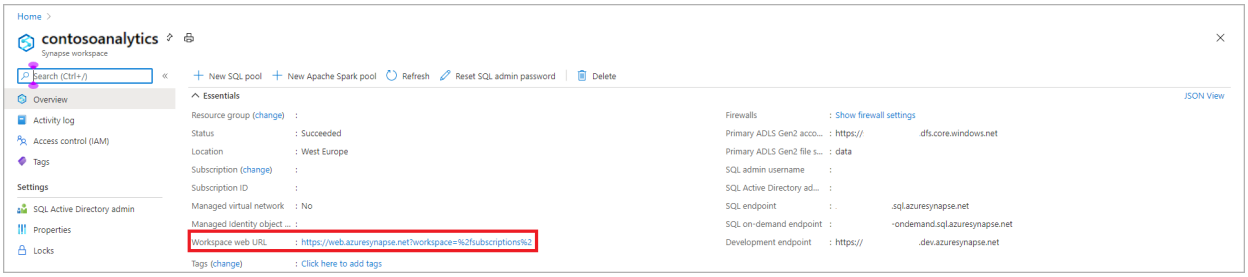


2. From the list of workspaces, type the name (or part of the name) of the workspace to open. For this example, we'll use a workspace named **contosoanalytics**.



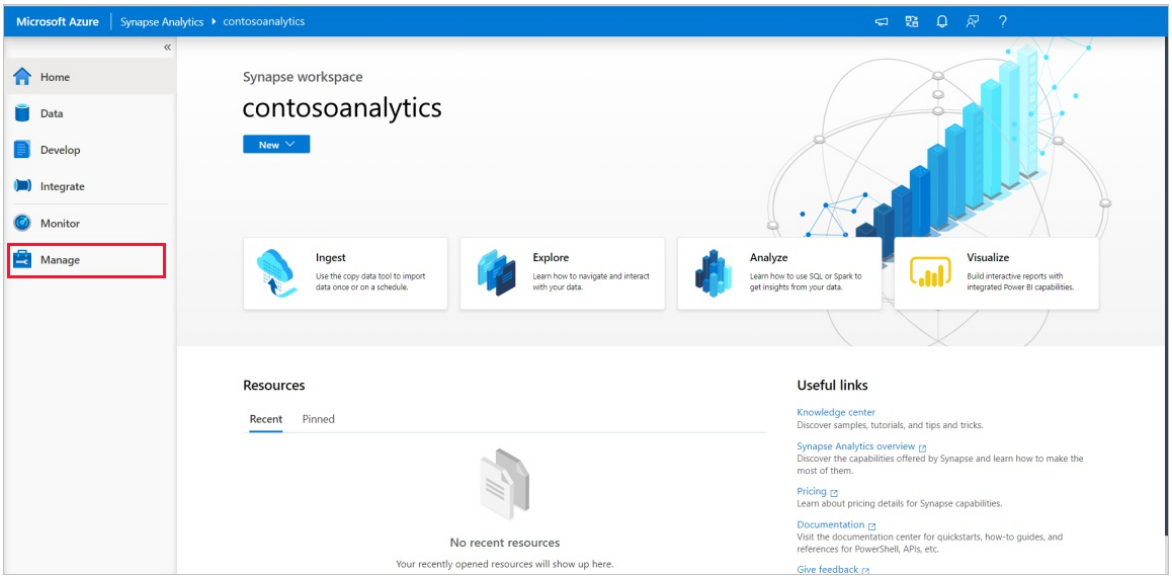
Launch Synapse Studio

From the workspace overview, select the **Workspace web URL** to open Synapse Studio.

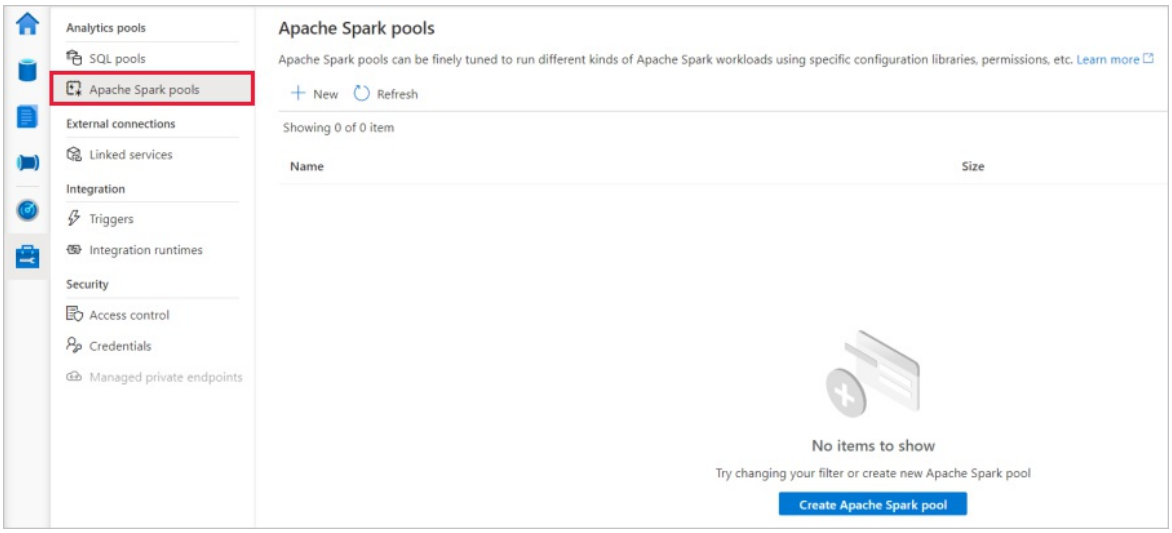


Create the Apache Spark pool in Synapse Studio

1. On the Synapse Studio home page, navigate to the **Management Hub** in the left navigation by selecting the **Manage** icon.



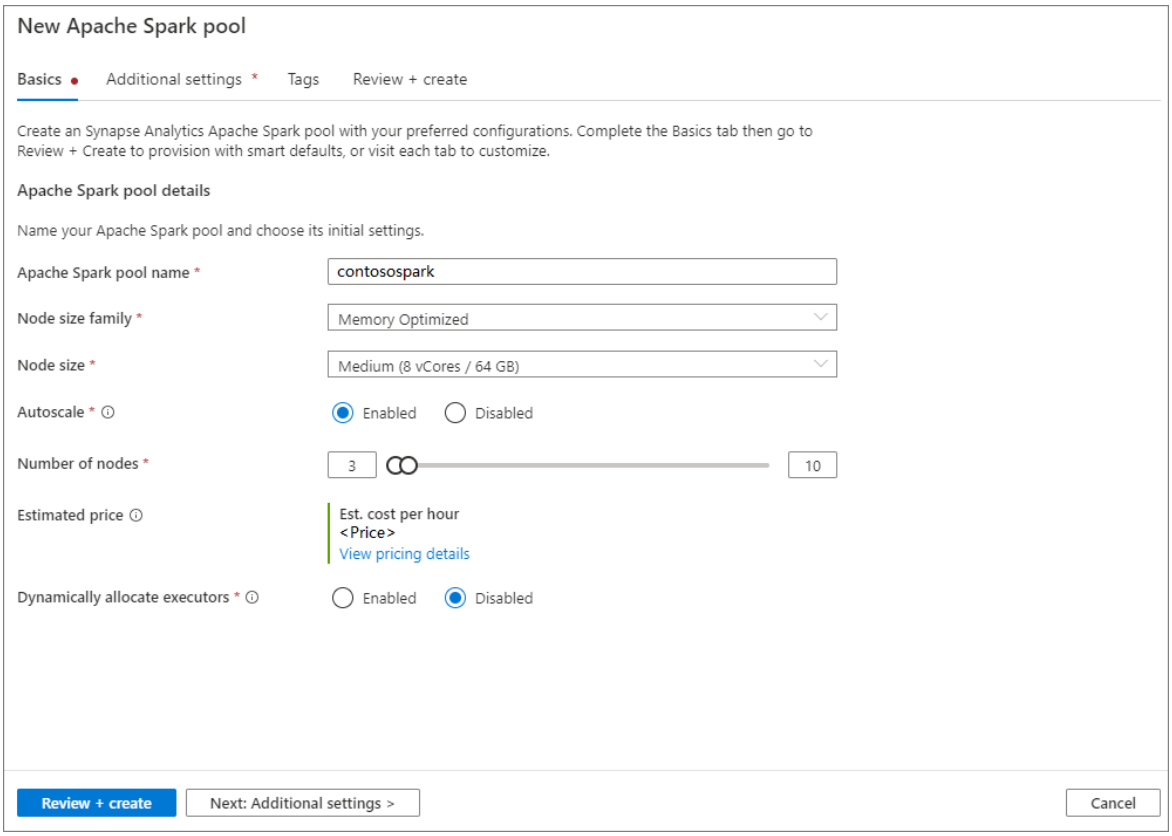
2. Once in the Management Hub, navigate to the **Apache Spark pools** section to see the current list of Apache Spark pools that are available in the workspace.



3. Select **+ New** and the new Apache Spark pool create wizard will appear.

4. Enter the following details in the **Basics** tab:

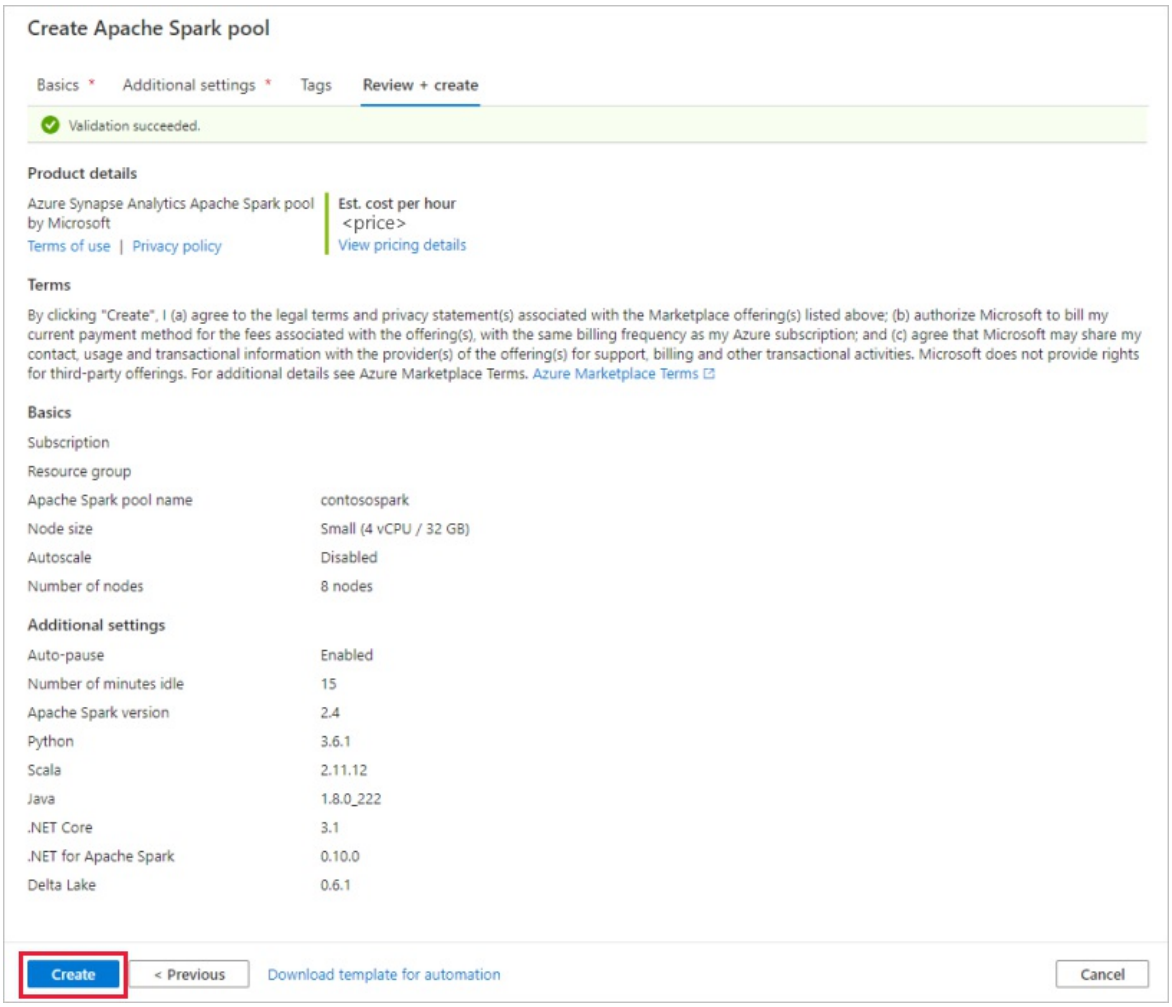
SETTING	SUGGESTED VALUE	DESCRIPTION
Apache Spark pool name	contosospark	This is the name that the Apache Spark pool will have.
Node size	Small (4 vCPU / 32 GB)	Set this to the smallest size to reduce costs for this quickstart
Autoscale	Disabled	We won't need autoscale in this quickstart
Number of nodes	8	Use a small size to limit costs in this quickstart
Dynamically allocate executors	Disabled	This setting maps to the dynamic allocation property in Spark configuration for Spark Application executors allocation. We won't need autoscale in this quickstart.



IMPORTANT

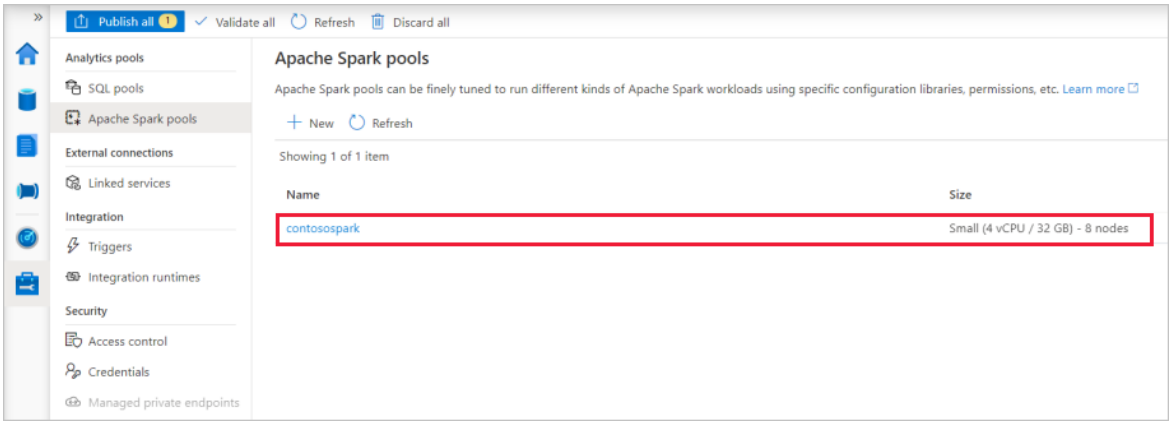
Note that there are specific limitations for the names that Apache Spark pools can use. Names must contain letters or numbers only, must be 15 or less characters, must start with a letter, not contain reserved words, and be unique in the workspace.

5. In the next tab (Additional settings), leave all settings as defaults.
6. We won't add any tags for now, so select **Review + create**.
7. In the **Review + create** tab, make sure that the details look correct based on what was previously entered, and press **Create**.



8. The Apache Spark pool will start the provisioning process.

9. Once the provisioning is complete, the new Apache Spark pool will appear in the list.



Clean up Apache Spark pool resources using Synapse Studio

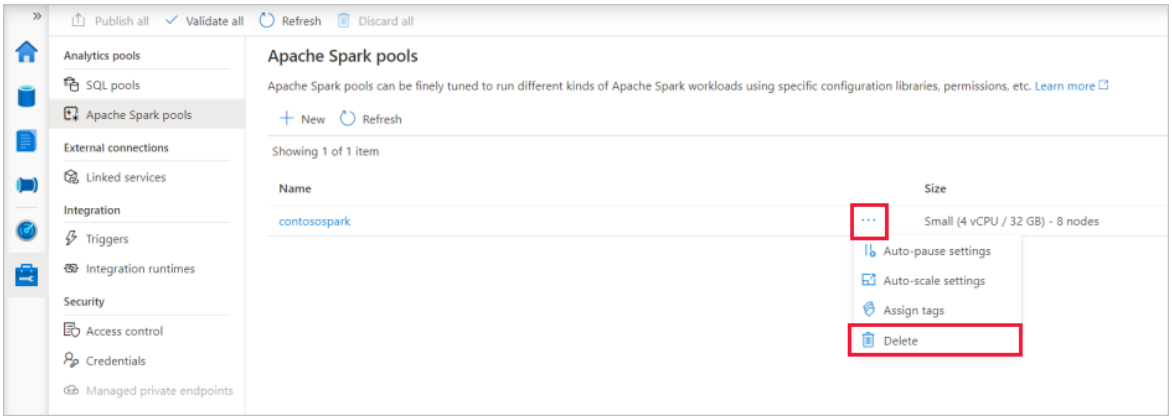
Follow the steps below to delete the Apache Spark pool from the workspace using Synapse Studio.

> **WARNING**
> Deleting a Spark pool will remove the analytics engine from the workspace. It will no longer be possible to connect to the pool, and all queries, pipelines, and notebooks that use this Spark pool will no longer work.

If you want to delete the Apache Spark pool, do the following:

1. Navigate to the Apache Spark pools in the Management Hub in Synapse Studio.

2. Select the ellipsis next to the Apache pool to be deleted (in this case, **contosospark**) to show the commands for the Apache Spark pool.



3. Press **delete**.

4. Confirm the deletion, and press **Delete** button.

5. When the process completes successfully, the Apache Spark pool will no longer be listed in the workspace resources.

Next steps

- See Quickstart: Create an Apache Spark pool in Synapse Studio using web tools.
- See Quickstart: Create an Apache Spark pool using the Azure portal.

Quickstart: Create an Apache Spark GPU-enabled Pool in Azure Synapse Analytics using the Azure portal

8/19/2022 • 2 minutes to read • Edit Online

An Apache Spark pool provides open-source big data compute capabilities where data can be loaded, modeled, processed, and distributed for faster analytic insight. Synapse now offers the ability to create Apache Spark pools that use GPUs on the backend to run your Spark workloads on GPUs for accelerated processing.

In this quickstart, you learn how to use the Azure portal to create an Apache Spark GPU-enabled pool in an Azure Synapse Analytics workspace.

> **NOTE**
> Azure Synapse GPU-enabled pools are currently in Public Preview.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

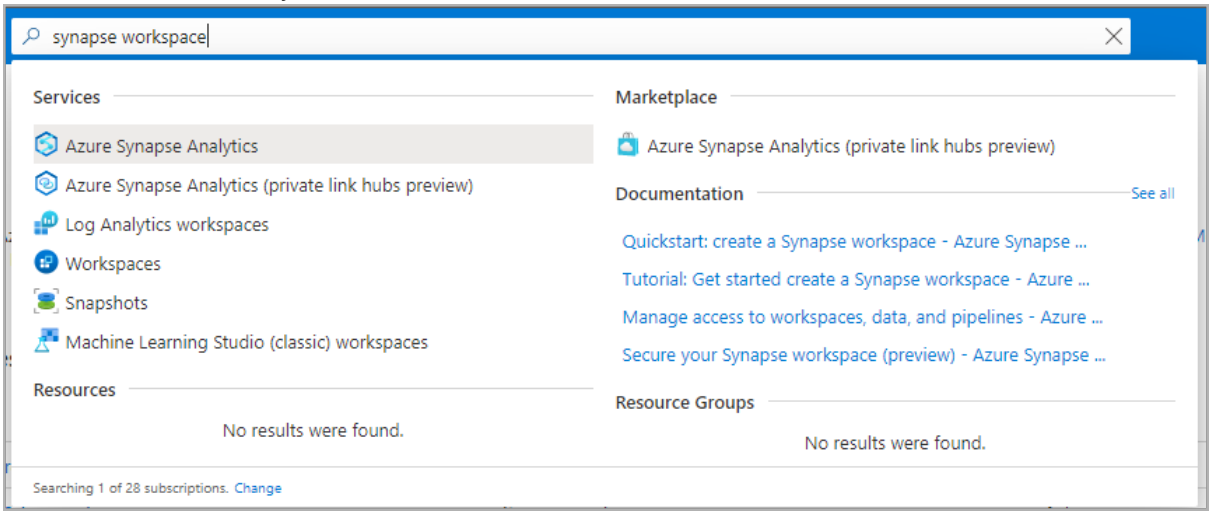
- Azure subscription - create one for free
- Synapse Analytics workspace

Sign in to the Azure portal

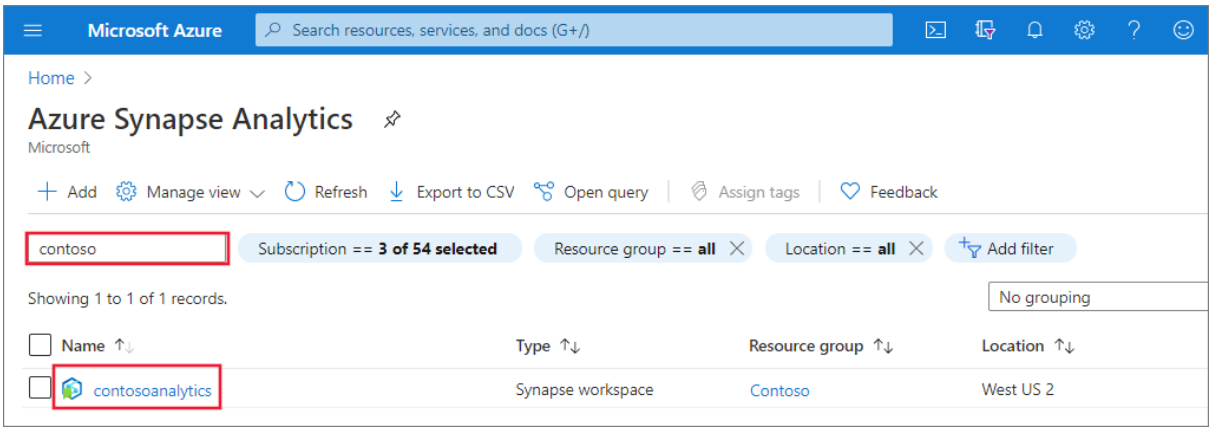
Sign in to the Azure portal

Navigate to the Synapse workspace

1. Navigate to the Synapse workspace where the Apache Spark pool will be created by typing the service name (or resource name directly) into the search bar.

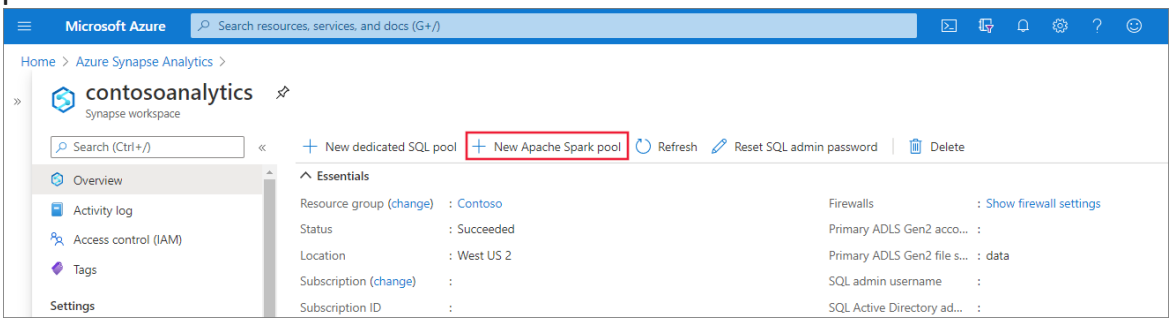


2. From the list of workspaces, type the name (or part of the name) of the workspace to open. For this example, we'll use a workspace named **contosoanalytics**.



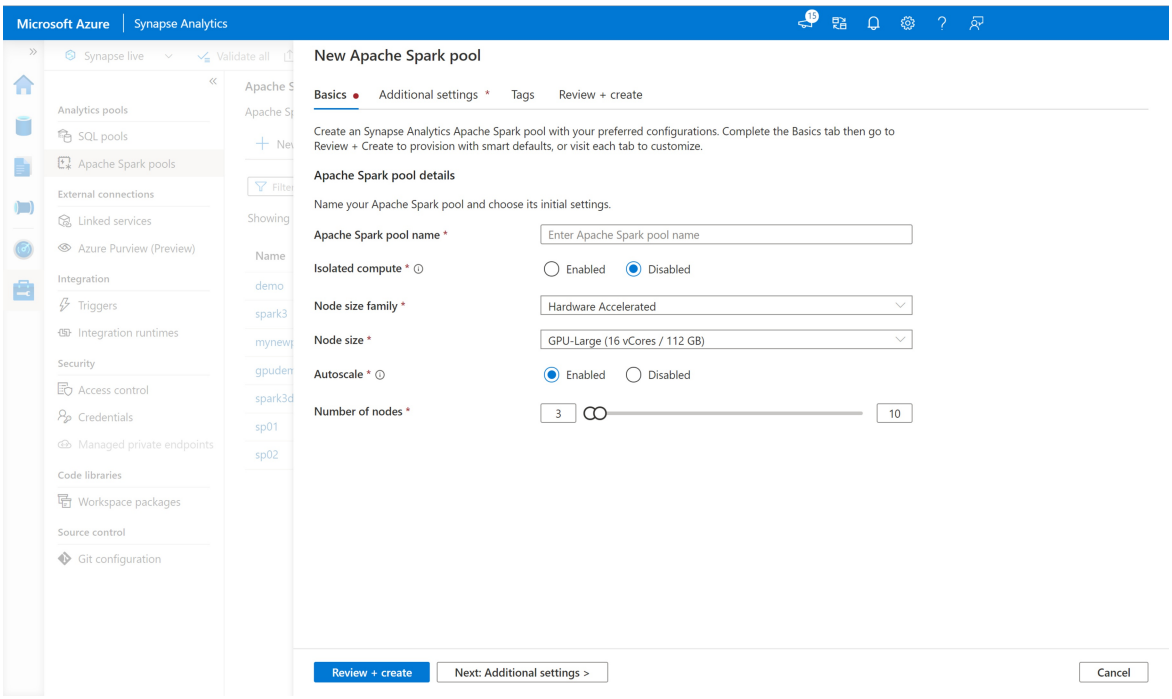
Create new Azure Synapse GPU-enabled pool

1. In the Synapse workspace where you want to create the Apache Spark pool, select **New Apache Spark pool**.



2. Enter the following details in the **Basics** tab:

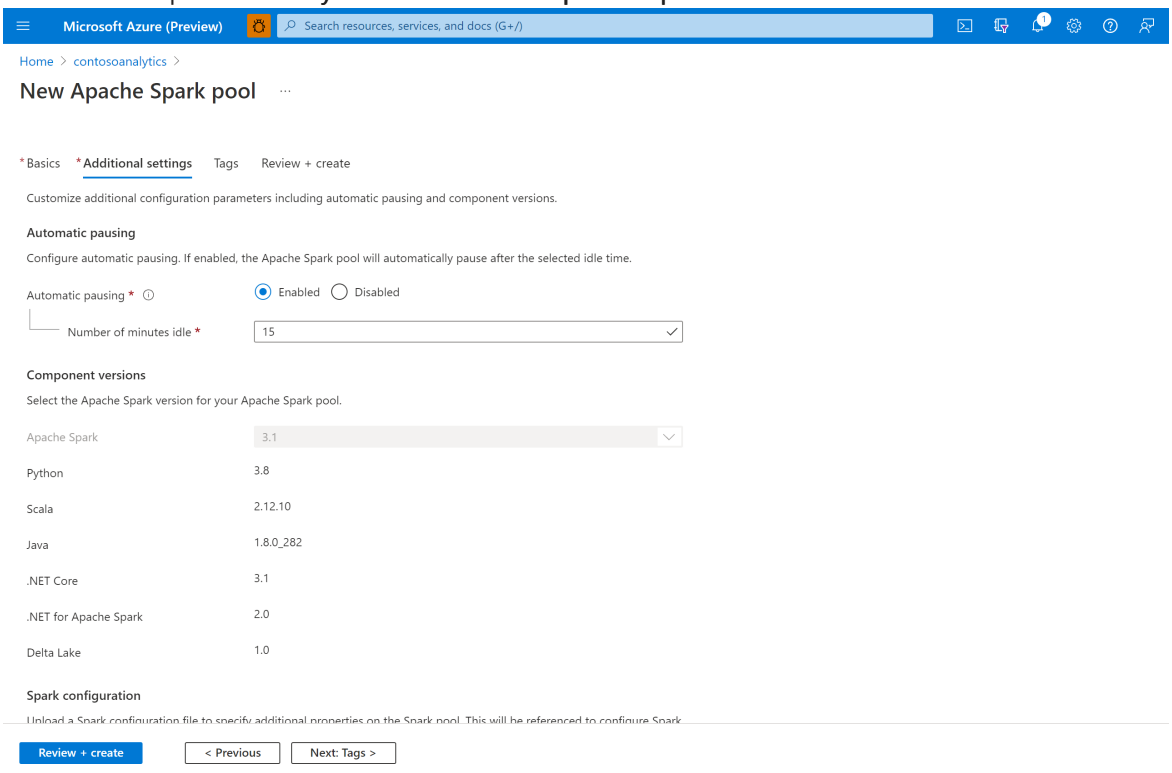
SETTING	SUGGESTED VALUE	DESCRIPTION
Apache Spark pool name	A valid pool name	This is the name that the Apache Spark pool will have.
Node size family	Hardware Accelerated	Choose Hardware Accelerated from the drop down menu
Node size	Large (16 vCPU / 110 GB / 1 GPU)	Set this to the smallest size to reduce costs for this quickstart
Autoscale	Disabled	We don't need autoscale for this quickstart
Number of nodes	3	Use a small size to limit costs for this quickstart



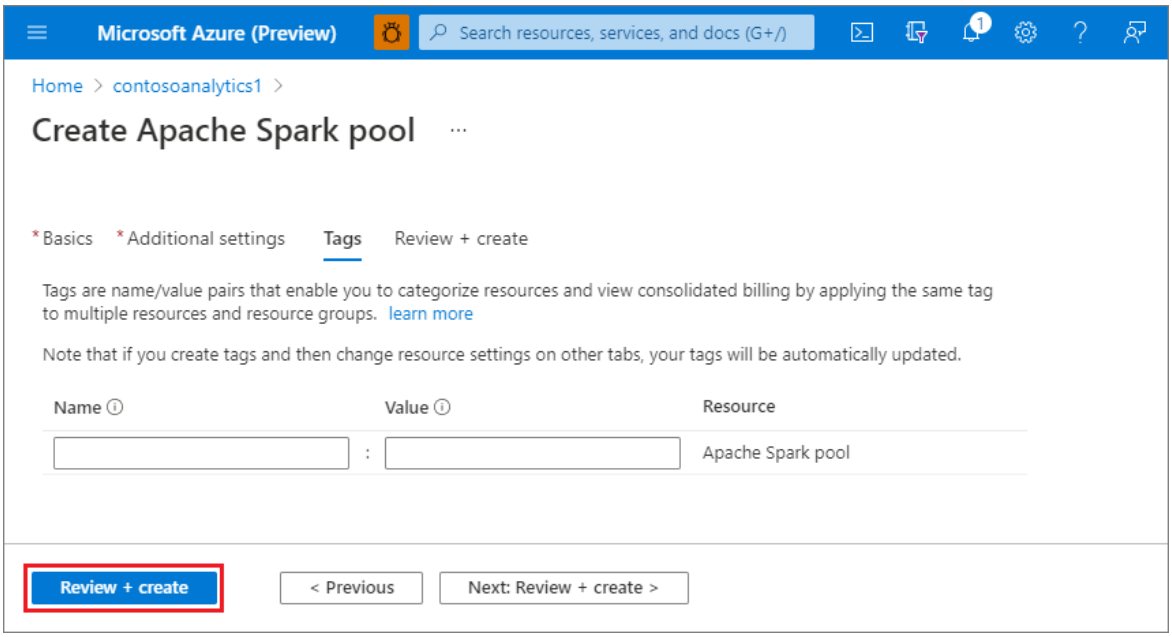
IMPORTANT

Note that there are specific limitations for the names that Apache Spark pools can use. Names must contain letters or numbers only, must be 15 or less characters, must start with a letter, not contain reserved words, and be unique in the workspace.

3. Select **Next: additional settings** and review the default settings. Do not modify any default settings. Note that GPU pools can **only be created with Apache Spark 3.1**.

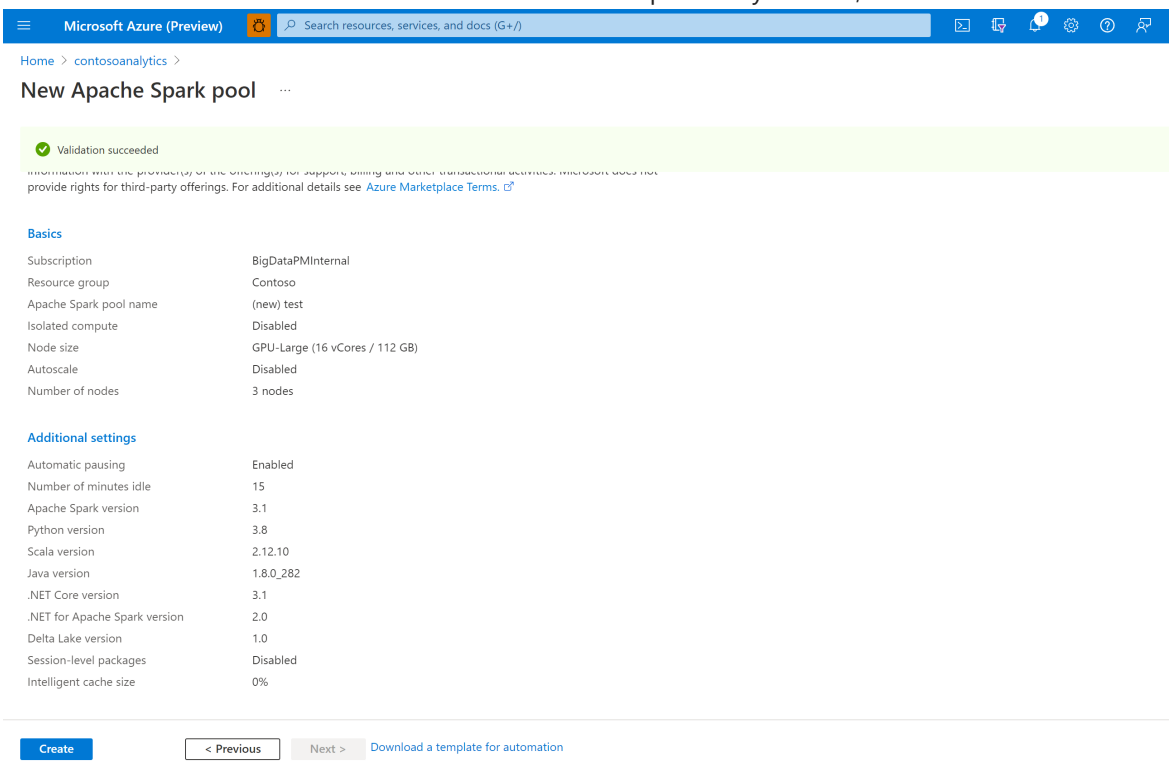


4. Select **Next: tags**. Don't add any tags.

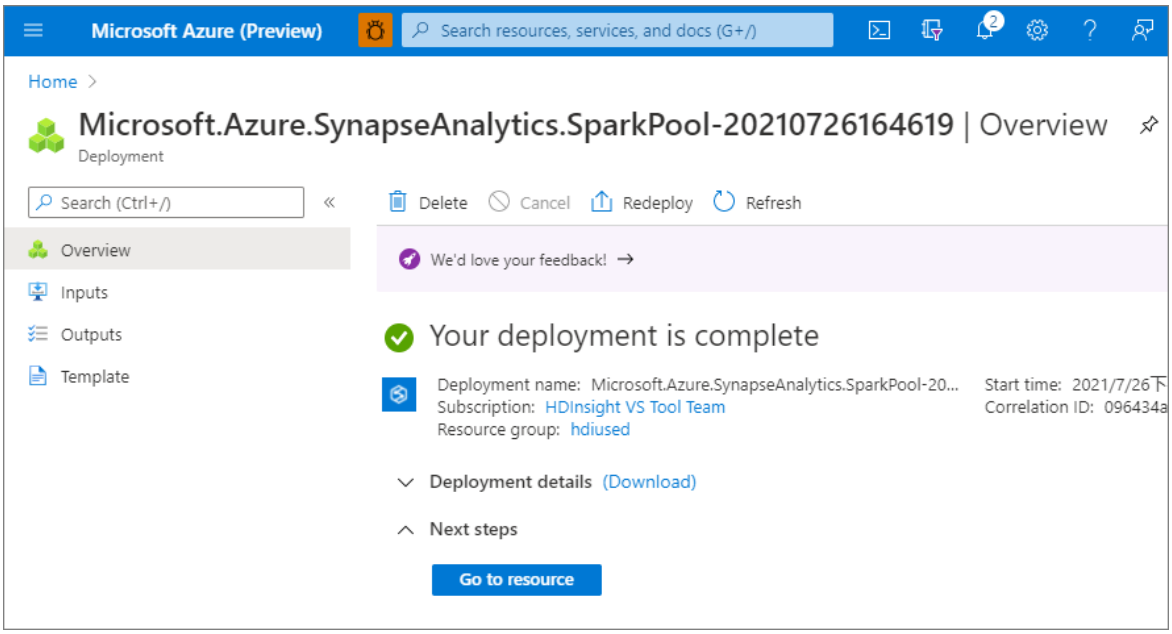


5. Select **Review + create**.

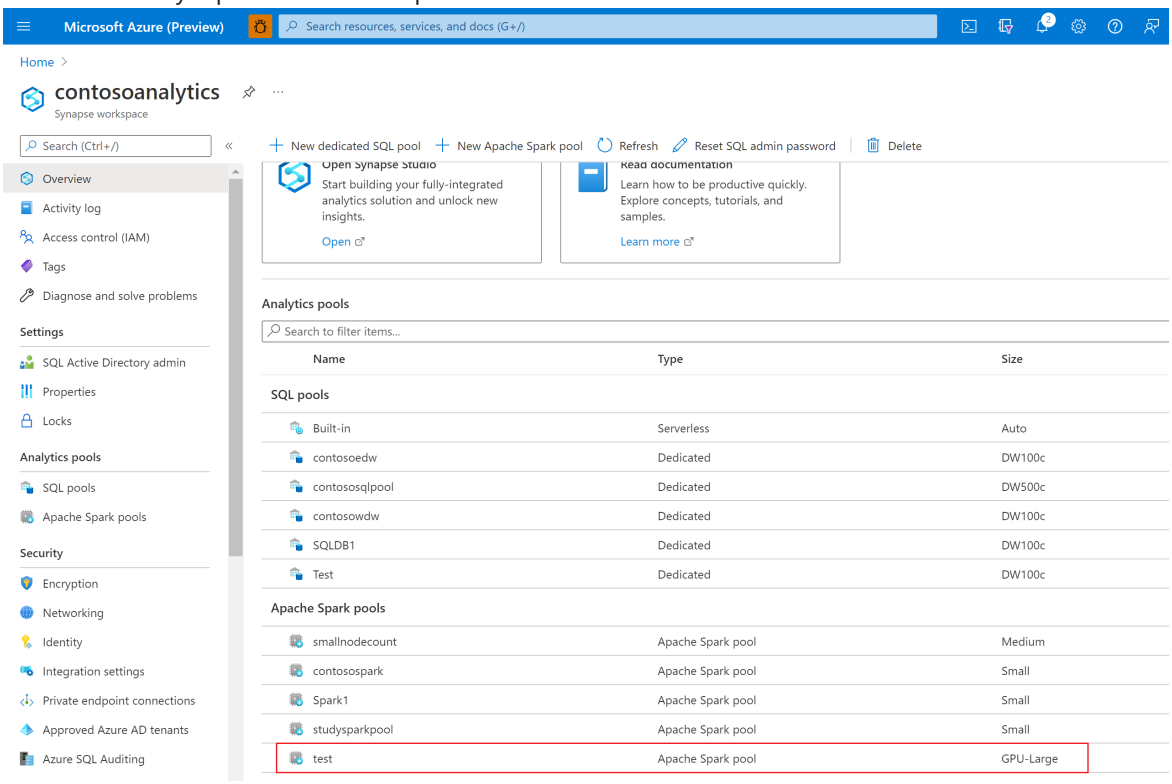
6. Make sure that the details look correct based on what was previously entered, and select **Create**.



7. At this point, the resource provisioning flow will start, indicating once it's complete.



8. After the provisioning completes, navigating back to the workspace will show a new entry for the newly created Azure Synapse GPU-enabled pool.



9. At this point, there are no resources running, no charges for Spark, you have created metadata about the Spark instances you want to create.

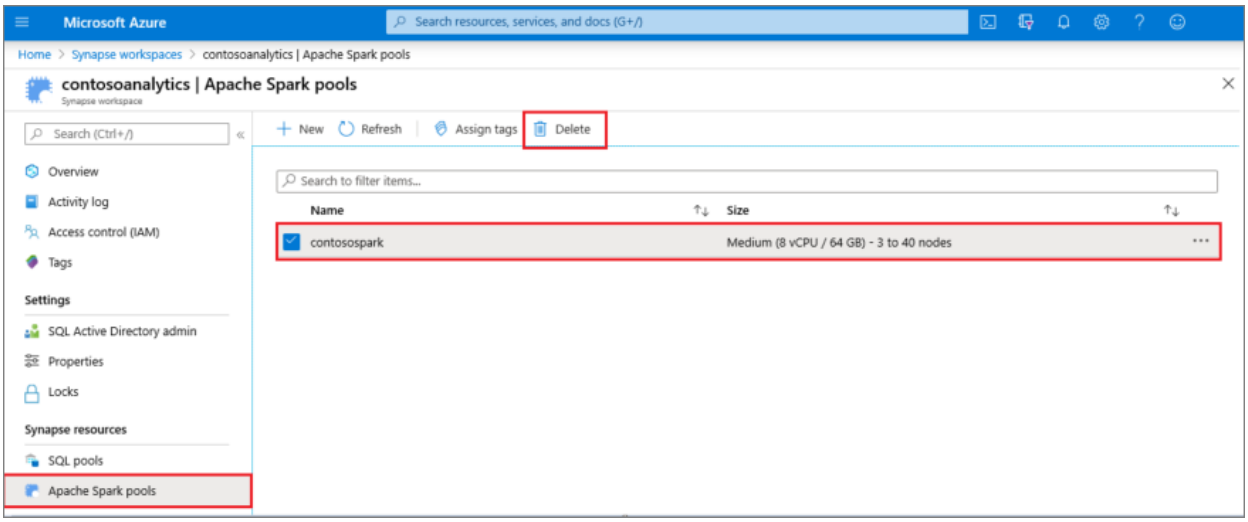
Clean up resources

Follow the steps below to delete the Apache Spark pool from the workspace.

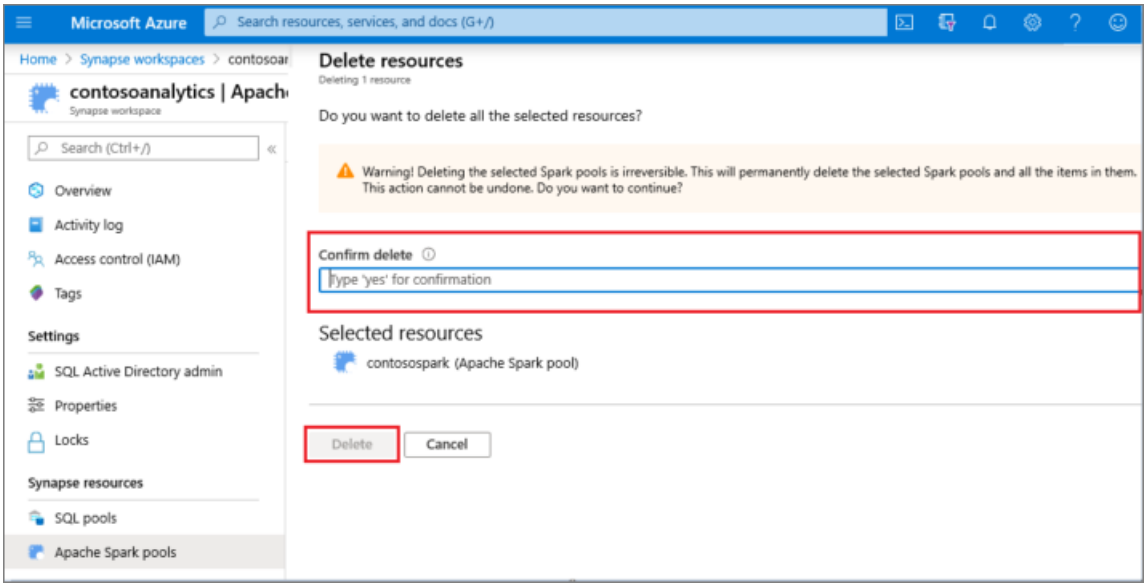
> **WARNING**
>
> Deleting an Apache Spark pool will remove the analytics engine from the workspace. It will no longer be possible to connect to the pool, and all queries, pipelines, and notebooks that use this Apache Spark pool will no longer work.

If you want to delete the Apache Spark pool, do the following:

1. Navigate to the Apache Spark pools blade in the workspace.
2. Select the Apache Spark pool to be deleted (in this case, **contosospark**).
3. Press **delete**.



4. Confirm the deletion, and press **Delete** button.



5. When the process completes successfully, the Apache Spark pool will no longer be listed in the workspace resources.

Next steps

- See Quickstart: Create an Apache Spark notebook to run on a GPU pool.

Analyze data with Apache Spark

8/19/2022 • 5 minutes to read • Edit Online

In this tutorial, you'll learn how to perform exploratory data analysis by using Azure Open Datasets and Apache Spark. You can then visualize the results in a Synapse Studio notebook in Azure Synapse Analytics.

In particular, we'll analyze the New York City (NYC) Taxi dataset. The data is available through Azure Open Datasets. This subset of the dataset contains information about yellow taxi trips: information about each trip, the start and end time and locations, the cost, and other interesting attributes.

Before you begin

Create an Apache Spark Pool by following the Create an Apache Spark pool tutorial.

Download and prepare the data

1. Create a notebook by using the PySpark kernel. For instructions, see Create a notebook.

   > **NOTE**
   > Because of the PySpark kernel, you don't need to create any contexts explicitly. The Spark context is automatically created for you when you run the first code cell.

2. In this tutorial, we'll use several different libraries to help us visualize the dataset. To do this analysis, import the following libraries:

   ```
   import matplotlib.pyplot as plt
   import seaborn as sns
   import pandas as pd
   ```

3. Because the raw data is in a Parquet format, you can use the Spark context to pull the file into memory as a DataFrame directly. Create a Spark DataFrame by retrieving the data via the Open Datasets API. Here, we use the Spark DataFrame *schema on read* properties to infer the datatypes and schema.

   ```
   from azureml.opendatasets import NycTlcYellow
   from datetime import datetime
   from dateutil import parser

   end_date = parser.parse('2018-06-06')
   start_date = parser.parse('2018-05-01')
   nyc_tlc = NycTlcYellow(start_date=start_date, end_date=end_date)
   df = nyc_tlc.to_spark_dataframe()
   ```

4. After the data is read, we'll want to do some initial filtering to clean the dataset. We might remove unneeded columns and add columns that extract important information. In addition, we'll filter out anomalies within the dataset.

```
# Filter the dataset
from pyspark.sql.functions import *

filtered_df = df.select('vendorID', 'passengerCount', 'tripDistance','paymentType', 'fareAmount',
'tipAmount'\
                                , date_format('tpepPickupDateTime', 'hh').alias('hour_of_day')\
                                , dayofweek('tpepPickupDateTime').alias('day_of_week')\
                                , dayofmonth(col('tpepPickupDateTime')).alias('day_of_month'))\
                            .filter((df.passengerCount > 0)\
                                & (df.tipAmount >= 0)\
                                & (df.fareAmount >= 1) & (df.fareAmount <= 250)\
                                & (df.tripDistance > 0) & (df.tripDistance <= 200))

filtered_df.createOrReplaceTempView("taxi_dataset")
```

Analyze data

As a data analyst, you have a wide range of tools available to help you extract insights from the data. In this part of the tutorial, we'll walk through a few useful tools available within Azure Synapse Analytics notebooks. In this analysis, we want to understand the factors that yield higher taxi tips for our selected period.

Apache Spark SQL Magic

First, we'll perform exploratory data analysis by Apache Spark SQL and magic commands with the Azure Synapse notebook. After we have our query, we'll visualize the results by using the built-in `chart options` capability.

1. Within your notebook, create a new cell and copy the following code. By using this query, we want to understand how the average tip amounts have changed over the period we've selected. This query will also help us identify other useful insights, including the minimum/maximum tip amount per day and the average fare amount.

   ```
   %%sql
   SELECT
       day_of_month
       , MIN(tipAmount) AS minTipAmount
       , MAX(tipAmount) AS maxTipAmount
       , AVG(tipAmount) AS avgTipAmount
       , AVG(fareAmount) as fareAmount
   FROM taxi_dataset
   GROUP BY day_of_month
   ORDER BY day_of_month ASC
   ```

2. After our query finishes running, we can visualize the results by switching to the chart view. This example creates a line chart by specifying the `day_of_month` field as the key and `avgTipAmount` as the value. After you've made the selections, select **Apply** to refresh your chart.

Visualize data

In addition to the built-in notebook charting options, you can use popular open-source libraries to create your own visualizations. In the following examples, we'll use Seaborn and Matplotlib. These are commonly used Python libraries for data visualization.

NOTE

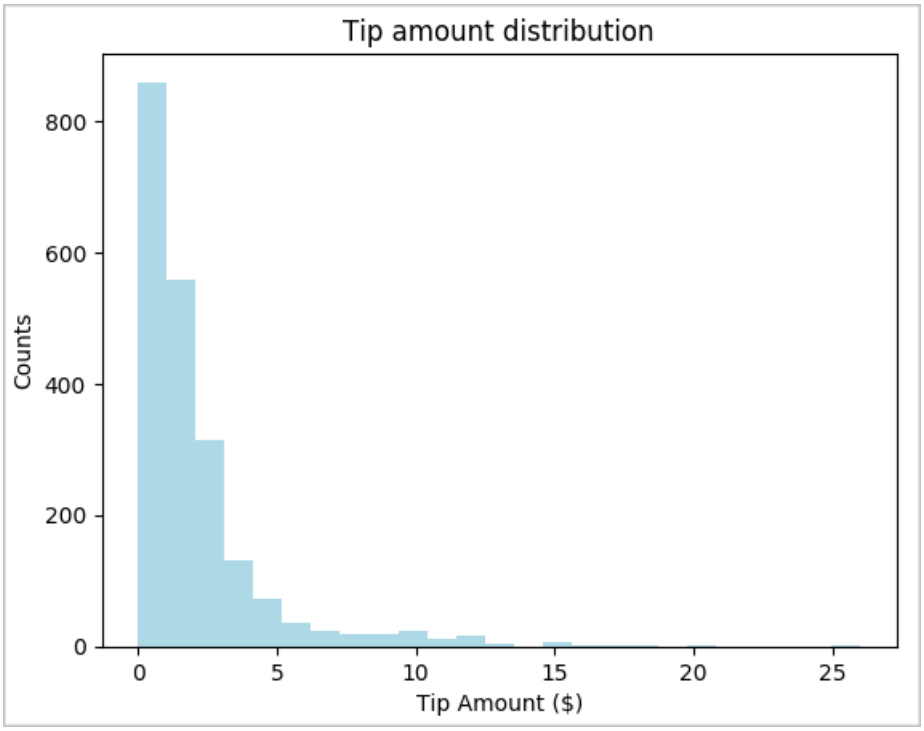
By default, every Apache Spark pool in Azure Synapse Analytics contains a set of commonly used and default libraries. You can view the full list of libraries in the Azure Synapse runtime documentation. In addition, to make third-party or locally built code available to your applications, you can install a library onto one of your Spark pools.

1. To make development easier and less expensive, we'll downsample the dataset. We'll use the built-in Apache Spark sampling capability. In addition, both Seaborn and Matplotlib require a Pandas DataFrame or NumPy array. To get a Pandas DataFrame, use the `toPandas()` command to convert the DataFrame.

```
# To make development easier, faster, and less expensive, downsample for now
sampled_taxi_df = filtered_df.sample(True, 0.001, seed=1234)

# The charting package needs a Pandas DataFrame or NumPy array to do the conversion
sampled_taxi_pd_df = sampled_taxi_df.toPandas()
```

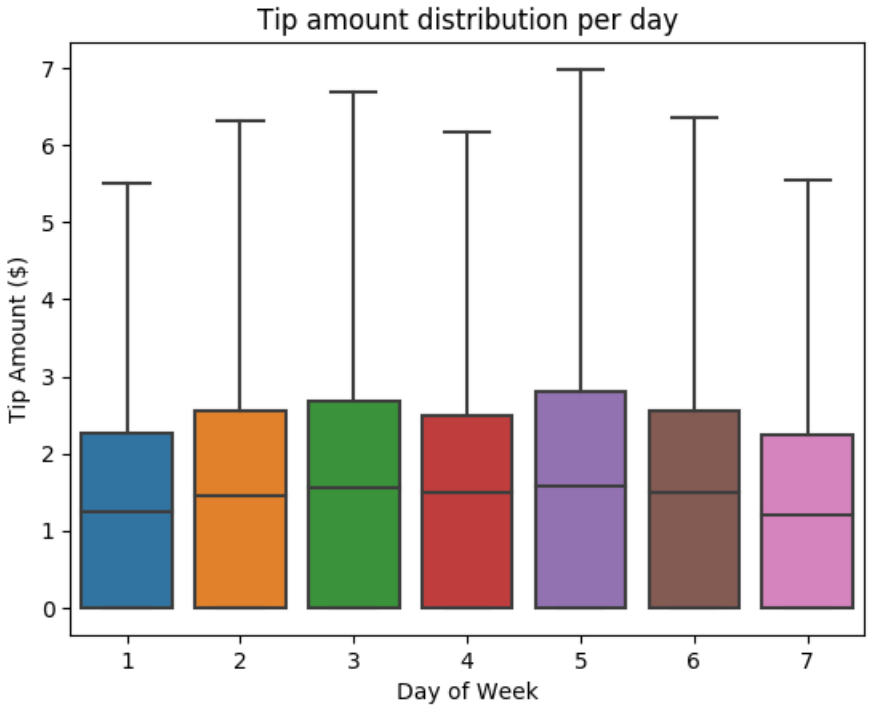
2. We want to understand the distribution of tips in our dataset. We'll use Matplotlib to create a histogram that shows the distribution of tip amount and count. Based on the distribution, we can see that tips are skewed toward amounts less than or equal to $10.

```
# Look at a histogram of tips by count by using Matplotlib

ax1 = sampled_taxi_pd_df['tipAmount'].plot(kind='hist', bins=25, facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()
```



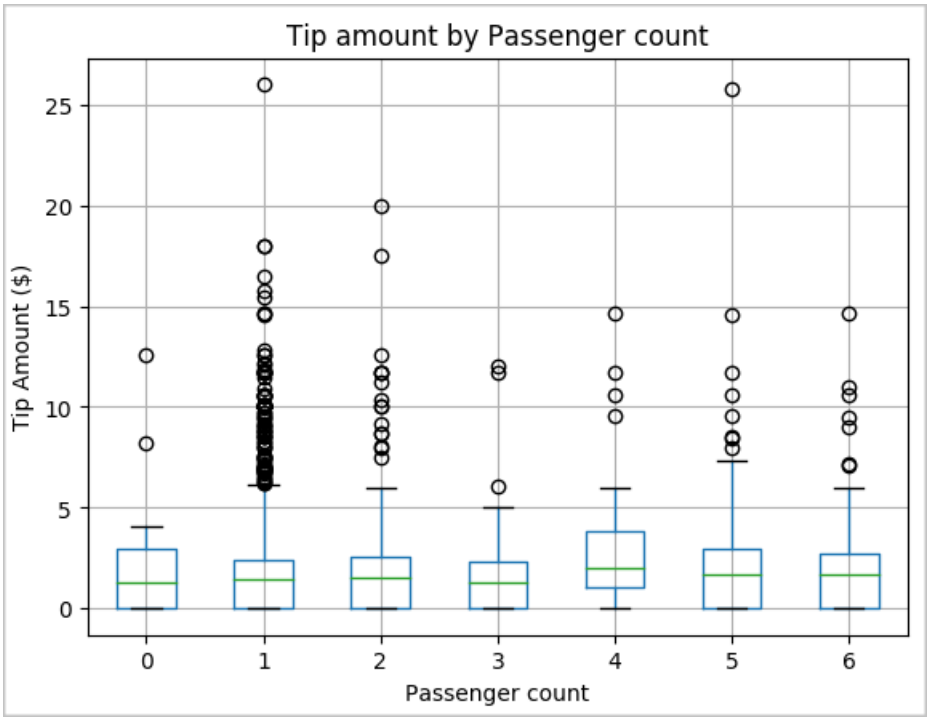
3. Next, we want to understand the relationship between the tips for a given trip and the day of the week. Use Seaborn to create a box plot that summarizes the trends for each day of the week.

```
# View the distribution of tips by day of week using Seaborn
ax = sns.boxplot(x="day_of_week", y="tipAmount",data=sampled_taxi_pd_df, showfliers = False)
ax.set_title('Tip amount distribution per day')
ax.set_xlabel('Day of Week')
ax.set_ylabel('Tip Amount ($)')
plt.show()
```



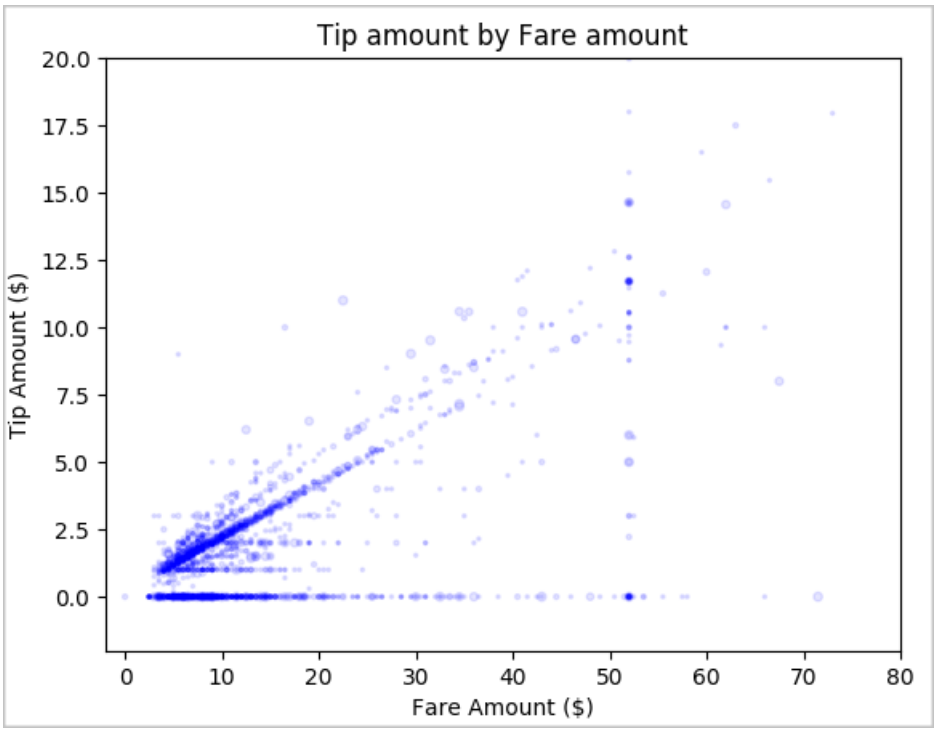
4. Another hypothesis of ours might be that there's a positive relationship between the number of passengers and the total taxi tip amount. To verify this relationship, run the following code to generate a box plot that illustrates the distribution of tips for each passenger count.

```
# How many passengers tipped by various amounts
ax2 = sampled_taxi_pd_df.boxplot(column=['tipAmount'], by=['passengerCount'])
ax2.set_title('Tip amount by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
ax2.set_ylim(0,30)
plt.suptitle('')
plt.show()
```



5. Last, we want to understand the relationship between the fare amount and the tip amount. Based on the results, we can see that there are several observations where people don't tip. However, we also see a positive relationship between the overall fare and tip amounts.

```
# Look at the relationship between fare and tip amounts

ax = sampled_taxi_pd_df.plot(kind='scatter', x= 'fareAmount', y = 'tipAmount', c='blue', alpha =
0.10, s=2.5*(sampled_taxi_pd_df['passengerCount']))
ax.set_title('Tip amount by Fare amount')
ax.set_xlabel('Fare Amount ($)')
ax.set_ylabel('Tip Amount ($)')
plt.axis([-2, 80, -2, 20])
plt.suptitle('')
plt.show()
```



Shut down the Spark instance

After you finish running the application, shut down the notebook to release the resources. Either close the tab or select **End Session** from the status panel at the bottom of the notebook.

See also

- Overview: Apache Spark on Azure Synapse Analytics
- Build a machine learning model with Apache SparkML

Next steps

- Azure Synapse Analytics
- Apache Spark official documentation

Tutorial: Create a Power BI report using Apache Spark and Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

Organizations often need to process large volumes of data before serving to key business stakeholders. In this tutorial, you will learn how to leverage the integrated experiences in Azure Synapse Analytics to process data using Apache Spark and later serve the data to end-users through Power BI and Serverless SQL.

Before you begin

- Azure Synapse Analytics workspace with an ADLS Gen2 storage account configured as the default storage.
- Power BI workspace and Power BI Desktop to visualize data. For details, see create a Power BI workspace and install Power BI desktop
- Linked service to connect your Azure Synapse Analytics and Power BI workspaces. For details, see link to a Power BI workspace
- Serverless Apache Spark pool in your Synapse Analytics workspace. For details, see create a serverless Apache Spark pool

Download and prepare the data

In this example, you will use Apache Spark to perform some analysis on taxi trip tip data from New York. The data is available through Azure Open Datasets. This subset of the dataset contains information about yellow taxi trips, including information about each trip, the start and end time and locations, the cost, and other interesting attributes.

1. Run the following lines to create a Spark dataframe by pasting the code into a new cell. This retrieves the data via the Open Datasets API. Pulling all of this data generates about 1.5 billion rows. The following code example uses start_date and end_date to apply a filter that returns a single month of data.

```
from azureml.opendatasets import NycTlcYellow
from dateutil import parser
from datetime import datetime

end_date = parser.parse('2018-06-06')
start_date = parser.parse('2018-05-01')
nyc_tlc = NycTlcYellow(start_date=start_date, end_date=end_date)
filtered_df = nyc_tlc.to_spark_dataframe()
```

2. Using Apache Spark SQL, we will create a database called NycTlcTutorial. We will use this database to store the results of our data processing.

```
%%pyspark
 spark.sql("CREATE DATABASE IF NOT EXISTS NycTlcTutorial")
```

3. Next, we will use Spark dataframe operations to process the data. In the following code, we perform the following transformations:

   a. The removal of columns which are not needed.
   b. The removal of outliers/incorrect values through filtering.

c. The creation of new features like `tripTimeSecs` and `tipped` for additional analysis.

```
from pyspark.sql.functions import unix_timestamp, date_format, col, when

taxi_df = filtered_df.select('totalAmount', 'fareAmount', 'tipAmount', 'paymentType', 'rateCodeId',
'passengerCount'\
                                , 'tripDistance', 'tpepPickupDateTime', 'tpepDropoffDateTime'\
                                , date_format('tpepPickupDateTime', 'hh').alias('pickupHour')\
                                , date_format('tpepPickupDateTime', 'EEEE').alias('weekdayString')\
                                , (unix_timestamp(col('tpepDropoffDateTime')) -
unix_timestamp(col('tpepPickupDateTime'))).alias('tripTimeSecs')\
                                , (when(col('tipAmount') > 0, 1).otherwise(0)).alias('tipped')
                                )\
                        .filter((filtered_df.passengerCount > 0) & (filtered_df.passengerCount < 8)\
                                & (filtered_df.tipAmount >= 0) & (filtered_df.tipAmount <= 25)\
                                & (filtered_df.fareAmount >= 1) & (filtered_df.fareAmount <= 250)\
                                & (filtered_df.tipAmount < filtered_df.fareAmount)\
                                & (filtered_df.tripDistance > 0) & (filtered_df.tripDistance <= 100)\
                                & (filtered_df.rateCodeId <= 5)
                                & (filtered_df.paymentType.isin({"1", "2"})))
```

4. Finally, we will save our dataframe using the Apache Spark `saveAsTable` method. This will allow you to later query and connect to the same table using serverless SQL pools.

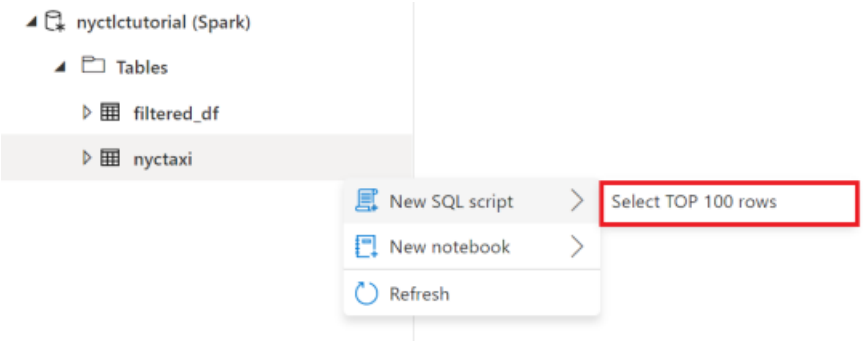
```
taxi_df.write.mode("overwrite").saveAsTable("NycTlcTutorial.nyctaxi")
```

Query data using serverless SQL pools

Azure Synapse Analytics allows the different workspace computational engines to share databases and tables between its serverless Apache Spark pools and serverless SQL pool. This is powered through the Synapse shared metadata management capability. As a result, the Spark created databases and their parquet-backed tables become visible in the workspace serverless SQL pool.

To query your Apache Spark table using your serverless SQL pool:

1. Once you have saved your Apache Spark table, switch over to the **data** tab.
2. Under **Workspaces**, find the Apache Spark table that you just created and select **New SQL script** and then **Select TOP 100 rows**.



3. You can continue to refine your query or even visualize your results using the SQL charting options.

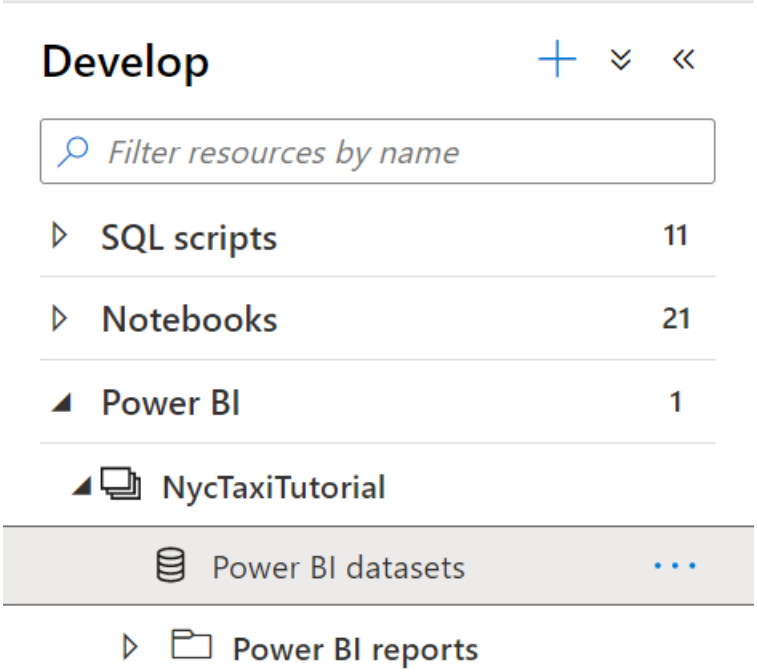
Connect to Power BI

Next, we will connect our serverless SQL pool to our Power BI workspace. Once you have connected your workspace, you will be able to create Power BI reports both directly from Azure Synapse Analytics as well as from Power BI desktop.

> **NOTE**
>
> Before you begin, you will need to set up a linked service to your Power BI workspace and download the Power BI desktop.

To connect our serverless SQL pool to our Power BI workspace:

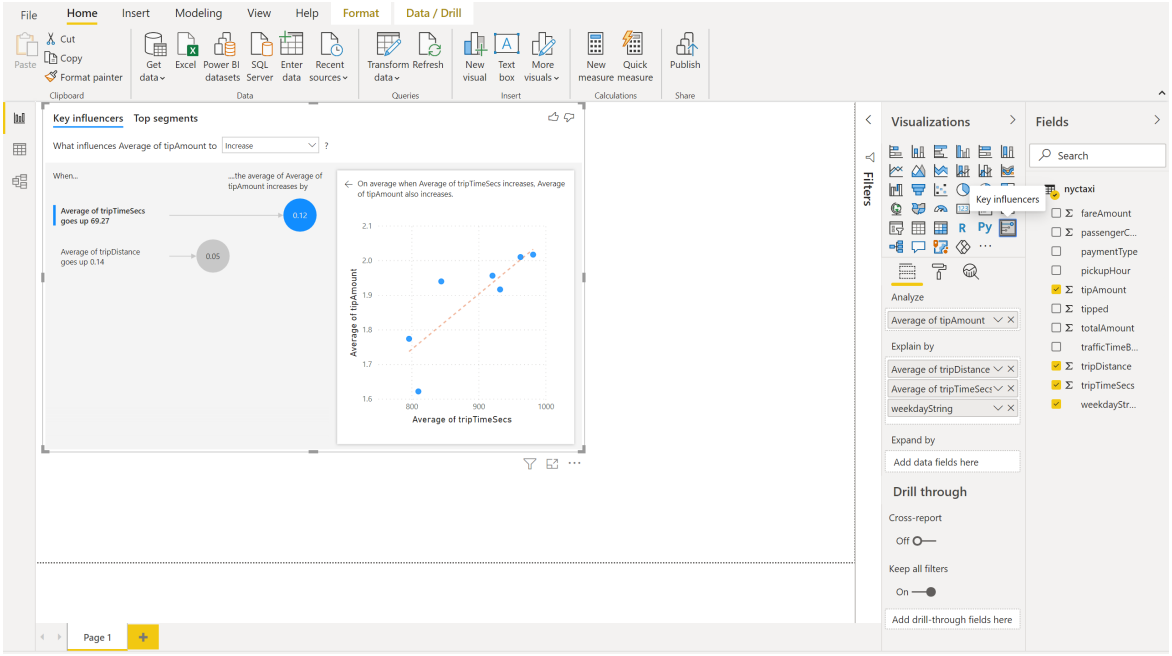
1. Navigate to the **Power BI datasets** tab and select the option to **+ New Dataset**. From the prompt, download the .pbids file from the SQL Analytics database you would like to use as a data source.



2. Open the file with Power BI Desktop to create a dataset. Once you open the file, connect to the SQL server database using the **Microsoft account** and **Import** option.

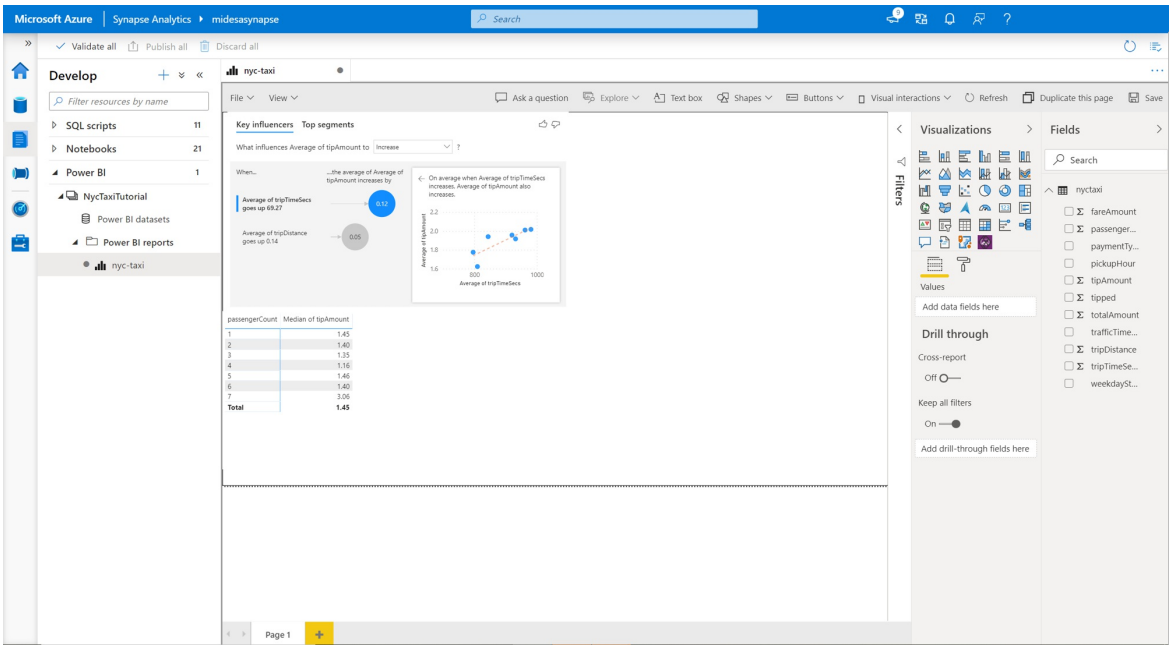
Create a Power BI report

1. From Power BI desktop, you can now add a **key influencers** chart to your report.
   a. Drag the average *tipAmount* column to **Analyze** axis.
   b. Drag the **weekdayString**, average **tripDistance**, and average **tripTimeSecs** columns to the **Explain by** axis.



2. On the Power BI desktop Home tab, select **Publish** and **Save** changes. Enter a file name and save this report to the the *NycTaxiTutorial Workspace*.

3. In addition, you can also create Power BI visualizations from within your Azure Synapse Analytics workspace. To do this, navigate to the **Develop** tab in your Azure Synapse workspace and open the Power BI tab. From here, you can select your report and continue building additional visualizations.



For more details on how to create a dataset through serverless SQL and connect to Power BI, you can visit this tutorial on connecting to Power BI desktop

Next steps

You can continue to learn more about data visualization capabilities in Azure Synapse Analytics by visiting the following documents and tutorials:

- Visualize data with serverless Apache Spark pools
- Overview of data visualization with Apache Spark pools

Tutorial: Use Pandas to read/write Azure Data Lake Storage Gen2 data in serverless Apache Spark pool in Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

Learn how to use Pandas to read/write data to Azure Data Lake Storage Gen2 (ADLS) using a serverless Apache Spark pool in Azure Synapse Analytics. Examples in this tutorial show you how to read csv data with Pandas in Synapse, as well as excel and parquet files.

In this tutorial, you'll learn how to:

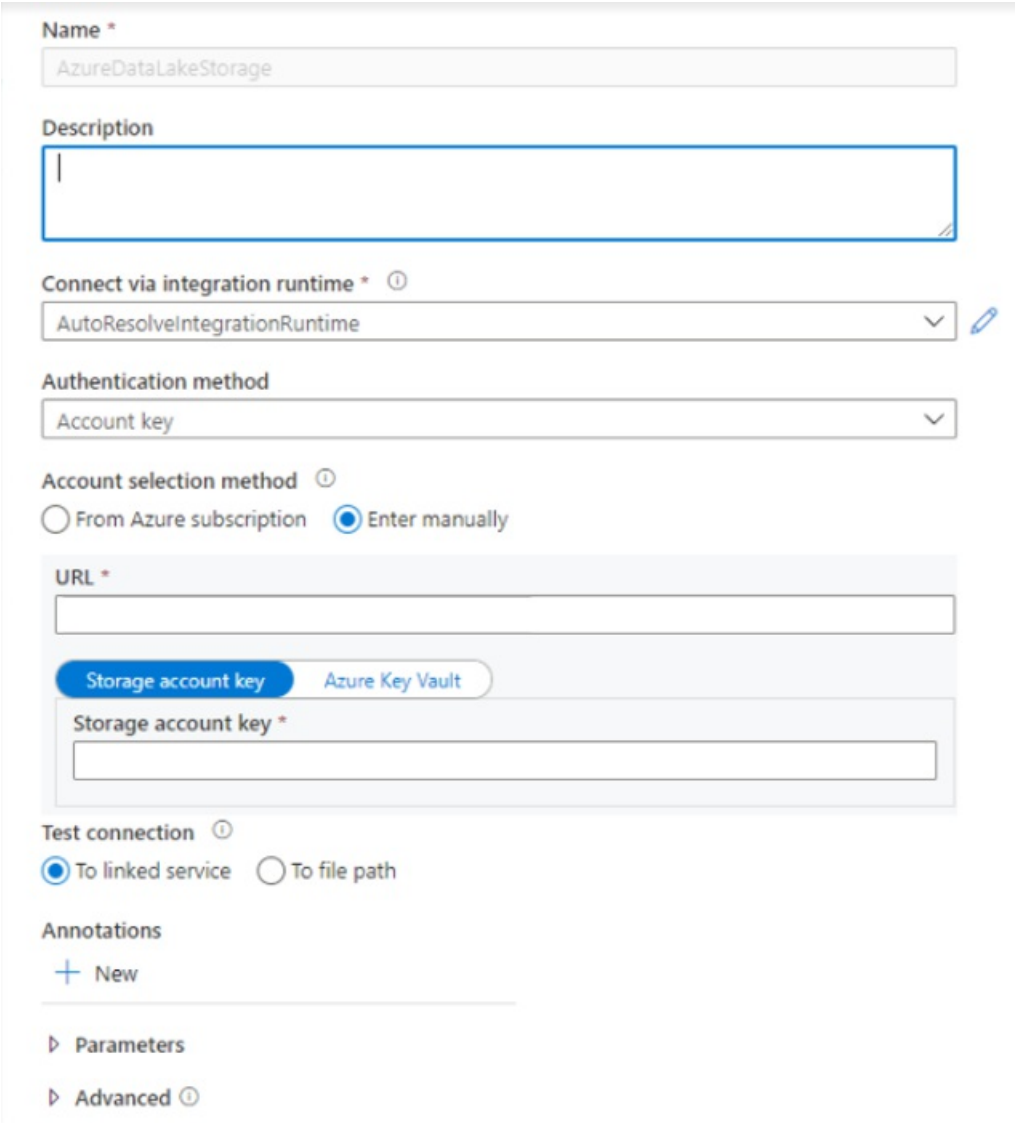
- Read/write ADLS Gen2 data using Pandas in a Spark session.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage (or primary storage). You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Serverless Apache Spark pool in your Azure Synapse Analytics workspace. For details, see Create a Spark pool in Azure Synapse.
- Configure Secondary Azure Data Lake Storage Gen2 account (which is not default to Synapse workspace). You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Create linked services - In Azure Synapse Analytics, a linked service defines your connection information to the service. In this tutorial, you'll add an Azure Synapse Analytics and Azure Data Lake Storage Gen2 linked service.

  1. Open the Azure Synapse Studio and select the **Manage** tab.
  2. Under **External connections**, select **Linked services**.
  3. To add a linked service, select **New**.
  4. Select the Azure Data Lake Storage Gen2 tile from the list and select **Continue**.
  5. Enter your authentication credentials. Account key, service principal (SP), Credentials and Manged service identity (MSI) are currently supported authentication types. Make sure that *Storage Blob Data Contributor* is assigned on storage for SP and MSI before you choose it for authentication. **Test connection** to verify your credentials are correct. Select **Create**.



NOTE

- Pandas feature is supported on **Python 3.8** and **Spark3** serverless Apache Spark pool in Azure Synapse Analytics.
- Support available for following versions: **pandas 1.2.3, fsspec 2021.10.0, adlfs 0.7.7**
- Have capabilities to support both **Azure Data Lake Storage Gen2 URI** (abfs[s]://file_system_name@account_name.dfs.core.windows.net/file_path) and **FSSPEC short URL** (abfs[s]://container_name/file_path).

Sign in to the Azure portal

Sign in to the Azure portal.

Read/Write data to default ADLS storage account of Synapse workspace

Pandas can read/write ADLS data by specifying the file path directly.

Run the following code.

NOTE

Update the file URL in this script before running it.

```
#Read data file from URI of default Azure Data Lake Storage Gen2

import pandas

#read csv file
df = pandas.read_csv('abfs[s]://file_system_name@account_name.dfs.core.windows.net/file_path')
print(df)

#write csv file
data = pandas.DataFrame({'Name':['A', 'B', 'C', 'D'], 'ID':[20, 21, 19, 18]})
data.to_csv('abfs[s]://file_system_name@account_name.dfs.core.windows.net/file_path')
```

```
#Read data file from FSSPEC short URL of default Azure Data Lake Storage Gen2

import pandas

#read csv file
df = pandas.read_csv('abfs[s]://container_name/file_path')
print(df)

#write csv file
data = pandas.DataFrame({'Name':['A', 'B', 'C', 'D'], 'ID':[20, 21, 19, 18]})
data.to_csv('abfs[s]://container_name/file_path')
```

Read/Write data using secondary ADLS account

Pandas can read/write secondary ADLS account data:

- using linked service (with authentication options - storage account key, service principal, manages service identity and credentials).
- using storage options to directly pass client ID & Secret, SAS key, storage account key and connection string.

Using linked service

Run the following code.

> **NOTE**
> Update the file URL and linked service name in this script before running it.

```
#Read data file from URI of secondary Azure Data Lake Storage Gen2

import pandas

#read data file
df = pandas.read_csv('abfs[s]://file_system_name@account_name.dfs.core.windows.net/   file_path',
storage_options = {'linked_service' : 'linked_service_name'})
print(df)

#write data file
data = pandas.DataFrame({'Name':['A', 'B', 'C', 'D'], 'ID':[20, 21, 19, 18]})
data.to_csv('abfs[s]://file_system_name@account_name.dfs.core.windows.net/file_path',    storage_options
= {'linked_service' : 'linked_service_name'})
```

```
   #Read data file from FSSPEC short URL of default Azure Data Lake Storage Gen2

   import pandas

   #read data file
   df = pandas.read_csv('abfs[s]://container_name/file_path', storage_options =       {'linked_service' :
'linked_service_name'})
   print(df)

   #write data file
   data = pandas.DataFrame({'Name':['A', 'B', 'C', 'D'], 'ID':[20, 21, 19, 18]})
   data.to_csv('abfs[s]://container_name/file_path', storage_options = {'linked_service' :
'linked_service_name'})
```

Using storage options to directly pass client ID & Secret, SAS key, storage account key, and connection string.

Run the following code.

> **NOTE**
>
> Update the file URL and storage_options in this script before running it.

```
   #Read data file from URI of secondary Azure Data Lake Storage Gen2

   import pandas

   #read data file
   df = pandas.read_csv('abfs[s]://file_system_name@account_name.dfs.core.windows.net/   file_path',
storage_options = {'account_key' : 'account_key_value'})

   ## or storage_options = {'sas_token' : 'sas_token_value'}
   ## or storage_options = {'connection_string' : 'connection_string_value'}
   ## or storage_options = {'tenant_id': 'tenant_id_value', 'client_id' : 'client_id_value',
'client_secret': 'client_secret_value'}

   print(df)

   #write data file
   data = pandas.DataFrame({'Name':['A', 'B', 'C', 'D'], 'ID':[20, 21, 19, 18]})
   data.to_csv('abfs[s]://file_system_name@account_name.dfs.core.windows.net/file_path',    storage_options
= {'account_key' : 'account_key_value'})

   ## or storage_options = {'sas_token' : 'sas_token_value'}
   ## or storage_options = {'connection_string' : 'connection_string_value'}
   ## or storage_options = {'tenant_id': 'tenant_id_value', 'client_id' : 'client_id_value',
'client_secret': 'client_secret_value'}
```

```
   #Read data file from FSSPEC short URL of default Azure Data Lake Storage Gen2

   import pandas

   #read data file
   df = pandas.read_csv('abfs[s]://container_name/file_path', storage_options = {'account_key' :
'account_key_value'})

   ## or storage_options = {'sas_token' : 'sas_token_value'}
   ## or storage_options = {'connection_string' : 'connection_string_value'}
   ## or storage_options = {'tenant_id': 'tenant_id_value', 'client_id' : 'client_id_value',
'client_secret': 'client_secret_value'}

   print(df)

   #write data file
   data = pandas.DataFrame({'Name':['A', 'B', 'C', 'D'], 'ID':[20, 21, 19, 18]})
   data.to_csv('abfs[s]://container_name/file_path', storage_options = {'account_key' :
'account_key_value'})

   ## or storage_options = {'sas_token' : 'sas_token_value'}
   ## or storage_options = {'connection_string' : 'connection_string_value'}
   ## or storage_options = {'tenant_id': 'tenant_id_value', 'client_id' : 'client_id_value',
'client_secret': 'client_secret_value'}
```

Example to read/write parquet file

Run the following code.

> **NOTE**
>
> Update the file URL in this script before running it.

```
   import pandas

   #read parquet file
   df = pandas.read_parquet('abfs[s]://file_system_name@account_name.dfs.core.windows.net/
parquet_file_path')
   print(df)

   #write parquet file
   df.to_parquet('abfs[s]://file_system_name@account_name.dfs.core.windows.net/   parquet_file_path')
```

Example to read/write excel file

Run the following code.

> **NOTE**
>
> Update the file URL in this script before running it.

```
import pandas

#read excel file
df = pandas.read_excel('abfs[s]://file_system_name@account_name.dfs.core.windows.net/   excel_file_path')
print(df)

#write excel file
df.to_excel('abfs[s]://file_system_name@account_name.dfs.core.windows.net/excel_file_path')
```

Next steps

- Azure Synapse Analytics

Tutorial: Use FSSPEC to read/write ADLS data in serverless Apache Spark pool in Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

Learn how to use Filesystem Spec (FSSPEC) to read/write data to Azure Data Lake Storage (ADLS) using a linked service in a serverless Apache Spark pool in Azure Synapse Analytics.

In this tutorial, you'll learn how to:

- Read/write ADLS data in a dedicated Spark session.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Spark pool in your Azure Synapse Analytics workspace. For details, see Create a Spark pool in Azure Synapse.

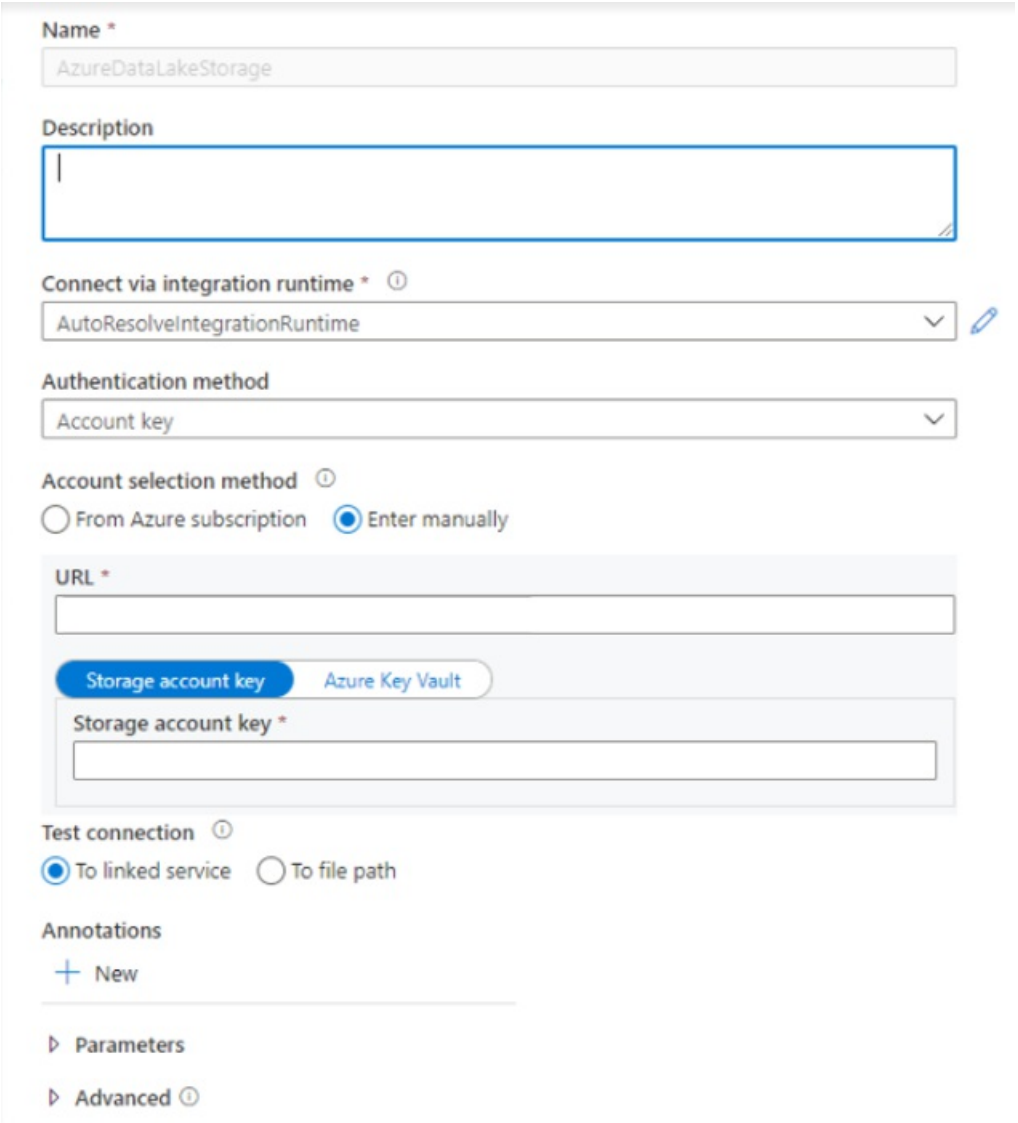
Sign in to the Azure portal

Sign in to the Azure portal.

Create linked services

In Azure Synapse Analytics, a linked service is where you define your connection information to other services. In this section, you'll add an Azure Synapse Analytics and Azure Data Lake Storage Gen2 linked service.

1. Open the Azure Synapse Studio and select the **Manage** tab.
2. Under **External connections**, select **Linked services**.
3. To add a linked service, select **New**.
4. Select the Azure Data Lake Storage Gen2 tile from the list and select **Continue**.
5. Enter your authentication credentials. Account key is currently supported authentication type. Select **Test connection** to verify your credentials are correct. Select **Create**.



Read/Write data using storage account name and key

FSSPEC can read/write ADLS data by specifying the storage account name and key directly.

1. In Synapse studio, open **Data** > **Linked** > **Azure Data Lake Storage Gen2**. Upload data to the default storage account.

2. Run the following code.

> **NOTE**
>
> Update the file URL, ADLS Gen2 storage name and key in this script before running it.

```
# To read data
import fsspec
import pandas

adls_account_name = '' #Provide exact ADLS account name
adls_account_key = '' #Provide exact ADLS account key

fsspec_handle = fsspec.open('abfs[s]://<container>/<path-to-file>', account_name=adls_account_name,
account_key=adls_account_key)

with fsspec_handle.open() as f:
    df = pandas.read_csv(f)

# To write data
import fsspec
import pandas

adls_account_name = '' #Provide exact ADLS account name
adls_account_key = '' #Provide exact ADLS account key

data = pandas.DataFrame({'Name':['Tom', 'nick', 'krish', 'jack'], 'Age':[20, 21, 19, 18]})

fsspec_handle = fsspec.open('abfs[s]://<container>/<path-to-file>', account_name=adls_account_name,
account_key=adls_account_key, mode="wt")

with fsspec_handle.open() as f:
 data.to_csv(f)
```

Read/Write data using linked service

FSSPEC can read/write ADLS data by specifying the linked service name.

1. In Synapse studio, open **Data** > **Linked** > **Azure Data Lake Storage Gen2**. Upload data to the default storage account.
2. Run the following code.

> **NOTE**
> Update the file URL, Linked Service Name and ADLS Gen2 storage name in this script before running it.

```
# To read data
import fsspec
import pandas

adls_account_name = '' #Provide exact ADLS account name
sas_key = TokenLibrary.getConnectionString(<LinkedServiceName>)

fsspec_handle = fsspec.open('abfs[s]://<container>/<path-to-file>', account_name =
adls_account_name, sas_token=sas_key)

with fsspec_handle.open() as f:
    df = pandas.read_csv(f)

# To write data
import fsspec
import pandas

adls_account_name = '' #Provide exact ADLS account name

data = pandas.DataFrame({'Name':['Tom', 'nick', 'krish', 'jack'], 'Age':[20, 21, 19, 18]})
sas_key = TokenLibrary.getConnectionString(<LinkedServiceName>)

fsspec_handle = fsspec.open('abfs[s]://<container>/<path-to-file>', account_name =
adls_account_name, sas_token=sas_key, mode="wt")

with fsspec_handle.open() as f:
    data.to_csv(f)
```

Upload file from local file system to default ADLS storage account of Synapse workspace

FSSPEC can upload a file from the local file system to a Synapse workspace default ADLS storage account.

Run the following code.

> **NOTE**
> Update the file URL in this script before running it.

```
# Import libraries
import fsspec
import os

# Set variables
local_file_name = "<local_file_name>"
ADLS_Store_Path = "abfs[s]://<filesystemname>@<account name>.dfs.windows.cor.net/"+local_file_name

# Generate local file for testing
with open(local_file_name, mode='w') as f:
    for i in range(1000):
        f.write("Testing local file functionality\n")
print("Created: " + local_file_name)

# Upload local file to ADLS
fs = fsspec.filesystem('abfs[s]')
fs.upload(local_file_name, ADLS_Store_Path)
```

Next steps

- Pandas in Synapse Analytics using serverless Apache Spark pool

- Azure Synapse Analytics
- FSSPEC official documentation

Tutorial: Build a machine learning app with Apache Spark MLlib and Azure Synapse Analytics

8/19/2022 • 8 minutes to read • Edit Online

In this article, you'll learn how to use Apache Spark MLlib to create a machine learning application that does simple predictive analysis on an Azure open dataset. Spark provides built-in machine learning libraries. This example uses *classification* through logistic regression.

SparkML and MLlib are core Spark libraries that provide many utilities that are useful for machine learning tasks, including utilities that are suitable for:

- Classification
- Regression
- Clustering
- Topic modeling
- Singular value decomposition (SVD) and principal component analysis (PCA)
- Hypothesis testing and calculating sample statistics

Understand classification and logistic regression

Classification, a popular machine learning task, is the process of sorting input data into categories. It's the job of a classification algorithm to figure out how to assign *labels* to input data that you provide. For example, you can think of a machine learning algorithm that accepts stock information as input and divide the stock into two categories: stocks that you should sell and stocks that you should keep.

Logistic regression is an algorithm that you can use for classification. Spark's logistic regression API is useful for *binary classification*, or classifying input data into one of two groups. For more information about logistic regression, see Wikipedia.

In summary, the process of logistic regression produces a *logistic function* that you can use to predict the probability that an input vector belongs in one group or the other.

Predictive analysis example on NYC taxi data

In this example, you use Spark to perform some predictive analysis on taxi-trip tip data from New York. The data is available through Azure Open Datasets. This subset of the dataset contains information about yellow taxi trips, including information about each trip, the start and end time and locations, the cost, and other interesting attributes.

> **IMPORTANT**
> There might be additional charges for pulling this data from its storage location.

In the following steps, you develop a model to predict whether a particular trip includes a tip or not.

Create an Apache Spark machine learning model

1. Create a notebook by using the PySpark kernel. For instructions, see Create a notebook.

2. Import the types required for this application. Copy and paste the following code into an empty cell, and

then press Shift+Enter. Or run the cell by using the blue play icon to the left of the code.

```
import matplotlib.pyplot as plt
from datetime import datetime
from dateutil import parser
from pyspark.sql.functions import unix_timestamp, date_format, col, when
from pyspark.ml import Pipeline
from pyspark.ml import PipelineModel
from pyspark.ml.feature import RFormula
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorIndexer
from pyspark.ml.classification import LogisticRegression
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

Because of the PySpark kernel, you don't need to create any contexts explicitly. The Spark context is automatically created for you when you run the first code cell.

Construct the input DataFrame

Because the raw data is in a Parquet format, you can use the Spark context to pull the file into memory as a DataFrame directly. Although the code in the following steps uses the default options, it's possible to force mapping of data types and other schema attributes if needed.

1. Run the following lines to create a Spark DataFrame by pasting the code into a new cell. This step retrieves the data via the Open Datasets API. Pulling all of this data generates about 1.5 billion rows.

   Depending on the size of your serverless Apache Spark pool, the raw data might be too large or take too much time to operate on. You can filter this data down to something smaller. The following code example uses `start_date` and `end_date` to apply a filter that returns a single month of data.

   ```
   from azureml.opendatasets import NycTlcYellow

   end_date = parser.parse('2018-06-06')
   start_date = parser.parse('2018-05-01')
   nyc_tlc = NycTlcYellow(start_date=start_date, end_date=end_date)
   filtered_df = nyc_tlc.to_spark_dataframe()
   ```

2. The downside to simple filtering is that, from a statistical perspective, it might introduce bias into the data. Another approach is to use the sampling built into Spark.

   The following code reduces the dataset to about 2,000 rows, if it's applied after the preceding code. You can use this sampling step instead of the simple filter or in conjunction with the simple filter.

   ```
   # To make development easier, faster, and less expensive, downsample for now
   sampled_taxi_df = filtered_df.sample(True, 0.001, seed=1234)
   ```

3. It's now possible to look at the data to see what was read. It's normally better to review data with a subset rather than the full set, depending on the size of the dataset.

   The following code offers two ways to view the data. The first way is basic. The second way provides a much richer grid experience, along with the capability to visualize the data graphically.

   ```
   #sampled_taxi_df.show(5)
   display(sampled_taxi_df)
   ```

4. Depending on the size of the generated dataset, and your need to experiment or run the notebook many

times, you might want to cache the dataset locally in the workspace. There are three ways to perform explicit caching:

- Save the DataFrame locally as a file.
- Save the DataFrame as a temporary table or view.
- Save the DataFrame as a permanent table.

The first two of these approaches are included in the following code examples.

Creating a temporary table or view provides different access paths to the data, but it lasts only for the duration of the Spark instance session.

```
sampled_taxi_df.createOrReplaceTempView("nytaxi")
```

Prepare the data

The data in its raw form is often not suitable for passing directly to a model. You must perform a series of actions on the data to get it into a state where the model can consume it.

In the following code, you perform four classes of operations:

- The removal of outliers or incorrect values through filtering.
- The removal of columns, which are not needed.
- The creation of new columns derived from the raw data to make the model work more effectively. This operation is sometimes called featurization.
- Labeling. Because you're undertaking binary classification (will there be a tip or not on a given trip), there's a need to convert the tip amount into a 0 or 1 value.

```
taxi_df = sampled_taxi_df.select('totalAmount', 'fareAmount', 'tipAmount', 'paymentType', 'rateCodeId',
'passengerCount'\
                                , 'tripDistance', 'tpepPickupDateTime', 'tpepDropoffDateTime'\
                                , date_format('tpepPickupDateTime', 'hh').alias('pickupHour')\
                                , date_format('tpepPickupDateTime', 'EEEE').alias('weekdayString')\
                                , (unix_timestamp(col('tpepDropoffDateTime')) -
unix_timestamp(col('tpepPickupDateTime'))).alias('tripTimeSecs')\
                                , (when(col('tipAmount') > 0, 1).otherwise(0)).alias('tipped')
                                )\
                        .filter((sampled_taxi_df.passengerCount > 0) & (sampled_taxi_df.passengerCount < 8)\
                                & (sampled_taxi_df.tipAmount >= 0) & (sampled_taxi_df.tipAmount <= 25)\
                                & (sampled_taxi_df.fareAmount >= 1) & (sampled_taxi_df.fareAmount <= 250)\
                                & (sampled_taxi_df.tipAmount < sampled_taxi_df.fareAmount)\
                                & (sampled_taxi_df.tripDistance > 0) & (sampled_taxi_df.tripDistance <=
100)\
                                & (sampled_taxi_df.rateCodeId <= 5)
                                & (sampled_taxi_df.paymentType.isin({"1", "2"}))
                                )
```

You then make a second pass over the data to add the final features.

```
taxi_featurised_df = taxi_df.select('totalAmount', 'fareAmount', 'tipAmount', 'paymentType',
'passengerCount'\
                                                , 'tripDistance', 'weekdayString',
'pickupHour','tripTimeSecs','tipped'\
                                                , when((taxi_df.pickupHour <= 6) | (taxi_df.pickupHour >=
20),"Night")\
                                                .when((taxi_df.pickupHour >= 7) & (taxi_df.pickupHour <=
10), "AMRush")\
                                                .when((taxi_df.pickupHour >= 11) & (taxi_df.pickupHour <=
15), "Afternoon")\
                                                .when((taxi_df.pickupHour >= 16) & (taxi_df.pickupHour <=
19), "PMRush")\
                                                .otherwise(0).alias('trafficTimeBins')
                                              )\
                                       .filter((taxi_df.tripTimeSecs >= 30) & (taxi_df.tripTimeSecs <=
7200))
```

Create a logistic regression model

The final task is to convert the labeled data into a format that can be analyzed through logistic regression. The input to a logistic regression algorithm needs to be a set of *label/feature vector pairs*, where the *feature vector* is a vector of numbers that represent the input point.

So, you need to convert the categorical columns into numbers. Specifically, you need to convert the `trafficTimeBins` and `weekdayString` columns into integer representations. There are multiple approaches to performing the conversion. The following example takes the `OneHotEncoder` approach, which is common.

```
# Because the sample uses an algorithm that works only with numeric features, convert them so they can be
consumed
sI1 = StringIndexer(inputCol="trafficTimeBins", outputCol="trafficTimeBinsIndex")
en1 = OneHotEncoder(dropLast=False, inputCol="trafficTimeBinsIndex", outputCol="trafficTimeBinsVec")
sI2 = StringIndexer(inputCol="weekdayString", outputCol="weekdayIndex")
en2 = OneHotEncoder(dropLast=False, inputCol="weekdayIndex", outputCol="weekdayVec")

# Create a new DataFrame that has had the encodings applied
encoded_final_df = Pipeline(stages=[sI1, en1, sI2,
en2]).fit(taxi_featurised_df).transform(taxi_featurised_df)
```

This action results in a new DataFrame with all columns in the right format to train a model.

Train a logistic regression model

The first task is to split the dataset into a training set and a testing or validation set. The split here is arbitrary. Experiment with different split settings to see if they affect the model.

```
# Decide on the split between training and testing data from the DataFrame
trainingFraction = 0.7
testingFraction = (1-trainingFraction)
seed = 1234

# Split the DataFrame into test and training DataFrames
train_data_df, test_data_df = encoded_final_df.randomSplit([trainingFraction, testingFraction], seed=seed)
```

Now that there are two DataFrames, the next task is to create the model formula and run it against the training DataFrame. Then you can validate against the testing DataFrame. Experiment with different versions of the model formula to see the impact of different combinations.

> **NOTE**
>
> To save the model, assign the *Storage Blob Data Contributor* role to the Azure SQL Database server resource scope. For detailed steps, see Assign Azure roles using the Azure portal. Only members with owner privileges can perform this step.

```
## Create a new logistic regression object for the model
logReg = LogisticRegression(maxIter=10, regParam=0.3, labelCol = 'tipped')

## The formula for the model
classFormula = RFormula(formula="tipped ~ pickupHour + weekdayVec + passengerCount + tripTimeSecs +
tripDistance + fareAmount + paymentType+ trafficTimeBinsVec")

## Undertake training and create a logistic regression model
lrModel = Pipeline(stages=[classFormula, logReg]).fit(train_data_df)

## Saving the model is optional, but it's another form of inter-session cache
datestamp = datetime.now().strftime('%m-%d-%Y-%s')
fileName = "lrModel_" + datestamp
logRegDirfilename = fileName
lrModel.save(logRegDirfilename)

## Predict tip 1/0 (yes/no) on the test dataset; evaluation using area under ROC
predictions = lrModel.transform(test_data_df)
predictionAndLabels = predictions.select("label","prediction").rdd
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)
```

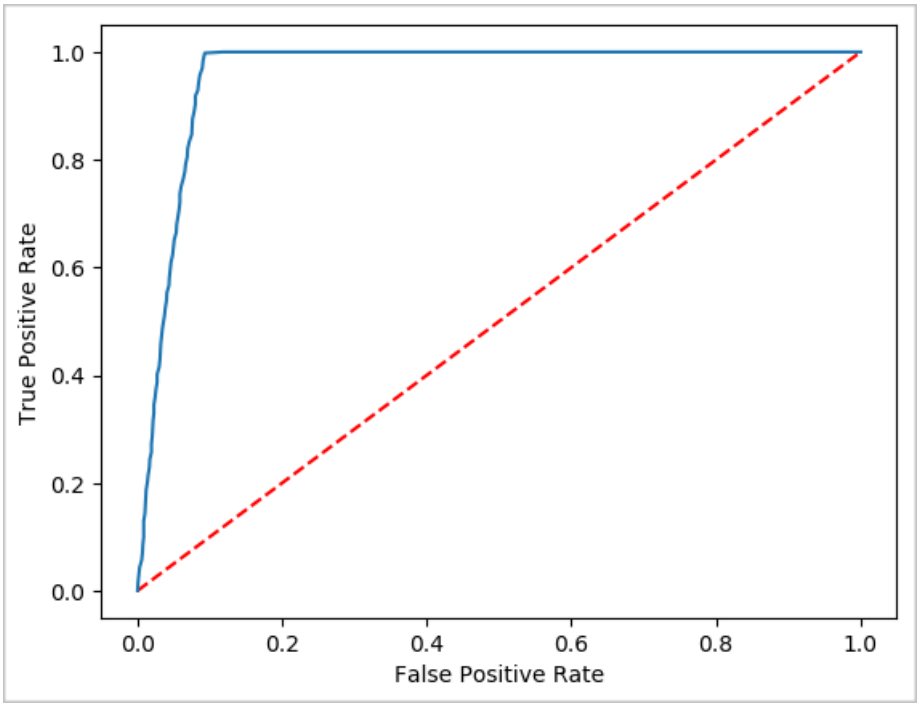
The output from this cell is:

```
Area under ROC = 0.9779470729751403
```

Create a visual representation of the prediction

You can now construct a final visualization to help you reason about the results of this test. An ROC curve is one way to review the result.

```
## Plot the ROC curve; no need for pandas, because this uses the modelSummary object
modelSummary = lrModel.stages[-1].summary

plt.plot([0, 1], [0, 1], 'r--')
plt.plot(modelSummary.roc.select('FPR').collect(),
         modelSummary.roc.select('TPR').collect())
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



Shut down the Spark instance

After you finish running the application, shut down the notebook to release the resources by closing the tab. Or select **End Session** from the status panel at the bottom of the notebook.

See also

- Overview: Apache Spark on Azure Synapse Analytics

Next steps

- .NET for Apache Spark documentation
- Azure Synapse Analytics
- Apache Spark official documentation

> **NOTE**
> Some of the official Apache Spark documentation relies on using the Spark console, which is not available on Apache Spark in Azure Synapse Analytics. Use the notebook or IntelliJ experiences instead.

Tutorial: Create Apache Spark job definition in Synapse Studio

8/19/2022 • 9 minutes to read • Edit Online

This tutorial demonstrates how to use the Synapse Studio to create Apache Spark job definitions, and then submit them to a serverless Apache Spark pool.

This tutorial covers the following tasks:

- Create an Apache Spark job definition for PySpark (Python)
- Create an Apache Spark job definition for Spark (Scala)
- Create an Apache Spark job definition for .NET Spark (C#/F#)
- Create job definition by importing a JSON file
- Exporting an Apache Spark job definition file to local
- Submit an Apache Spark job definition as a batch job
- Add an Apache Spark job definition into pipeline

Prerequisites

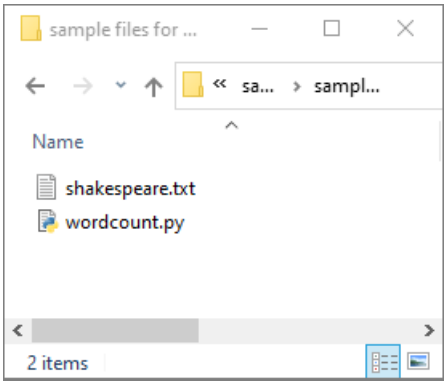
Before you start with this tutorial, make sure to meet the following requirements:

- An Azure Synapse Analytics workspace. For instructions, see Create an Azure Synapse Analytics workspace.
- A serverless Apache Spark pool.
- An ADLS Gen2 storage account. You need to be the **Storage Blob Data Contributor** of the ADLS Gen2 filesystem you want to work with. If you aren't, you need to add the permission manually.
- If you don't want to use the workspace default storage, link the required ADLS Gen2 storage account in Synapse Studio.

Create an Apache Spark job definition for PySpark (Python)

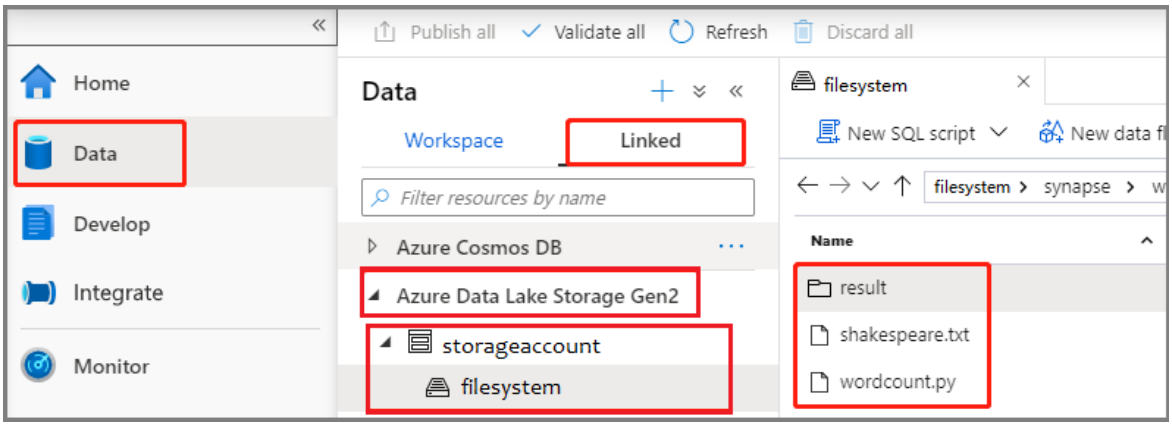
In this section, you create an Apache Spark job definition for PySpark (Python).

1. Open Synapse Studio.
2. You can go to Sample files for creating Apache Spark job definitions to download **sample files for python.zip**, then unzip the compressed package, and extract the **wordcount.py** and **shakespeare.txt** files.

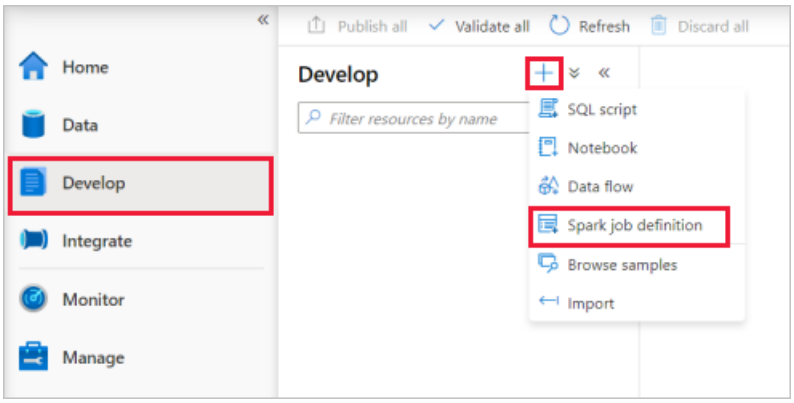


3. Select **Data** -> **Linked** -> **Azure Data Lake Storage Gen2**, and

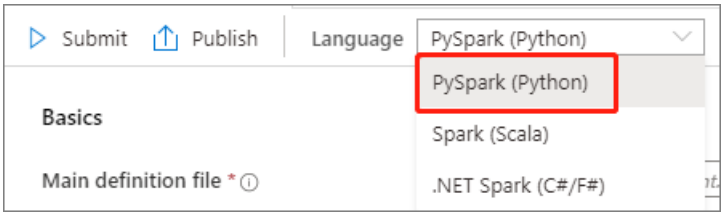
upload **wordcount.py** and **shakespeare.txt** into your ADLS Gen2 filesystem.



4. Select **Develop** hub, select the '+' icon and select **Spark job definition** to create a new Spark job definition.



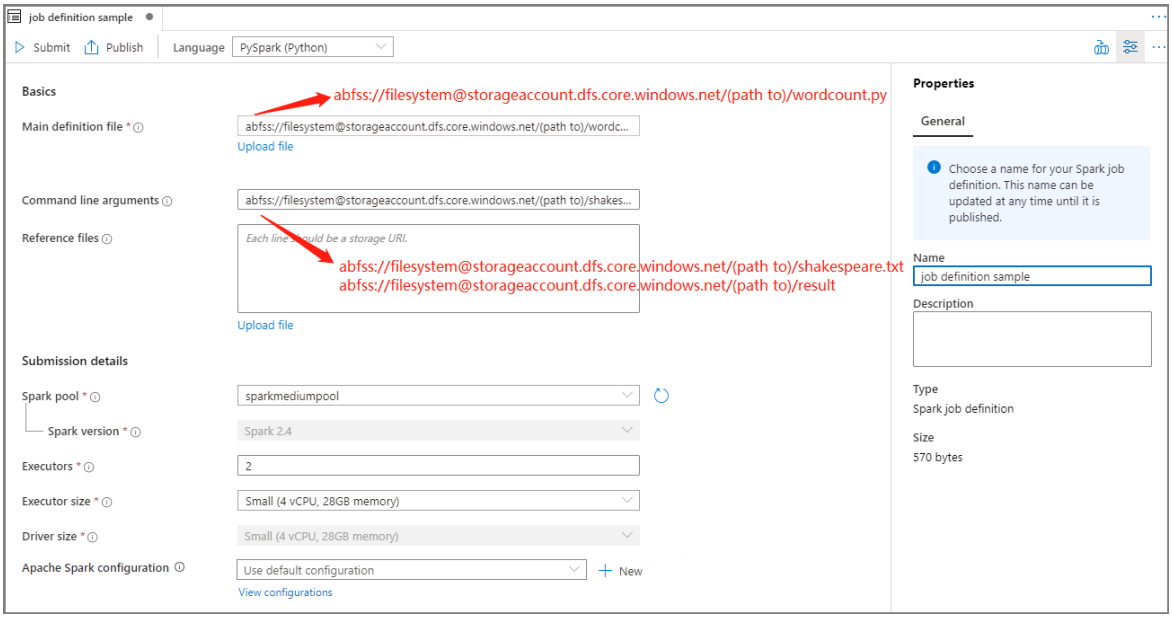
5. Select **PySpark (Python)** from the Language drop down list in the Apache Spark job definition main window.



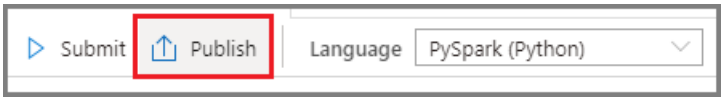
6. Fill in information for Apache Spark job definition.

PROPERTY	DESCRIPTION
Job definition name	Enter a name for your Apache Spark job definition. This name can be updated at any time until it's published. Sample: `job definition sample`
Main definition file	The main file used for the job. Select a PY file from your storage. You can select **Upload file** to upload the file to a storage account. Sample: `abfss://…/path/to/wordcount.py`
Command-line arguments	Optional arguments to the job. Sample: `abfss://…/path/to/shakespeare.txt` `abfss://…/path/to/result` *Note: Two arguments for the sample job definition are separated by a space.*

PROPERTY	DESCRIPTION
Reference files	Additional files used for reference in the main definition file. You can select **Upload file** to upload the file to a storage account.
Spark pool	The job will be submitted to the selected Apache Spark pool.
Spark version	Version of Apache Spark that the Apache Spark pool is running.
Executors	Number of executors to be given in the specified Apache Spark pool for the job.
Executor size	Number of cores and memory to be used for executors given in the specified Apache Spark pool for the job.
Driver size	Number of cores and memory to be used for driver given in the specified Apache Spark pool for the job.
Apache Spark configuration	Customize configurations by adding properties below. If you do not add a property, Azure Synapse will use the default value when applicable.



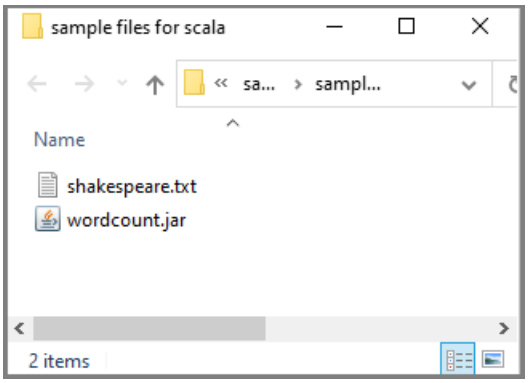
7. Select **Publish** to save the Apache Spark job definition.



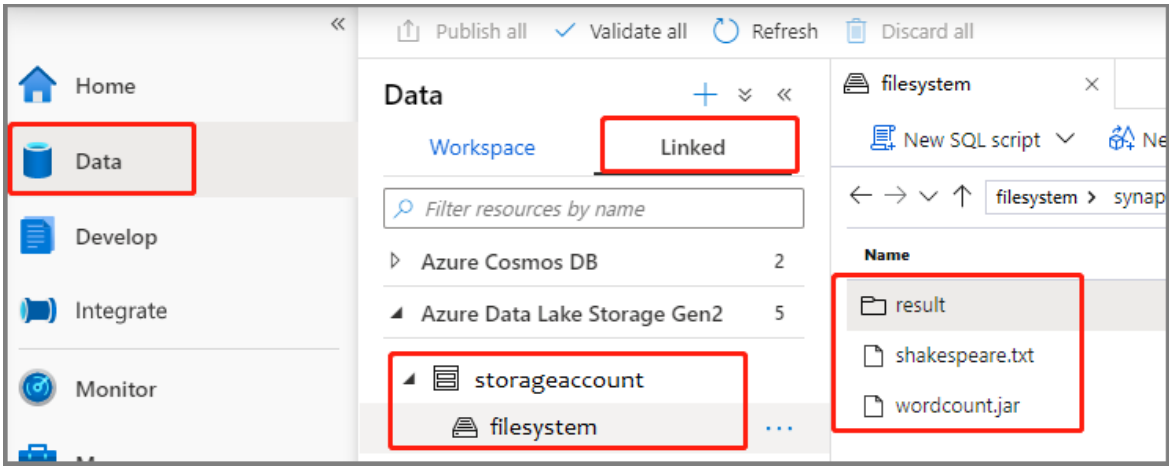
Create an Apache Spark job definition for Apache Spark(Scala)

In this section, you create an Apache Spark job definition for Apache Spark(Scala).

1. Open Azure Synapse Studio.
2. You can go to Sample files for creating Apache Spark job definitions to download **sample files for scala.zip**, then unzip the compressed package, and extract the **wordcount.jar** and **shakespeare.txt** files.

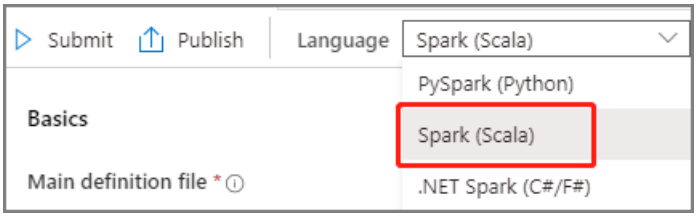


3. Select **Data** -> **Linked** -> **Azure Data Lake Storage Gen2**, and upload **wordcount.jar** and **shakespeare.txt** into your ADLS Gen2 filesystem.



4. Select **Develop** hub, select the '+' icon and select **Spark job definition** to create a new Spark job definition. (The sample image is the same as step 4 of **Create an Apache Spark job definition (Python) for PySpark**.)

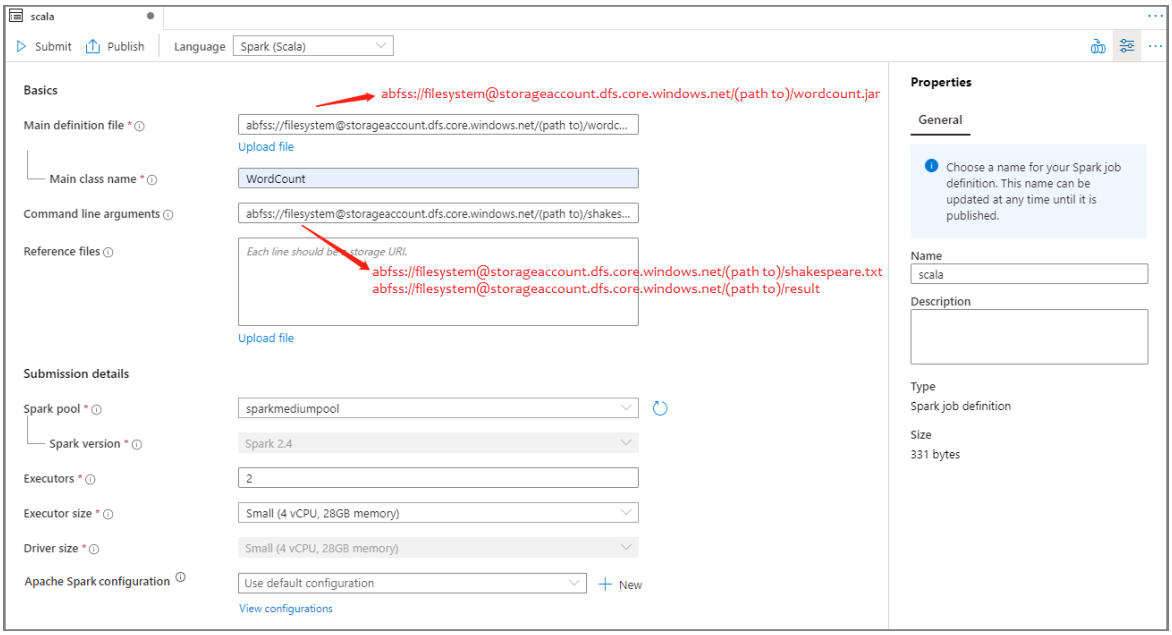
5. Select **Spark(Scala)** from the Language drop down list in the Apache Spark job definition main window.



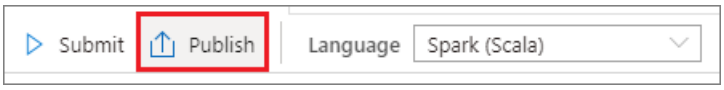
6. Fill in information for Apache Spark job definition. You can copy the sample information.

PROPERTY	DESCRIPTION
Job definition name	Enter a name for your Apache Spark job definition. This name can be updated at any time until it's published. Sample: `scala`
Main definition file	The main file used for the job. Select a JAR file from your storage. You can select **Upload file** to upload the file to a storage account. Sample: `abfss://…/path/to/wordcount.jar`
Main class name	The fully qualified identifier or the main class that is in the main definition file. Sample: `WordCount`

PROPERTY	DESCRIPTION
Command-line arguments	Optional arguments to the job. Sample: `abfss://…/path/to/shakespeare.txt` `abfss://…/path/to/result` *Note: Two arguments for the sample job definition are separated by a space.*
Reference files	Additional files used for reference in the main definition file. You can select **Upload file** to upload the file to a storage account.
Spark pool	The job will be submitted to the selected Apache Spark pool.
Spark version	Version of Apache Spark that the Apache Spark pool is running.
Executors	Number of executors to be given in the specified Apache Spark pool for the job.
Executor size	Number of cores and memory to be used for executors given in the specified Apache Spark pool for the job.
Driver size	Number of cores and memory to be used for driver given in the specified Apache Spark pool for the job.
Apache Spark configuration	Customize configurations by adding properties below. If you do not add a property, Azure Synapse will use the default value when applicable.



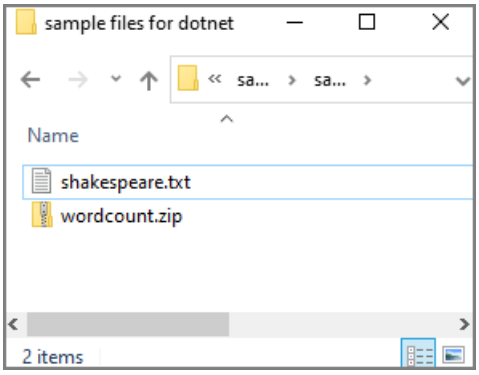
7. Select **Publish** to save the Apache Spark job definition.



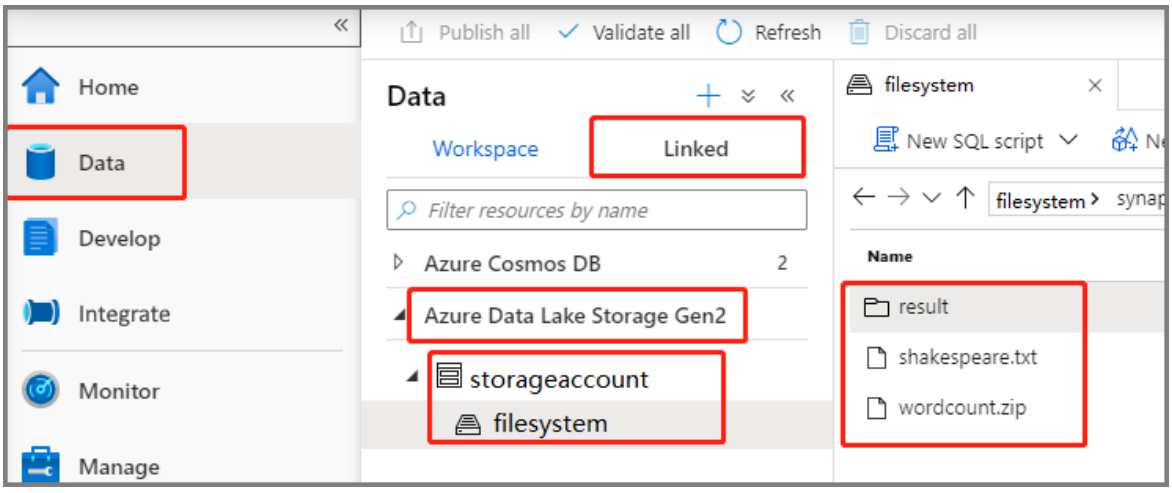
Create an Apache Spark job definition for .NET Spark(C#/F#)

In this section, you create an Apache Spark job definition for .NET Spark(C#/F#).

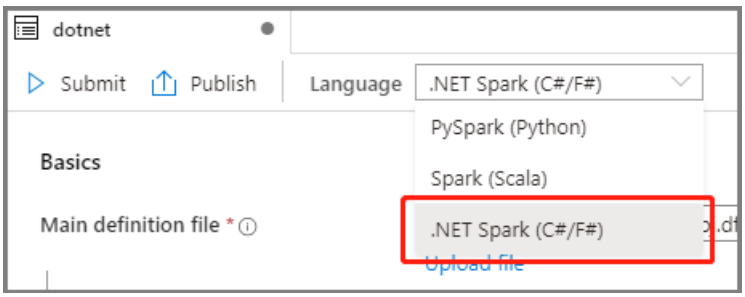
1. Open Azure Synapse Studio.
2. You can go to Sample files for creating Apache Spark job definitions to download **sample files for dotnet.zip**, then unzip the compressed package, and extract the **wordcount.zip** and **shakespeare.txt** files.



3. Select **Data** -> **Linked** -> **Azure Data Lake Storage Gen2**, and upload **wordcount.zip** and **shakespeare.txt** into your ADLS Gen2 filesystem.



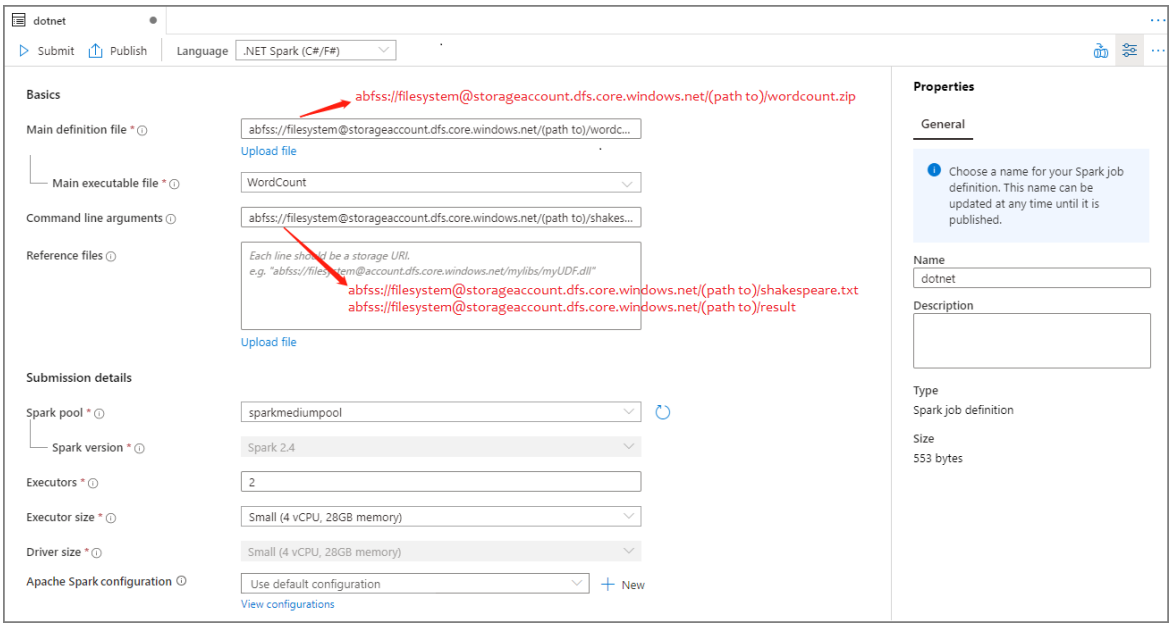
4. Select **Develop** hub, select the '+' icon and select **Spark job definition** to create a new Spark job definition. (The sample image is the same as step 4 of **Create an Apache Spark job definition (Python) for PySpark**.)
5. Select **.NET Spark(C#/F#)** from the Language drop down list in the Apache Spark Job Definition main window.



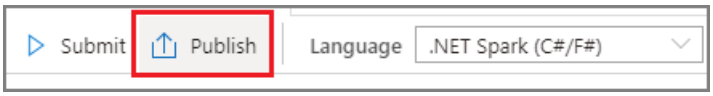
6. Fill in information for Apache Spark Job Definition. You can copy the sample information.

PROPERTY	DESCRIPTION

PROPERTY	DESCRIPTION
Job definition name	Enter a name for your Apache Spark job definition. This name can be updated at any time until it's published. Sample: `dotnet`
Main definition file	The main file used for the job. Select a ZIP file that contains your .NET for Apache Spark application (that is, the main executable file, DLLs containing user-defined functions, and other required files) from your storage. You can select **Upload file** to upload the file to a storage account. Sample: `abfss://…/path/to/wordcount.zip`
Main executable file	The main executable file in the main definition ZIP file. Sample: `WordCount`
Command-line arguments	Optional arguments to the job. Sample: `abfss://…/path/to/shakespeare.txt` `abfss://…/path/to/result` *Note: Two arguments for the sample job definition are separated by a space.*
Reference files	Additional files needed by the worker nodes for executing the .NET for Apache Spark application that isn't included in the main definition ZIP file (that is, dependent jars, additional user-defined function DLLs, and other config files). You can select **Upload file** to upload the file to a storage account.
Spark pool	The job will be submitted to the selected Apache Spark pool.
Spark version	Version of Apache Spark that the Apache Spark pool is running.
Executors	Number of executors to be given in the specified Apache Spark pool for the job.
Executor size	Number of cores and memory to be used for executors given in the specified Apache Spark pool for the job.
Driver size	Number of cores and memory to be used for driver given in the specified Apache Spark pool for the job.
Apache Spark configuration	Customize configurations by adding properties below. If you do not add a property, Azure Synapse will use the default value when applicable.



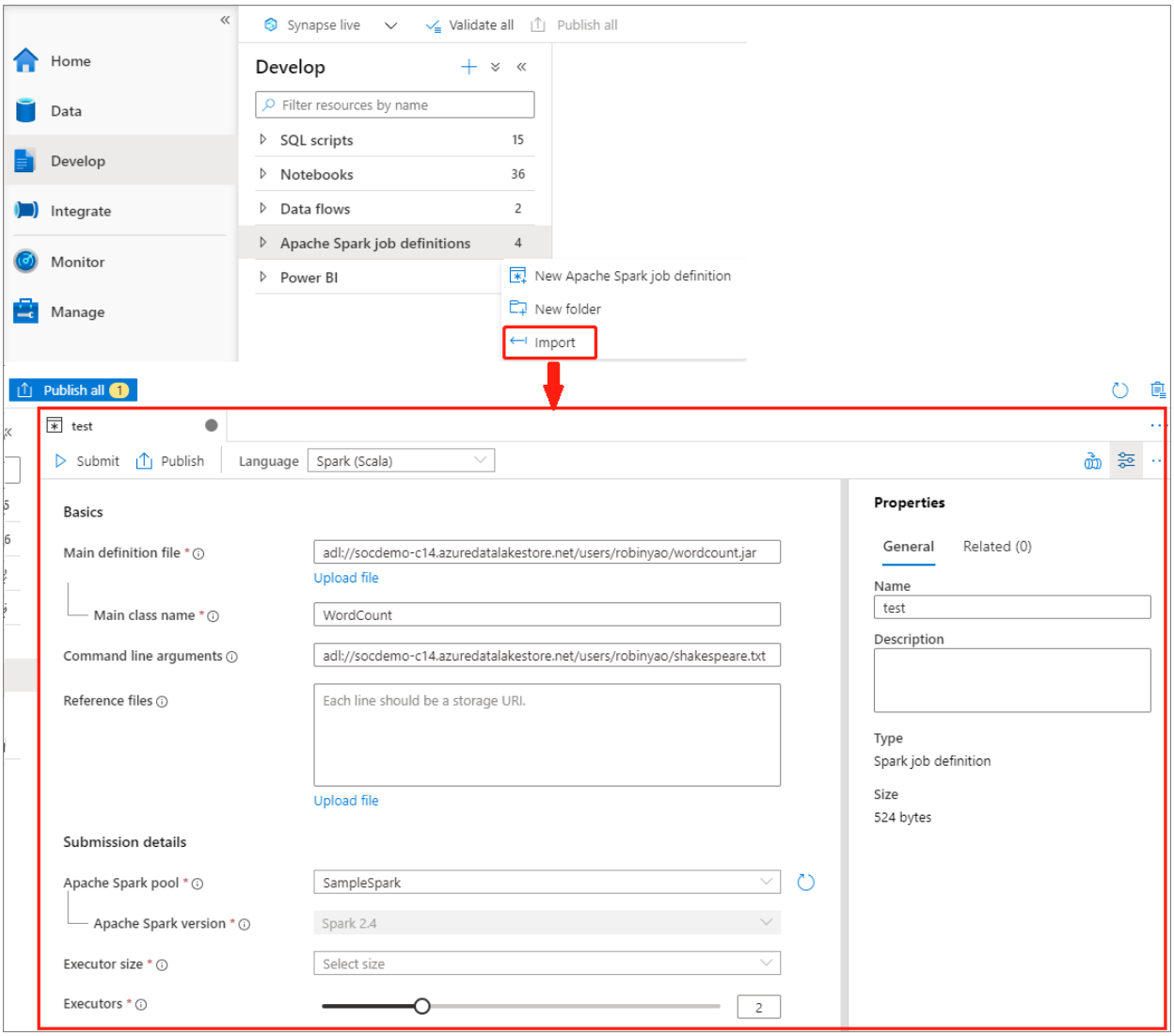
7. Select **Publish** to save the Apache Spark job definition.



> **NOTE**
>
> For Apache Spark configuration, if the Apache Spark configuration Apache Spark job definition does not do anything special, the default configuration will be used when running the job.

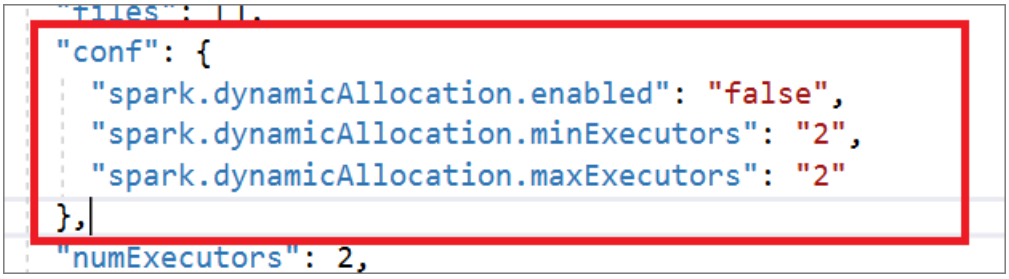
Create Apache Spark job definition by importing a JSON file

You can import an existing local JSON file into Azure Synapse workspace from the **Actions** (...) menu of the Apache Spark job definition Explorer to create a new Apache Spark job definition.



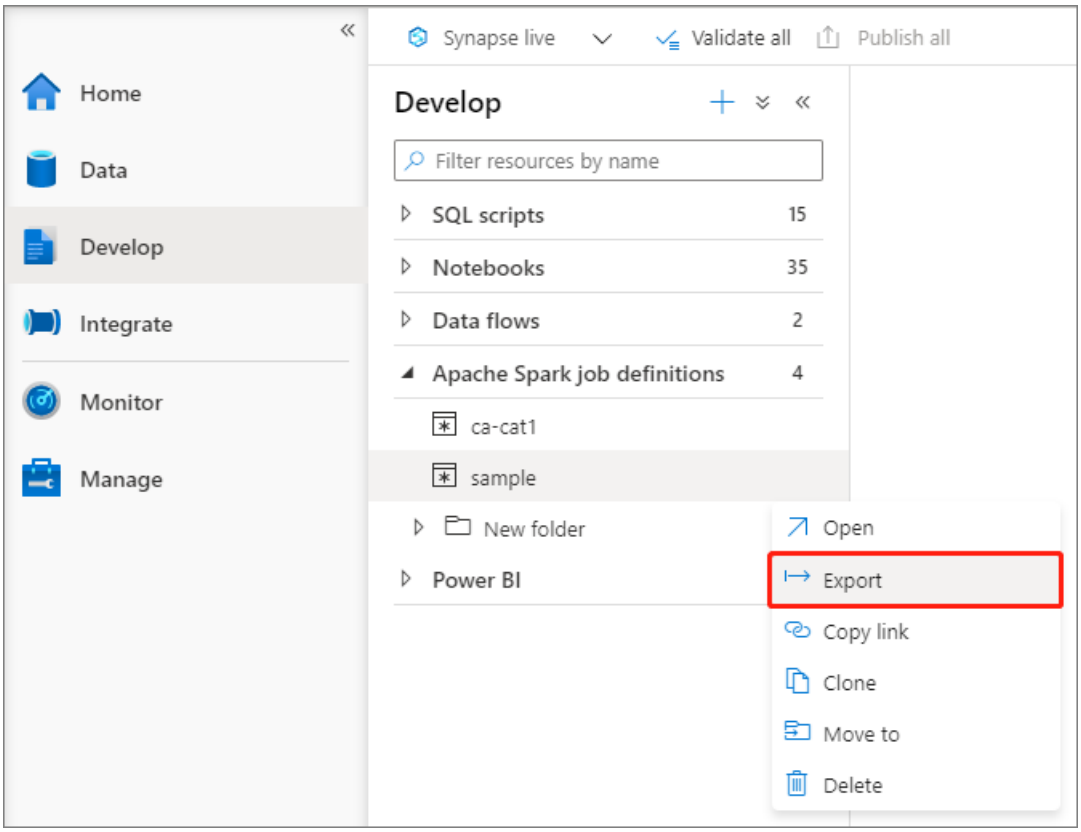
The Spark job definition is fully compatible with Livy API. You can add additional parameters for other Livy properties (Livy Docs - REST API (apache.org) in the local JSON file. You can also specify the Spark configuration related parameters in the config property as shown below. Then, you can import the JSON file back to create a new Apache Spark job definition for your batch job. Example JSON for spark definition import:

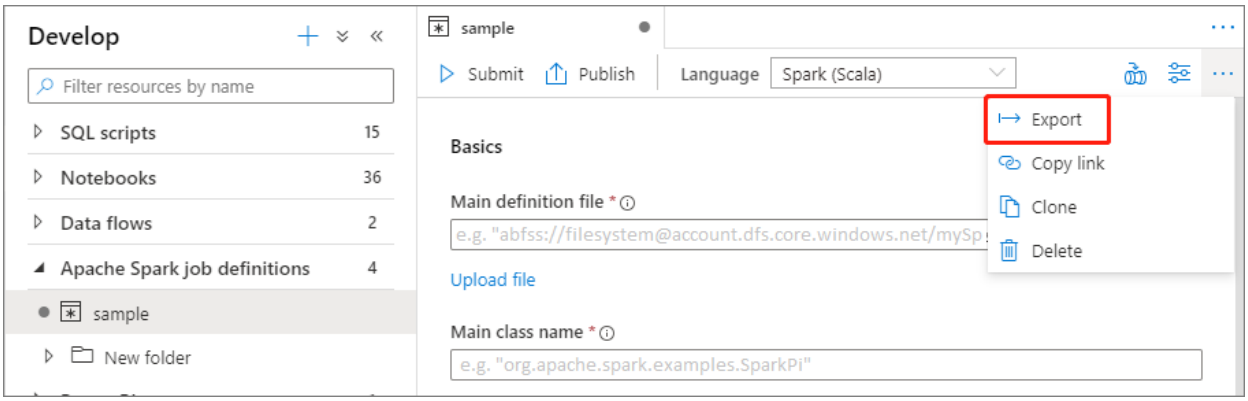
```
 {
"targetBigDataPool": {
  "referenceName": "socdemolarge",
  "type": "BigDataPoolReference"
},
"requiredSparkVersion": "2.3",
"language": "scala",
"jobProperties": {
  "name": "robinSparkDefinitiontest",
  "file": "adl://socdemo-c14.azuredatalakestore.net/users/robinyao/wordcount.jar",
  "className": "WordCount",
  "args": [
    "adl://socdemo-c14.azuredatalakestore.net/users/robinyao/shakespeare.txt"
  ],
  "jars": [],
  "files": [],
  "conf": {
    "spark.dynamicAllocation.enabled": "false",
    "spark.dynamicAllocation.minExecutors": "2",
    "spark.dynamicAllocation.maxExecutors": "2"
  },
  "numExecutors": 2,
  "executorCores": 8,
  "executorMemory": "24g",
  "driverCores": 8,
  "driverMemory": "24g"
}
}
```



Export an existing Apache Spark job definition file

You can export existing Apache Spark job definition files to local from **Actions** (...) menu of the File Explorer. You can further update the JSON file for additional Livy properties, and import it back to create new job definition if necessary.



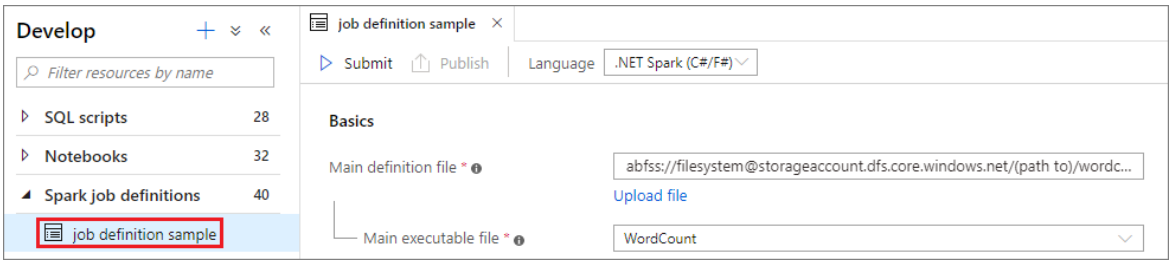


Submit an Apache Spark job definition as a batch job

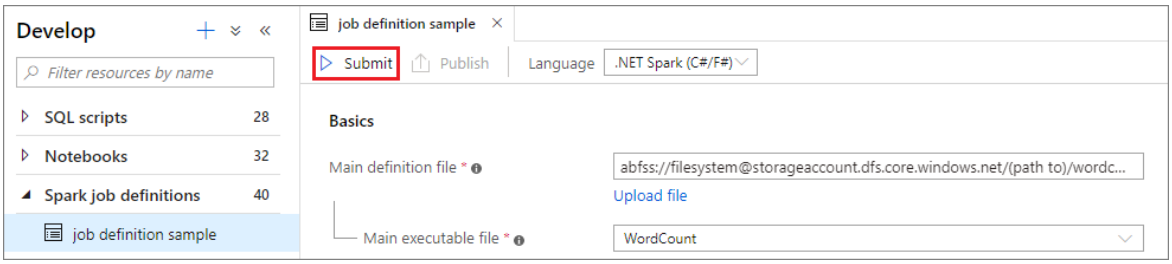
After creating an Apache Spark job definition, you can submit it to an Apache Spark pool. Make sure you are the **Storage Blob Data Contributor** of the ADLS Gen2 filesystem you want to work with. If you aren't, you need to add the permission manually.

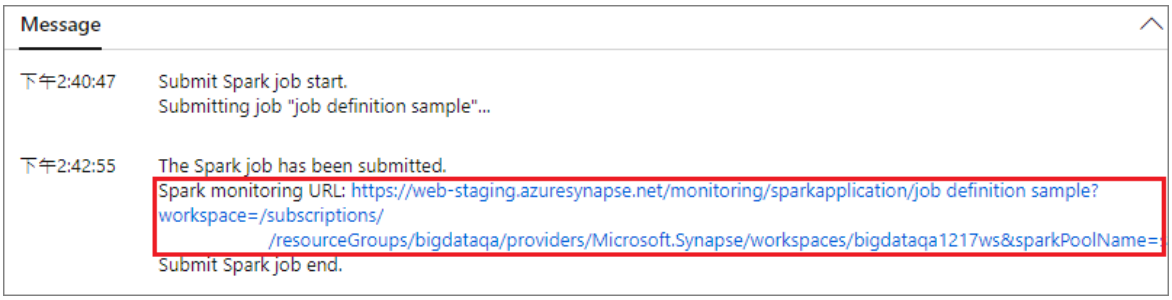
Scenario 1: Submit Apache Spark job definition

1. Open an Apache spark job definition window by selecting it.



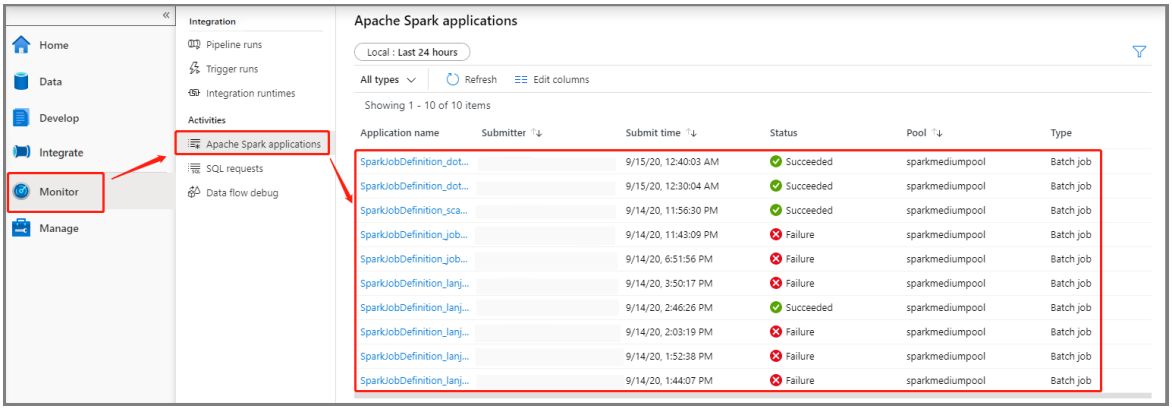
2. Select **Submit** button to submit your project to the selected Apache Spark Pool. You can select **Spark monitoring URL** tab to see the LogQuery of the Apache Spark application.



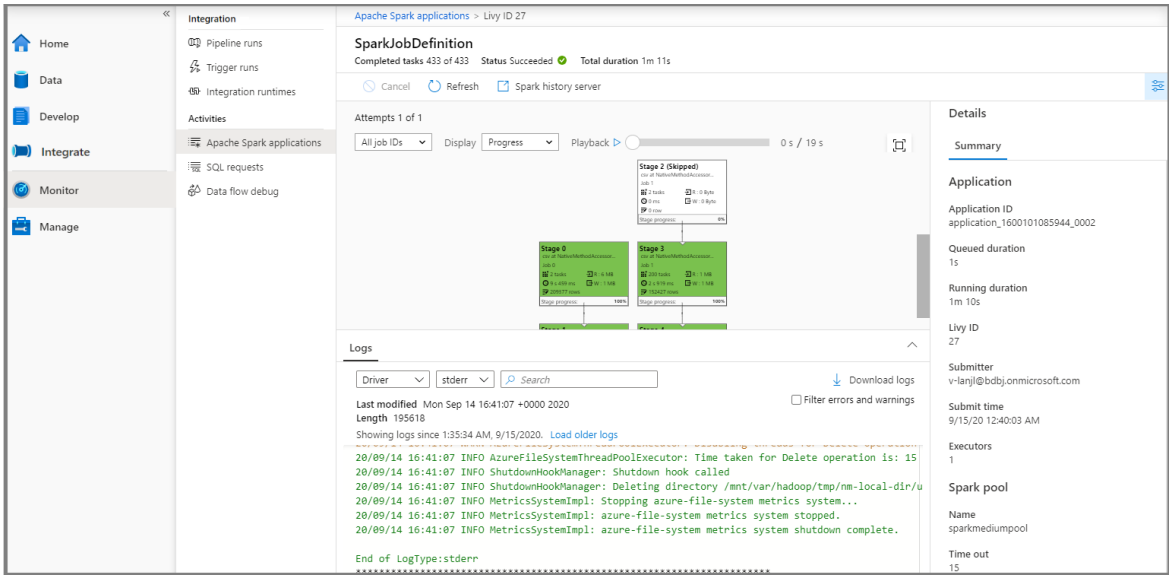


Scenario 2: View Apache Spark job running progress

1. Select **Monitor**, then select the **Apache Spark applications** option. You can find the submitted Apache Spark application.

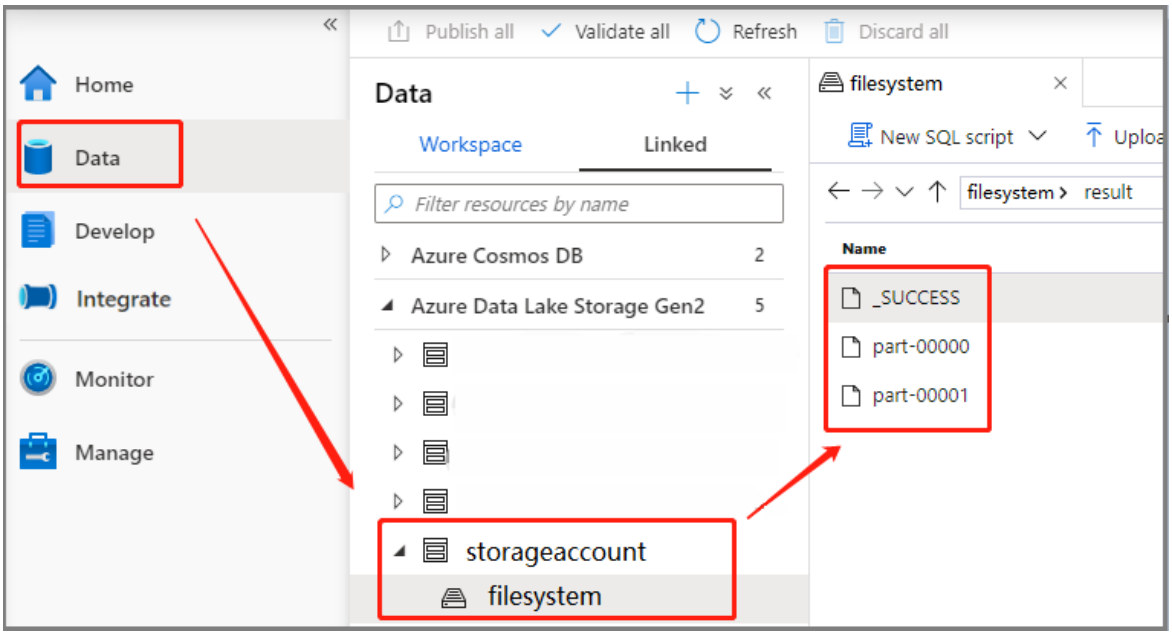


2. Then select an Apache Spark application, the **SparkJobDefinition** job window displays. You can view the job execution progress from here.



Scenario 3: Check output file

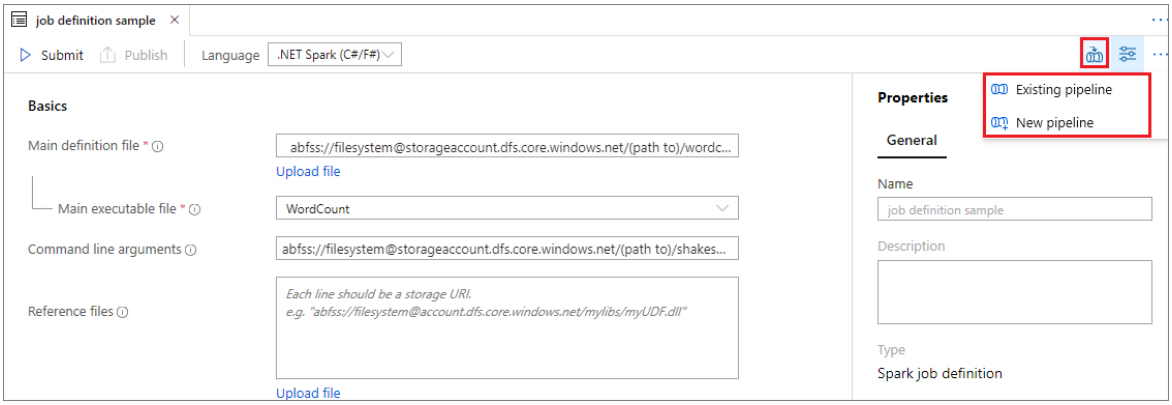
1. Select **Data -> Linked -> Azure Data Lake Storage Gen2** (hozhaobdbj), open the **result** folder created earlier, you can go to the result folder and check if the output is generated.

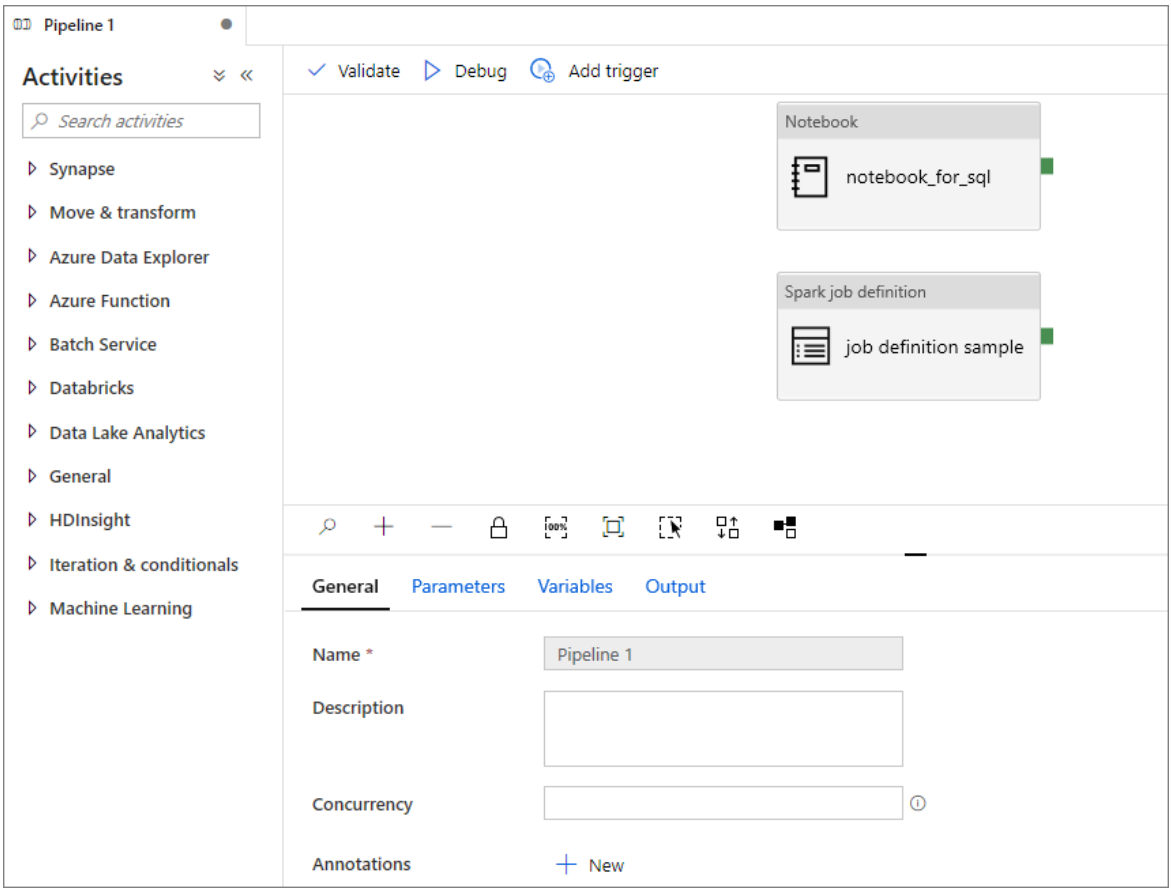


Add an Apache Spark job definition into pipeline

In this section, you add an Apache Spark job definition into pipeline.

1. Open an existing Apache Spark job definition.
2. Select the icon on the top right of Apache Spark job definition, choose **Existing Pipeline**, or **New pipeline**. You can refer to Pipeline page for more information.





Next steps

Next you can use Azure Synapse Studio to create Power BI datasets and manage Power BI data. Advance to the Linking a Power BI workspace to a Synapse workspace article to learn more.

Apache Spark in Azure Synapse Analytics Core Concepts

8/19/2022 • 4 minutes to read • Edit Online

Apache Spark is a parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. Apache Spark in Azure Synapse Analytics is one of Microsoft's implementations of Apache Spark in the cloud.

Azure Synapse makes it easy to create and configure Spark capabilities in Azure. Azure Synapse provides a different implementation of these Spark capabilities that are documented here.

Spark pools

A serverless Apache Spark pool is created in the Azure portal. It's the definition of a Spark pool that, when instantiated, is used to create a Spark instance that processes data. When a Spark pool is created, it exists only as metadata, and no resources are consumed, running, or charged for. A Spark pool has a series of properties that control the characteristics of a Spark instance. These characteristics include but aren't limited to name, size, scaling behavior, time to live.

As there's no dollar or resource cost associated with creating Spark pools, any number can be created with any number of different configurations. Permissions can also be applied to Spark pools allowing users only to have access to some and not others.

A best practice is to create smaller Spark pools that may be used for development and debugging and then larger ones for running production workloads.

You can read how to create a Spark pool and see all their properties here Get started with Spark pools in Azure Synapse Analytics

Spark instances

Spark instances are created when you connect to a Spark pool, create a session, and run a job. As multiple users may have access to a single Spark pool, a new Spark instance is created for each user that connects.

When you submit a second job, if there is capacity in the pool, the existing Spark instance also has capacity. Then, the existing instance will process the job. Otherwise, if capacity is available at the pool level, then a new Spark instance will be created.

Examples

Example 1

- You create a Spark pool called SP1; it has a fixed cluster size of 20 nodes.
- You submit a notebook job, J1 that uses 10 nodes, a Spark instance, SI1 is created to process the job.
- You now submit another job, J2, that uses 10 nodes because there is still capacity in the pool and the instance, the J2, is processed by SI1.
- If J2 had asked for 11 nodes, there would not have been capacity in SP1 or SI1. In this case, if J2 comes from a notebook, then the job will be rejected; if J2 comes from a batch job, then it will be queued.

Example 2

- You create a Spark pool call SP2; it has an autoscale enabled 10 – 20 nodes

- You submit a notebook job, J1 that uses 10 nodes, a Spark instance, SI1, is created to process the job.
- You now submit another job, J2, that uses 10 nodes, because there is still capacity in the pool the instance auto grows to 20 nodes and processes J2.

Example 3

- You create a Spark pool called SP1; it has a fixed cluster size of 20 nodes.
- You submit a notebook job, J1 that uses 10 nodes, a Spark instance, SI1 is created to process the job.
- Another user, U2, submits a Job, J3, that uses 10 nodes, a new Spark instance, SI2, is created to process the job.
- You now submit another job, J2, that uses 10 nodes because there's still capacity in the pool and the instance, J2, is processed by SI1.

Quotas and resource constraints in Apache Spark for Azure Synapse

Workspace level

Every Azure Synapse workspace comes with a default quota of vCores that can be used for Spark. The quota is split between the user quota and the dataflow quota so that neither usage pattern uses up all the vCores in the workspace. The quota is different depending on the type of your subscription but is symmetrical between user and dataflow. However if you request more vCores than are remaining in the workspace, then you will get the following error:

```
Failed to start session: [User] MAXIMUM_WORKSPACE_CAPACITY_EXCEEDED
Your Spark job requested 480 vcores.
However, the workspace only has xxx vcores available out of quota of yyy vcores.
Try reducing the numbers of vcores requested or increasing your vcore quota. Click here for more information
- https://go.microsoft.com/fwlink/?linkid=213499
```

The link in the message points to this article.

The following article describes how to request an increase in workspace vCore quota.

- Select "Azure Synapse Analytics" as the service type.
- In the Quota details window, select Apache Spark (vCore) per workspace

Request a capacity increase via the Azure portal

Spark pool level

When you define a Spark pool you are effectively defining a quota per user for that pool, if you run multiple notebooks or jobs or a mix of the 2 it is possible to exhaust the pool quota. If you do, then an error message like the following will be generated

```
Failed to start session: Your Spark job requested xx vcores.
However, the pool is consuming yy vcores out of available zz vcores.Try ending the running job(s) in the
pool, reducing the numbers of vcores requested, increasing the pool maximum size or using another pool
```

To solve this problem you have to reduce your usage of the pool resources before submitting a new resource request by running a notebook or a job.

Next steps

- Azure Synapse Analytics
- Apache Spark Documentation

Apache Spark pool configurations in Azure Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

A Spark pool is a set of metadata that defines the compute resource requirements and associated behavior characteristics when a Spark instance is instantiated. These characteristics include but aren't limited to name, number of nodes, node size, scaling behavior, and time to live. A Spark pool in itself does not consume any resources. There are no costs incurred with creating Spark pools. Charges are only incurred once a Spark job is executed on the target Spark pool and the Spark instance is instantiated on demand.

You can read how to create a Spark pool and see all their properties here Get started with Spark pools in Synapse Analytics

Isolated Compute

The Isolated Compute option provides additional security to Spark compute resources from untrusted services by dedicating the physical compute resource to a single customer. Isolated compute option is best suited for workloads that require a high degree of isolation from other customer's workloads for reasons that include meeting compliance and regulatory requirements.
The Isolate Compute option is only available with the XXXLarge (80 vCPU / 504 GB) node size and only available in the following regions. The isolated compute option can be enabled or disabled after pool creation although the instance may need to be restarted. If you expect to enable this feature in the future, ensure that your Synapse workspace is created in an isolated compute supported region.

- East US
- West US 2
- South Central US
- US Gov Arizona
- US Gov Virginia

Nodes

Apache Spark pool instance consists of one head node and two or more worker nodes with a minimum of three nodes in a Spark instance. The head node runs additional management services such as Livy, Yarn Resource Manager, Zookeeper, and the Spark driver. All nodes run services such as Node Agent and Yarn Node Manager. All worker nodes run the Spark Executor service.

Node Sizes

A Spark pool can be defined with node sizes that range from a Small compute node with 4 vCore and 32 GB of memory up to a XXLarge compute node with 64 vCore and 512 GB of memory per node. Node sizes can be altered after pool creation although the instance may need to be restarted.

SIZE	VCORE	MEMORY
Small	4	32 GB
Medium	8	64 GB

SIZE	VCORE	MEMORY
Large	16	128 GB
XLarge	32	256 GB
XXLarge	64	512 GB
XXX Large (Isolated Compute)	80	504 GB

Autoscale

Apache Spark pools provide the ability to automatically scale up and down compute resources based on the amount of activity. When the autoscale feature is enabled, you can set the minimum and maximum number of nodes to scale. When the autoscale feature is disabled, the number of nodes set will remain fixed. This setting can be altered after pool creation although the instance may need to be restarted.

Elastic pool storage

Apache Spark pools utilize temporary disk storage while the pool is instantiated. For many Spark jobs, it is difficult to estimate cluster storage requirements, which may cause your Spark jobs to fail if the worker nodes exhaust storage. Elastic pool storage allows the Spark engine to monitor worker node temporary cluster storage, and attach additional disks if needed. No action is required by customers. Customers should see fewer job failures as a result of elastic pool storage.

> **NOTE**
> Azure Synapse Elastic pool storage is currently in Public Preview. During Public Preview there is no charge for use of Elastic Pool Storage.

Automatic pause

The automatic pause feature releases resources after a set idle period, reducing the overall cost of an Apache Spark pool. The number of minutes of idle time can be set once this feature is enabled. The automatic pause feature is independent of the autoscale feature. Resources can be paused whether the autoscale is enabled or disabled. This setting can be altered after pool creation although active sessions will need to be restarted.

Next steps

- Azure Synapse Analytics
- Apache Spark Documentation

GPU-accelerated Apache Spark pools in Azure Synapse Analytics (Preview)

8/19/2022 • 3 minutes to read • Edit Online

Azure Synapse Analytics now supports Apache Spark pools accelerated with graphics processing units (GPUs).

By using NVIDIA GPUs, data scientists and engineers can reduce the time necessary to run data integration pipelines, score machine learning models, and more. This article describes how GPU-accelerated pools can be created and used with Azure Synapse Analytics. This article also details the GPU drivers and libraries that are pre-installed as part of the GPU-accelerated runtime.

> **NOTE**
> Azure Synapse GPU-enabled pools are currently in Public Preview.

Create a GPU-accelerated pool

To simplify the process for creating and managing pools, Azure Synapse takes care of pre-installing low-level libraries and setting up all the complex networking requirements between compute nodes. This integration allows users to get started with GPU- accelerated pools within just a few minutes. To learn more about how to create a GPU-accelerated pool, you can visit the quickstart on how to create a GPU-accelerated pool.

> **NOTE**
> - GPU-accelerated pools can be created in workspaces located in East US, Australia East, and North Europe.
> - GPU-accelerated pools are only availble with the Apache Spark 3 runtime.
> - You might need to request a limit increase in order to create GPU-enabled clusters.

GPU-accelerated runtime

NVIDIA GPU driver, CUDA, and cuDNN

Azure Synapse Analytics now offers GPU-accelerated Apache Spark pools, which include various NVIDIA libraries and configurations. By default, Azure Synapse Analytics installs the NVIDIA driver and libraries required to use GPUs on Spark driver and worker instances:

- CUDA 11.2
- libnccl2=2.8.4
- libnccl-dev=2.8.4
- libcudnn8=8.1.1
- libcudnn8-dev=8.1.1

> **NOTE**
> This software contains source code provided by NVIDIA Corporation. Specifically, to support the GPU-accelerated pools, Azure Synapse Apache Spark pools include code from CUDA Samples.

NVIDIA End User License Agreement (EULA)

When you select a GPU-accelerated Hardware option in Synapse Spark, you implicitly agree to the terms and conditions outlined in the NVIDIA EULA with respect to:

- CUDA 11.2: EULA :: CUDA Toolkit Documentation (nvidia.com)
- libnccl2=2.8.4: nccl/LICENSE.txt at master · NVIDIA/nccl (github.com)
- libnccl-dev=2.8.4: nccl/LICENSE.txt at master · NVIDIA/nccl (github.com)
- libcudnn8=8.1.1: Software License Agreement :: NVIDIA Deep Learning cuDNN Documentation
- libcudnn8-dev=8.1.1: Software License Agreement :: NVIDIA Deep Learning cuDNN Documentation
- The CUDA, NCCL, and cuDNN libraries, and the NVIDIA End User License Agreement (with NCCL Supplement) for the NCCL library

Accelerate ETL workloads

With built-in support for NVIDIA's RAPIDS Accelerator for Apache Spark, GPU-accelerated Spark pools in Azure Synapse can provide significant performance improvements compared to standard analytical benchmarks without requiring any code changes. Built on top of NVIDIA CUDA and UCX, NVIDIA RAPIDS enables GPU-accelerated SQL, DataFrame operations, and Spark shuffles. Since there are no code changes required to leverage these accelerations, users can also accelerate their data pipelines that rely on Linux Foundation's Delta Lake or Microsoft's Hyperspace indexing.

To learn more about how you can use the NVIDIA RAPIDS Accelerator with your GPU-accelerated pool in Azure Synapse Analytics, visit this guide on how to improve performance with RAPIDS.

Train deep learning models

Deep learning models are often data and computation intensive. Because of this, organizations often accelerate their training process with GPU-enabled clusters. In Azure Synapse Analytics, organizations can build models using frameworks like Tensorflow and PyTorch. Then, users can scale up their deep learning models with Horovod and Petastorm.

To learn more about how you can train distributed deep learning models, visit the following guides: - Tutorial: Distributed training with Horovod and Tensorflow - Tutorial: Distributed training with Horovod and PyTorch

Improve machine learning scoring workloads

Many organizations rely on large batch scoring jobs to frequently execute during narrow windows of time. To achieve improved batch scoring jobs, you can also use GPU-accelerated Spark pools with Microsoft's Hummingbird library. With Hummingbird, users can take their traditional, tree-based ML models and compile them into tensor computations. Hummingbird allows users to then seamlessly leverage native hardware acceleration and neural network frameworks to accelerate their ML model scoring without needing to rewrite their models.

Next steps

- Azure Synapse Analytics

What is Delta Lake

8/19/2022 • 2 minutes to read • Edit Online

Azure Synapse Analytics is compatible with Linux Foundation Delta Lake. Delta Lake is an open-source storage layer that brings ACID (atomicity, consistency, isolation, and durability) transactions to Apache Spark and big data workloads.

The current version of Delta Lake included with Azure Synapse has language support for Scala, PySpark, and .NET. There are links at the bottom of the page to more detailed examples and documentation.

Key features

FEATURE	DESCRIPTION
ACID Transactions	Data lakes are typically populated via multiple processes and pipelines, some of which are writing data concurrently with reads. Prior to Delta Lake and the addition of transactions, data engineers had to go through a manual error prone process to ensure data integrity. Delta Lake brings familiar ACID transactions to data lakes. It provides serializability, the strongest level of isolation level. Learn more at Diving into Delta Lake: Unpacking the Transaction Log.
Scalable Metadata Handling	In big data, even the metadata itself can be "big data". Delta Lake treats metadata just like data, leveraging Spark's distributed processing power to handle all its metadata. As a result, Delta Lake can handle petabyte-scale tables with billions of partitions and files at ease.
Time Travel (data versioning)	The ability to "undo" a change or go back to a previous version is one of the key features of transactions. Delta Lake provides snapshots of data enabling you to revert to earlier versions of data for audits, rollbacks or to reproduce experiments. Learn more in Introducing Delta Lake Time Travel for Large Scale Data Lakes.
Open Format	Apache Parquet is the baseline format for Delta Lake, enabling you to leverage the efficient compression and encoding schemes that are native to the format.
Unified Batch and Streaming Source and Sink	A table in Delta Lake is both a batch table, as well as a streaming source and sink. Streaming data ingest, batch historic backfill, and interactive queries all just work out of the box.
Schema Enforcement	Schema enforcement helps ensure that the data types are correct and required columns are present, preventing bad data from causing data inconsistency. For more information, see Diving Into Delta Lake: Schema Enforcement & Evolution
Schema Evolution	Delta Lake enables you to make changes to a table schema that can be applied automatically, without having to write migration DDL. For more information, see Diving Into Delta Lake: Schema Enforcement & Evolution

FEATURE	DESCRIPTION
Audit History	Delta Lake transaction log records details about every change made to data providing a full audit trail of the changes.
Updates and Deletes	Delta Lake supports Scala / Java / Python and SQL APIs for a variety of functionality. Support for merge, update, and delete operations helps you to meet compliance requirements. For more information, see Announcing the Delta Lake 0.6.1 Release, Announcing the Delta Lake 0.7 Release and Simple, Reliable Upserts and Deletes on Delta Lake Tables using Python APIs, which includes code snippets for merge, update, and delete DML commands.
100% Compatible with Apache Spark API	Developers can use Delta Lake with their existing data pipelines with minimal change as it is fully compatible with existing Spark implementations.

For full documentation, see the Delta Lake Documentation Page

For more information, see Delta Lake Project.

Next steps

- .NET for Apache Spark documentation
- Azure Synapse Analytics

The need for optimize write on Apache Spark

8/19/2022 • 3 minutes to read • Edit Online

Analytical workloads on Big Data processing engines such as Apache Spark perform most efficiently when using standardized larger file sizes. The relation between the file size, the number of files, the number of Spark workers and its configurations, play a critical role on performance. Ingestion workloads into data lake tables may have the inherited characteristic of constantly writing lots of small files; this scenario is commonly known as the "small file problem".

Optimize Write is a Delta Lake on Synapse feature that reduces the number of files written and aims to increase individual file size of the written data. It dynamically optimizes partitions while generating files with a default 128 MB size. The target file size may be changed per workload requirements using configurations.

This feature achieves the file size by using an extra data shuffle phase over partitions, causing an extra processing cost while writing the data. The small write penalty should be outweighed by read efficiency on the tables.

> **NOTE**
> - Optimize write is available as a Preview feature.
> - It is available on Synapse Pools for Apache Spark versions 3.1 and 3.2.

Benefits of Optimize Writes

- It's available on Delta Lake tables for both Batch and Streaming write patterns.
- There's no need to change the `spark.write` command pattern. The feature is enabled by a configuration setting or a table property.
- It reduces the number of write transactions as compared to the OPTIMIZE command.
- OPTIMIZE operations will be faster as it will operate on fewer files.
- VACUUM command for deletion of old unreferenced files will also operate faster.
- Queries will scan fewer files with more optimal file sizes, improving either read performance or resource usage.

Optimize write usage scenarios

When to use it

- Delta lake partitioned tables subject to write patterns that generate suboptimal (less than 128 MB) or non-standardized files sizes (files with different sizes between itself).
- Repartitioned data frames that will be written to disk with suboptimal files size.
- Delta lake partitioned tables targeted by small batch SQL commands like UPDATE, DELETE, MERGE, CREATE TABLE AS SELECT, INSERT INTO, etc.
- Streaming ingestion scenarios with append data patterns to Delta lake partitioned tables where the extra write latency is tolerable.

When to avoid it

- Non partitioned tables.
- Use cases where extra write latency isn't acceptable.
- Large tables with well defined optimization schedules and read patterns.

How to enable and disable the optimize write feature

The optimize write feature is disabled by default.

Once the configuration is set for the pool or session, all Spark write patterns will use the functionality.

To use the optimize write feature, enable it using the following configuration:

1. Scala and PySpark

```
spark.conf.set("spark.microsoft.delta.optimizeWrite.enabled", "true")
```

2. Spark SQL

```
SET `spark.microsoft.delta.optimizeWrite.enabled` = true
```

To check the current configuration value, use the command as shown below:

1. Scala and PySpark

```
spark.conf.get("spark.microsoft.delta.optimizeWrite.enabled")
```

2. Spark SQL

```
SET `spark.microsoft.delta.optimizeWrite.enabled`
```

To disable the optimize write feature, change the following configuration as shown below:

1. Scala and PySpark

```
spark.conf.set("spark.microsoft.delta.optimizeWrite.enabled", "false")
```

2. Spark SQL

```
SET `spark.microsoft.delta.optimizeWrite.enabled` = false
```

Controlling optimize write using table properties

On new tables

1. SQL

```
CREATE TABLE <table_name> TBLPROPERTIES (delta.autoOptimize.optimizeWrite = true)
```

2. Scala

Using the DeltaTableBuilder API:

```
val table = DeltaTable.create()
  .tableName("<table_name>")
  .addColumnn("<colName>", <dataType>)
  .location("<table_location>")
  .property("delta.autoOptimize.optimizeWrite", "true")
  .execute()
```

On existing tables

1. SQL

```
ALTER TABLE <table_name> SET TBLPROPERTIES (delta.autoOptimize.optimizeWrite = true)
```

2. Scala

Using the DeltaTableBuilder API

```
val table = DeltaTable.replace()
  .tableName("<table_name>")
  .location("<table_location>")
  .property("delta.autoOptimize.optimizeWrite", "true")
  .execute()
```

How to get & change the current max file size configuration for Optimize Write

To get the current config value, use the bellow commands. The default is 128 MB.

1. Scala and PySpark

```
spark.conf.get("spark.microsoft.delta.optimizeWrite.binSize")
```

2. SQL

```
SET `spark.microsoft.delta.optimizeWrite.binSize`
```

- To change the config value

1. Scala and PySpark

```
spark.conf.set("spark.microsoft.delta.optimizeWrite.binSize", "134217728")
```

2. SQL

```
SET `spark.microsoft.delta.optimizeWrite.binSize` = 134217728
```

Next steps

- Use serverless Apache Spark pool in Synapse Studio.
- Run a Spark application in notebook.
- Create Apache Spark job definition in Azure Studio.

Automatically scale Azure Synapse Analytics Apache Spark pools

8/19/2022 • 3 minutes to read • Edit Online

Apache Spark for Azure Synapse Analytics pool's Autoscale feature automatically scales the number of nodes in a cluster instance up and down. During the creation of a new Apache Spark for Azure Synapse Analytics pool, a minimum and maximum number of nodes can be set when Autoscale is selected. Autoscale then monitors the resource requirements of the load and scales the number of nodes up or down. There's no additional charge for this feature.

Metrics monitoring

Autoscale continuously monitors the Spark instance and collects the following metrics:

METRIC	DESCRIPTION
Total Pending CPU	The total number of cores required to start execution of all pending jobs.
Total Pending Memory	The total memory (in MB) required to start execution of all pending jobs.
Total Free CPU	The sum of all unused cores on the active nodes.
Total Free Memory	The sum of unused memory (in MB) on the active nodes.
Used Memory per Node	The load on a node. A node on which 10 GB of memory is used, is considered under more load than a worker with 2 GB of used memory.

The above metrics are checked every 30 seconds. Autoscale makes scale-up and scale-down decisions based on these metrics.

Load-based scale conditions

When the following conditions are detected, Autoscale will issue a scale request:

SCALE-UP	SCALE-DOWN
Total pending CPU is greater than total free CPU for more than 1 minute.	Total pending CPU is less than total free CPU for more than 2 minutes.
Total pending memory is greater than total free memory for more than 1 minute.	Total pending memory is less than total free memory for more than 2 minutes.

For scale-up, the Azure Synapse Autoscale service calculates how many new nodes are needed to meet the current CPU and memory requirements, and then issues a scale-up request to add the required number of nodes.

For scale-down, based on the number of executors, application masters per node, the current CPU and memory

requirements, Autoscale issues a request to remove a certain number of nodes. The service also detects which nodes are candidates for removal based on current job execution. The scale down operation first decommissions the nodes, and then removes them from the cluster.

Get started

Create a serverless Apache Spark pool with Autoscaling

To enable the Autoscale feature, complete the following steps as part of the normal pool creation process:

1. On the **Basics** tab, select the **Enable autoscale** checkbox.
2. Enter the desired values for the following properties:
   - **Min** number of nodes.
   - **Max** number of nodes.

The initial number of nodes will be the minimum. This value defines the initial size of the instance when it's created. The minimum number of nodes can't be fewer than three.

Optionally, you can enable dynamic allocation of executors in scenarios where the executor requirements are vastly different across stages of a Spark Job or the volume of data processed fluctuates with time. By enabling Dynamic Allocation of Executors, we can utilize capacity as required.

On enabling dynamic allocation, it allows the job to scale the number of executors within min and max number of executors specified.

Apache Spark enables configuration of Dynamic Allocation of Executors through code as below:

```
%%configure -f
{
    "conf" : {
        "spark.dynamicAllocation.maxExecutors" : "6",
        "spark.dynamicAllocation.enable": "true",
        "spark.dynamicAllocation.minExecutors": "2"
 }
}
```

The defaults specified through the code override the values set through the user interface.

On enabling Dynamic allocation, Executors scale up or down based on the utilization of the Executors. This ensure that the Executors are provisioned in accordance with the needs of the job being run.

Best practices

Consider the latency of scale up or scale down operations

It can take 1 to 5 minutes for a scaling operation to complete.

Prepare for scaling down

During the instance scaling down process, Autoscale will put the nodes in decommissioning state so that no new executors can launch on that node.

The running jobs will continue to run and finish. The pending jobs will wait to be scheduled as normal with fewer available nodes.

Next steps

Quickstart to set up a new Spark pool Create a Spark pool

Intelligent Cache in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

The Intelligent Cache works seamlessly behind the scenes and caches data to help speed up the execution of Spark as it reads from your ADLS Gen2 data lake. It also automatically detects changes to the underlying files and will automatically refresh the files in the cache, providing you with the most recent data and when the cache size reaches its limit, the cache will automatically release the least read data to make space for more recent data. This feature lowers the total cost of ownership by improving performance up to 65% on subsequent reads of the files that are stored in the available cache for Parquet files and 50% for CSV files.

When querying a file or table from your data lake, the Apache Spark engine in Synapse will make a call to the remote ADLS Gen2 storage to read the underlying files. With every query request to read the same data, the Spark engine must make a call to remote ADLS Gen2 storage. This redundant process adds latency to your total processing time. Spark provides a caching feature that you must manually set the cache and release the cache to minimize the latency and improve overall performance. However, this can cause results to have stale data if the underlying data changes.

The Synapse Intelligent Cache simplifies this process by automatically caching each read within the allocated cache storage space on each Spark node. Each request for a file will check to see if the file exists in the cache and compare the tag from the remote storage to determine if the file is stale. If the file doesn't exist or if the file is stale, then Spark will read the file and store it in the cache. When the cache becomes full, the file with the oldest last access time will be evicted from the cache to allow for more recent files.

The Synapse cache is a single cache per node. If you're using a medium size node and run with two small executors on a single medium size node, these two executors would share the same cache.

Enable or Disable the cache

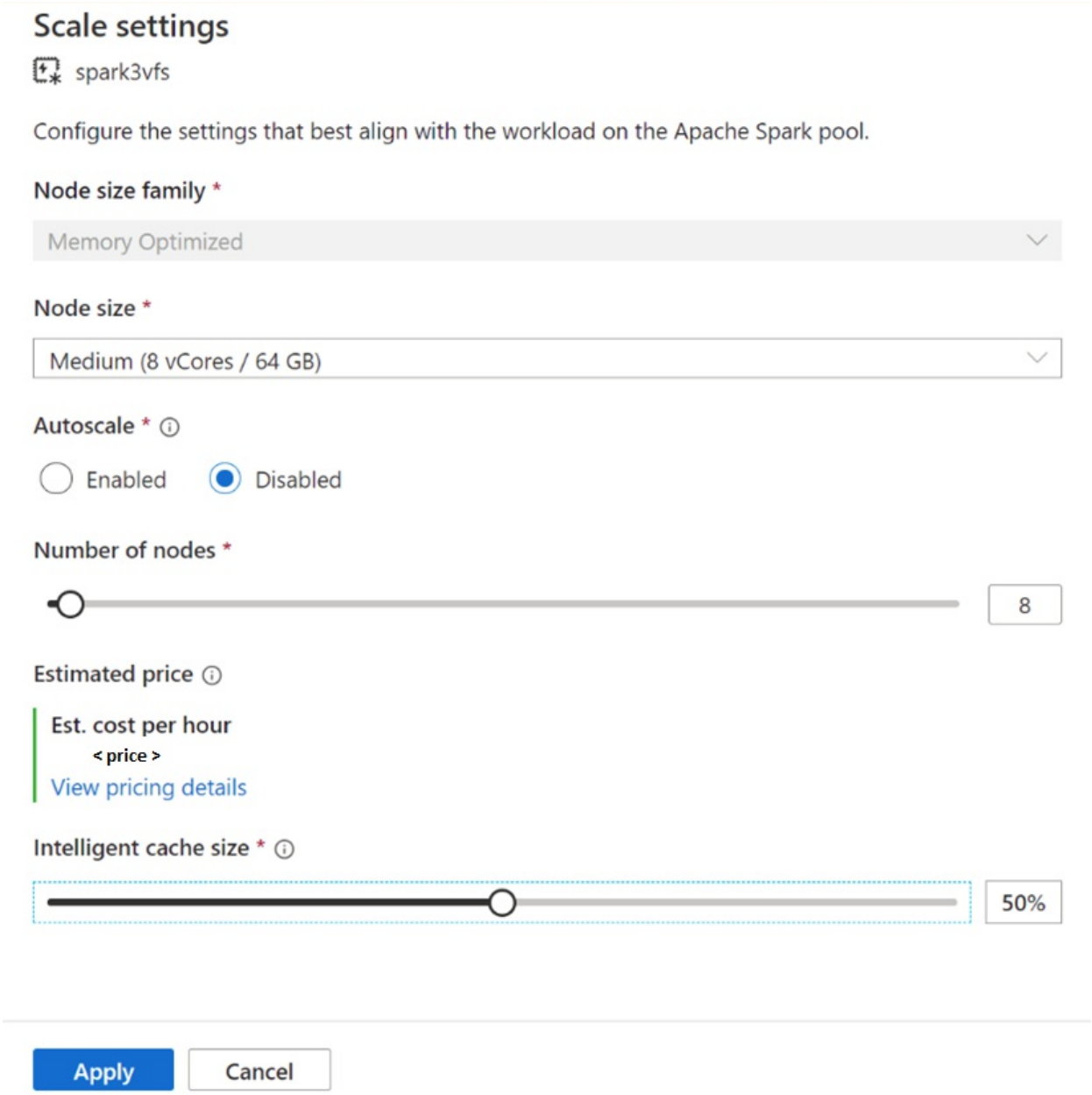
The cache size can be adjusted based on the percent of total disk size available for each Apache Spark pool.By default, the cache is set to disabled but it's as easy as moving the **slider** bar from 0 (disabled) to the desired percentage for your cache size to enable it.We reserve a minimum of 20% of available disk space for data shuffles.For shuffle intensive workloads, you can minimize the cache size or disable the cache. We recommend starting with a 50% cache size and adjust as necessary. It's important to note that if your workload requires a lot of disk space on the local SSD for shuffle or RDD caching, then consider reducing the cache size to reduce the chance of failure due to insufficient storage. The actual size of the available storage and the cache size on each node will depend on the node family and node size.

Enabling cache for new Spark pools

When creating a new Spark pool, browse under the **additional settings** tab to find the **Intelligent Cache slider** you can move to your preferred size to enable the feature.

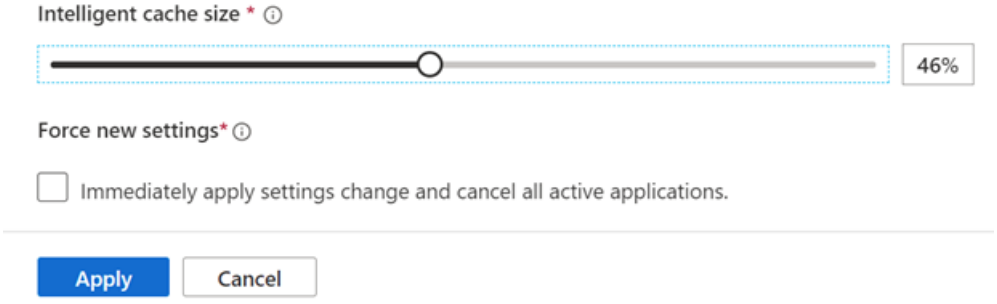
Enabling/Disabling cache for existing Spark pools

For existing Spark pools, browse to the **Scale settings** of your Apache Spark pool of choice to enable, by moving the **slider** to a value more than 0, or disable it, by moving **slider** to 0.



Changing cache size for existing Spark pools

To change the Intelligent Cache size of a pool, you must force a restart if the pool has active sessions. If the Spark pool has an active session, then it will show **Force new settings**. Click on the **check box** and select **Apply** to automatically restart the session.



Enabling and disabling the cache within the session

Easily disable the Intelligent Cache within a session by running the following code in your notebook:

```
%spark

spark.conf.set("spark.synapse.vegas.useCache", "false")
```

```
%pyspark

spark.conf.set('spark.synapse.vegas.useCache', 'false')
```

And enable by running:

```
%spark

spark.conf.set("spark.synapse.vegas.useCache", "true")
```

```
%pyspark

spark.conf.set('spark.synapse.vegas.useCache', 'true')
```

When to use the Intelligent Cache and when not to?

This feature will benefit you if:

- Your workload requires reading the same file multiple times and the file size can fit into the cache.
- Your workload uses Delta tables, parquet file formats and CSV files.
- You're using Apache Spark 3 or higher on Azure Synapse.

You won't see the benefit of this feature if:

- You're reading a file that exceeds the cache size because the beginning of the files could be evicted and subsequent queries will have to refetch the data from the remote storage. In this case, you won't see any benefits from the Intelligent Cache and you may want to increase your cache size and/or node size.
- Your workload requires large amounts of shuffle, then disabling the Intelligent Cache will free up available space to prevent your job from failing due to insufficient storage space.
- You're using a Spark 2.4 pool, you'll need to upgrade your pool to the latest version of Spark.

Learn more

To learn more on Apache Spark, see the following articles:

- What is Apache Spark
- Apache Spark core concepts
- Azure Synapse Runtime for Apache Spark 3.2
- Apache Spark pool sizes and configurations

To learn about configuring Spark session settings

- Configure Spark session settings
- How to set Spark / Pyspark custom configs

Next steps

An Apache Spark pool provides open-source big data compute capabilities where data can be loaded, modeled, processed, and distributed for faster analytic insight. To learn more about how to create one to run your Spark workloads, visit the following tutorials:

- Create a new serverless Apache Spark pool using the Azure portal

- Create a serverless Apache Spark pool using Synapse Studio

Azure Synapse runtimes

8/19/2022 • 2 minutes to read • Edit Online

Apache Spark pools in Azure Synapse use runtimes to tie together essential component versions such as Azure Synapse optimizations, packages, and connectors with a specific Apache Spark version. Each runtime will be upgraded periodically to include new improvements, features, and patches.

When you create a serverless Apache Spark pool, you will have the option to select the corresponding Apache Spark version. Based on this, the pool will come pre-installed with the associated runtime components and packages. The runtimes have the following advantages:

- Faster session startup times
- Tested compatibility with specific Apache Spark versions
- Access to popular, compatible connectors and open-source packages

NOTE

- Maintenance updates will be automatically applied to new sessions for a given serverless Apache Spark pool.
- You should test and validate that your applications run properly when using new runtime versions.

IMPORTANT

Log4j 1.2.x security patches

Open-source Log4j library version 1.2.x has several known CVEs (Common Vulnerabilities and Exposures), as described here.

On all Synapse Spark Pool runtimes, we have patched the Log4j 1.2.17 JARs to mitigate the following CVEs: CVE-2019-1751, CVE-2020-9488, CVE-2021-4104, CVE-2022-23302, CVE-2022-2330, CVE-2022-23307

The applied patch works by removing the following files which are required to invoke the vulnerabilities:

- `org/apache/log4j/net/SocketServer.class`
- `org/apache/log4j/net/SMTPAppender.class`
- `org/apache/log4j/net/JMSAppender.class`
- `org/apache/log4j/net/JMSSink.class`
- `org/apache/log4j/jdbc/JDBCAppender.class`
- `org/apache/log4j/chainsaw/*`

While the above classes were not used in the default Log4j configurations in Synapse, it is possible that some user application could still depend on it. If your application needs to use these classes, use Library Management to add a secure version of Log4j to the Spark Pool. **Do not use Log4j version 1.2.17**, as it would be reintroducing the vulnerabilities.

Supported Azure Synapse runtime releases

The following table lists the runtime name, Apache Spark version, and release date for supported Azure Synapse Runtime releases.

RUNTIME NAME	RELEASE DATE	RELEASE STAGE	END OF LIFE ANNOUNCEMENT DATE	END OF LIFE EFFECTIVE DATE
Azure Synapse Runtime for Apache Spark 3.2	July 8, 2022	GA	July 8, 2023	July 8, 2024
Azure Synapse Runtime for Apache Spark 3.1	May 26, 2021	LTS	January 26, 2023	January 26, 2024
Azure Synapse Runtime for Apache Spark 2.4	December 15, 2020	**End of Life Announced (EOLA)**	**July 29, 2022**	**July 28, 2023**

Runtime release stages

For the complete runtime for Apache Spark lifecycle and support policies, refer to Synapse runtime for Apache Spark lifecycle and supportability.

Runtime patching

Azure Synapse runtime for Apache Spark patches are rolled out monthly containing bug, feature and security fixes to the Apache Spark core engine, language environments, connectors and libraries.

The patch policy differs based on the runtime lifecycle stage:

1. Generally Available (GA) runtime: Receive no upgrades on major versions (i.e. 3.x -> 4.x). And will upgrade a minor version (i.e. 3.x -> 3.y) as long as there are no deprecation or regression impacts.
2. Preview runtime: No major version upgrades unless strictly necessary. Minor versions (3.x -> 3.y) will be upgraded to add latest features to a runtime.
3. Long Term Support (LTS) runtime will be patched with security fixes only.
4. End of life announced (EOLA) runtime will not have bug and feature fixes. Security fixes will be backported based on risk assessment.

Azure Synapse Runtime for Apache Spark 3.2

8/19/2022 • 5 minutes to read • Edit Online

Azure Synapse Analytics supports multiple runtimes for Apache Spark. This document will cover the runtime components and versions for the Azure Synapse Runtime for Apache Spark 3.2.

Component versions

COMPONENT	VERSION
Apache Spark	3.2.1
Operating System	Ubuntu 18.04
Java	1.8.0_282
Scala	2.12.15
Hadoop	3.3.1
.NET Core	3.1
.NET	2.0.0
Delta Lake	1.2
Python	3.8

Scala and Java libraries

HikariCP-2.5.1.jar

JLargeArrays-1.5.jar

JTransforms-3.1.jar

RoaringBitmap-0.9.0.jar

ST4-4.0.4.jar

SparkCustomEvents-3.2.0-1.0.0.jar

TokenLibrary-assembly-3.0.0.jar

VegasConnector-1.1.01_2.12_3.2.0-SNAPSHOT.jar

activation-1.1.1.jar

adal4j-1.6.3.jar

aircompressor-0.21.jar

algebra_2.12-2.0.1.jar

aliyun-java-sdk-core-3.4.0.jar

aliyun-java-sdk-ecs-4.2.0.jar

aliyun-java-sdk-ram-3.0.0.jar

aliyun-java-sdk-sts-3.0.0.jar

aliyun-sdk-oss-3.4.1.jar

annotations-17.0.0.jar

antlr-runtime-3.5.2.jar

antlr4-runtime-4.8.jar

aopalliance-repackaged-2.6.1.jar

apache-log4j-extras-1.2.17.jar

apiguardian-api-1.1.0.jar

arpack-2.2.1.jar

arpack_combined_all-0.1.jar

arrow-format-2.0.0.jar

arrow-memory-core-2.0.0.jar

arrow-memory-netty-2.0.0.jar

arrow-vector-2.0.0.jar

audience-annotations-0.5.0.jar

avro-1.10.2.jar

avro-ipc-1.10.2.jar

avro-mapred-1.10.2.jar

aws-java-sdk-bundle-1.11.901.jar

azure-data-lake-store-sdk-2.3.6.jar

azure-eventhubs-3.3.0.jar

azure-eventhubs-spark_2.12-2.3.21.jar

azure-keyvault-core-1.0.0.jar

azure-storage-7.0.1.jar

azure-synapse-ml-pandas_2.12-1.0.0.jar

azure-synapse-ml-predict_2.12-1.0.jar

blas-2.2.1.jar

bonecp-0.8.0.RELEASE.jar

breeze-macros_2.12-1.2.jar

breeze_2.12-1.2.jar

cats-kernel_2.12-2.1.1.jar

chill-java-0.10.0.jar

chill_2.12-0.10.0.jar

client-sdk-1.14.0.jar

cntk-2.4.jar

commons-cli-1.2.jar

commons-codec-1.15.jar

commons-collections-3.2.2.jar

commons-compiler-3.0.16.jar

commons-compress-1.21.jar

commons-crypto-1.1.0.jar

commons-dbcp-1.4.jar

commons-io-2.8.0.jar

commons-lang-2.6.jar

commons-lang3-3.12.0.jar

commons-logging-1.1.3.jar

commons-math3-3.4.1.jar

commons-net-3.1.jar

commons-pool-1.5.4.jar

commons-pool2-2.6.2.jar

commons-text-1.6.jar

compress-lzf-1.0.3.jar

config-1.3.4.jar

core-1.1.2.jar

cos_api-bundle-5.6.19.jar

cosmos-analytics-spark-3.2.1-connector-1.6.0.jar

cudf-22.02.0-cuda11.jar

curator-client-2.13.0.jar

curator-framework-2.13.0.jar

curator-recipes-2.13.0.jar

datanucleus-api-jdo-4.2.4.jar

datanucleus-core-4.1.17.jar

datanucleus-rdbms-4.1.19.jar

delta-core_2.12-1.2.1.2.jar

derby-10.14.2.0.jar

dropwizard-metrics-hadoop-metrics2-reporter-0.1.2.jar

flatbuffers-java-1.9.0.jar

fluent-logger-jar-with-dependencies-jdk8.jar

gson-2.8.6.jar

guava-14.0.1.jar

hadoop-aliyun-3.3.1.5.0-57088972.jar

hadoop-annotations-3.3.1.5.0-57088972.jar

hadoop-aws-3.3.1.5.0-57088972.jar

hadoop-azure-3.3.1.5.0-57088972.jar

hadoop-azure-datalake-3.3.1.5.0-57088972.jar

hadoop-client-api-3.3.1.5.0-57088972.jar

hadoop-client-runtime-3.3.1.5.0-57088972.jar

hadoop-cloud-storage-3.3.1.5.0-57088972.jar

hadoop-cos-3.3.1.5.0-57088972.jar

hadoop-openstack-3.3.1.5.0-57088972.jar

hadoop-shaded-guava-1.1.0.jar

hadoop-yarn-server-web-proxy-3.3.1.5.0-57088972.jar

hdinsight-spark-metrics-3.2.0-1.0.0.jar

hive-beeline-2.3.9.jar

hive-cli-2.3.9.jar

hive-common-2.3.9.jar

hive-exec-2.3.9-core.jar

hive-jdbc-2.3.9.jar

hive-llap-common-2.3.9.jar

hive-metastore-2.3.9.jar

hive-serde-2.3.9.jar

hive-service-rpc-3.1.2.jar

hive-shims-0.23-2.3.9.jar

hive-shims-2.3.9.jar

hive-shims-common-2.3.9.jar

hive-shims-scheduler-2.3.9.jar

hive-storage-api-2.7.2.jar

hive-vector-code-gen-2.3.9.jar

hk2-api-2.6.1.jar

hk2-locator-2.6.1.jar

hk2-utils-2.6.1.jar

htrace-core4-4.1.0-incubating.jar

httpclient-4.5.13.jar

httpclient-4.5.6.jar

httpcore-4.4.14.jar

httpmime-4.5.13.jar

httpmime-4.5.6.jar

hyperspace-core-spark3.2_2.12-0.5.1-synapse.jar

impulse-core_spark3.2_2.12-0.1.8.jar

impulse-telemetry-mds_spark3.2_2.12-0.1.8.jar

isolation-forest_3.2.0_2.12-2.0.8.jar

istack-commons-runtime-3.0.8.jar

ivy-2.5.0.jar

jackson-annotations-2.12.3.jar

jackson-core-2.12.3.jar

jackson-core-asl-1.9.13.jar

jackson-databind-2.12.3.jar

jackson-dataformat-cbor-2.12.3.jar

jackson-mapper-asl-1.9.13.jar

jackson-module-scala_2.12-2.12.3.jar

jakarta.annotation-api-1.3.5.jar

jakarta.inject-2.6.1.jar

jakarta.servlet-api-4.0.3.jar

jakarta.validation-api-2.0.2.jar

jakarta.ws.rs-api-2.1.6.jar

jakarta.xml.bind-api-2.3.2.jar

janino-3.0.16.jar

javassist-3.25.0-GA.jar

javatuples-1.2.jar

javax.jdo-3.2.0-m3.jar

javolution-5.5.1.jar

jaxb-api-2.2.11.jar

jaxb-runtime-2.3.2.jar

jcl-over-slf4j-1.7.30.jar

jdo-api-3.0.1.jar

jdom-1.1.jar

jersey-client-2.34.jar

jersey-common-2.34.jar

jersey-container-servlet-2.34.jar

jersey-container-servlet-core-2.34.jar

jersey-hk2-2.34.jar

jersey-server-2.34.jar

jettison-1.1.jar

jetty-util-9.4.43.v20210629.jar

jetty-util-ajax-9.4.43.v20210629.jar

jline-2.14.6.jar

joda-time-2.10.10.jar

jodd-core-3.5.2.jar

jpam-1.1.jar

jsch-0.1.54.jar

json-1.8.jar

json-20090211.jar

json-20210307.jar

json-simple-1.1.jar

json4s-ast_2.12-3.7.0-M11.jar

json4s-core_2.12-3.7.0-M11.jar

json4s-jackson_2.12-3.7.0-M11.jar

json4s-scalap_2.12-3.7.0-M11.jar

jsr305-3.0.0.jar

jta-1.1.jar

jul-to-slf4j-1.7.30.jar

junit-jupiter-5.5.2.jar

junit-jupiter-api-5.5.2.jar

junit-jupiter-engine-5.5.2.jar

junit-jupiter-params-5.5.2.jar

junit-platform-commons-1.5.2.jar

junit-platform-engine-1.5.2.jar

kafka-clients-2.8.0.jar

kryo-shaded-4.0.2.jar

kusto-data-2.7.0.jar

kusto-ingest-2.7.0.jar

kusto-spark_3.0_2.12-2.7.5.jar

lapack-2.2.1.jar

leveldbjni-all-1.8.jar

libfb303-0.9.3.jar

libshufflejni.so

libthrift-0.12.0.jar

libvegasjni.so

lightgbmlib-3.2.110.jar

log4j-1.2.17.jar

lz4-java-1.7.1.jar

macro-compat_2.12-1.1.1.jar

mdsdclientdynamic-2.0.jar

metrics-core-4.2.0.jar

metrics-graphite-4.2.0.jar

metrics-jmx-4.2.0.jar

metrics-json-4.2.0.jar

metrics-jvm-4.2.0.jar

microsoft-catalog-metastore-client-1.0.63.jar

microsoft-log4j-etwappender-1.0.jar

microsoft-spark.jar

minlog-1.3.0.jar

mmlspark-1.0.0-rc3-194-14bef9b1-SNAPSHOT.jar

mmlspark-cognitive-1.0.0-rc3-194-14bef9b1-SNAPSHOT.jar

mmlspark-core-1.0.0-rc3-194-14bef9b1-SNAPSHOT.jar

mmlspark-deep-learning-1.0.0-rc3-194-14bef9b1-SNAPSHOT.jar

mmlspark-lightgbm-1.0.0-rc3-194-14bef9b1-SNAPSHOT.jar

mmlspark-opencv-1.0.0-rc3-194-14bef9b1-SNAPSHOT.jar

mmlspark-vw-1.0.0-rc3-194-14bef9b1-SNAPSHOT.jar

mssql-jdbc-8.4.1.jre8.jar

mysql-connector-java-8.0.18.jar

netty-all-4.1.68.Final.jar

notebook-utils-3.2.0-20220208.5.jar

objenesis-2.6.jar

onnxruntime_gpu-1.8.1.jar

opencsv-2.3.jar

opencv-3.2.0-1.jar

opentest4j-1.2.0.jar

orc-core-1.6.12.jar

orc-mapreduce-1.6.12.jar

orc-shims-1.6.12.jar

oro-2.0.8.jar

osgi-resource-locator-1.0.3.jar

paranamer-2.8.jar

parquet-column-1.12.2.jar

parquet-common-1.12.2.jar

parquet-encoding-1.12.2.jar

parquet-format-structures-1.12.2.jar

parquet-hadoop-1.12.2.jar

parquet-jackson-1.12.2.jar

peregrine-spark-0.10.jar

postgresql-42.2.9.jar

protobuf-java-2.5.0.jar

proton-j-0.33.8.jar

py4j-0.10.9.3.jar

pyrolite-4.30.jar

qpid-proton-j-extensions-1.2.4.jar

rapids-4-spark_2.12-22.02.0-SNAPSHOT.jar

rocksdbjni-6.20.3.jar

scala-collection-compat_2.12-2.1.1.jar

scala-compiler-2.12.15.jar

scala-java8-compat_2.12-0.9.0.jar

scala-library-2.12.15.jar

scala-parser-combinators_2.12-1.1.2.jar

scala-reflect-2.12.15.jar

scala-xml_2.12-1.2.0.jar

scalactic_2.12-3.0.5.jar

shapeless_2.12-2.3.3.jar

shims-0.9.0.jar

slf4j-api-1.7.30.jar

slf4j-log4j12-1.7.16.jar

snappy-java-1.1.8.4.jar

spark-3.2-rpc-history-server-app-listener_2.12-1.0.0.jar

spark-3.2-rpc-history-server-core_2.12-1.0.0.jar

spark-avro_2.12-3.2.1.5.0-57088972.jar

spark-catalyst_2.12-3.2.1.5.0-57088972.jar

spark-cdm-connector-assembly-1.19.2.jar

spark-core_2.12-3.2.1.5.0-57088972.jar

spark-enhancement_2.12-3.2.1.5.0-57088972.jar

spark-enhancementui_2.12-3.0.0.jar

spark-graphx_2.12-3.2.1.5.0-57088972.jar

spark-hadoop-cloud_2.12-3.2.1.5.0-57088972.jar

spark-hive-thriftserver_2.12-3.2.1.5.0-57088972.jar

spark-hive_2.12-3.2.1.5.0-57088972.jar

spark-kusto-synapse-connector_3.1_2.12-1.0.0.jar

spark-kvstore_2.12-3.2.1.5.0-57088972.jar

spark-launcher_2.12-3.2.1.5.0-57088972.jar

spark-microsoft-tools_2.12-3.2.1.5.0-57088972.jar

spark-mllib-local_2.12-3.2.1.5.0-57088972.jar

spark-mllib_2.12-3.2.1.5.0-57088972.jar

spark-mssql-connector-1.2.0.jar

spark-network-common_2.12-3.2.1.5.0-57088972.jar

spark-network-shuffle_2.12-3.2.1.5.0-57088972.jar

spark-repl_2.12-3.2.1.5.0-57088972.jar

spark-sketch_2.12-3.2.1.5.0-57088972.jar

spark-sql-kafka-0-10_2.12-3.2.1.5.0-57088972.jar

spark-sql_2.12-3.2.1.5.0-57088972.jar

spark-streaming-kafka-0-10-assembly_2.12-3.2.1.5.0-57088972.jar

spark-streaming-kafka-0-10_2.12-3.2.1.5.0-57088972.jar

spark-streaming_2.12-3.2.1.5.0-57088972.jar

spark-tags_2.12-3.2.1.5.0-57088972.jar

spark-token-provider-kafka-0-10_2.12-3.2.1.5.0-57088972.jar

spark-unsafe_2.12-3.2.1.5.0-57088972.jar

spark-yarn_2.12-3.2.1.5.0-57088972.jar

spark_diagnostic_cli-1.0.11_spark-3.2.0.jar

spire-macros_2.12-0.17.0.jar

spire-platform_2.12-0.17.0.jar

spire-util_2.12-0.17.0.jar

spire_2.12-0.17.0.jar

spray-json_2.12-1.3.2.jar

sqlanalyticsconnector_3.2.0-1.0.0.jar

stax-api-1.0.1.jar

stream-2.9.6.jar

structuredstreamforspark_2.12-3.0.1-2.1.3.jar

super-csv-2.2.0.jar

synapse-spark-telemetry_2.12-0.0.6.jar

synfs-3.2.0-20220208.5.jar

threeten-extra-1.5.0.jar

tink-1.6.0.jar

transaction-api-1.1.jar

univocity-parsers-2.9.1.jar

velocity-1.5.jar

vw-jni-8.9.1.jar

wildfly-openssl-1.0.7.Final.jar

xbean-asm9-shaded-4.20.jar

xz-1.8.jar

zookeeper-3.6.2.5.0-57088972.jar

zookeeper-jute-3.6.2.5.0-57088972.jar

zstd-jni-1.5.0-4.jar

Python libraries (Normal VMs)

_libgcc_mutex=0.1

_openmp_mutex=4.5 _py-xgboost-mutex=2.0

abseil-cpp=20210324.0

absl-py=0.13.0

adal=1.2.7

adlfs=0.7.7

aiohttp=3.7.4.post0

alsa-lib=1.2.3

appdirs=1.4.4

arrow-cpp=3.0.0

astor=0.8.1

astunparse=1.6.3

async-timeout=3.0.1

attrs=21.2.0

aws-c-cal=0.5.11

aws-c-common=0.6.2

aws-c-event-stream=0.2.7

aws-c-io=0.10.5

aws-checksums=0.1.11

aws-sdk-cpp=1.8.186

azure-datalake-store=0.0.51

azure-identity=2021.03.15b1

azure-storage-blob=12.8.1

backcall=0.2.0

backports=1.0

backports.functools_lru_cache=1.6.4

beautifulsoup4=4.9.3

blas=2.109

blas-devel=3.9.0

blinker=1.4

blosc=1.21.0

bokeh=2.3.2

brotli=1.0.9

brotli-bin=1.0.9

brotli-python=1.0.9

brotlipy=0.7.0

brunsli=0.1

bzip2=1.0.8

c-ares=1.17.1

ca-certificates=2021.7.5

cachetools=4.2.2

cairo=1.16.0

certifi=2021.5.30

cffi=1.14.5

chardet=4.0.0

charls=2.2.0

click=8.0.1

cloudpickle=1.6.0

conda=4.9.2

conda-package-handling=1.7.3

configparser=5.0.2

cryptography=3.4.7

cudatoolkit=11.1.1

cycler=0.10.0

cython=0.29.23

cytoolz=0.11.0

dash=1.20.0

dash-core-components=1.16.0

dash-html-components=1.1.3

dash-renderer=1.9.1

dash-table=4.11.3

dash_cytoscape=0.2.0

dask-core=2021.6.2

databricks-cli=0.12.1

dataclasses=0.8

dbus=1.13.18

debugpy=1.3.0

decorator=4.4.2

dill=0.3.4

entrypoints=0.3

et_xmlfile=1.1.0

expat=2.4.1

fire=0.4.0

flask=2.0.1

flask-compress=1.10.1

fontconfig=2.13.1

freetype=2.10.4

fsspec=2021.6.1

future=0.18.2

gast=0.3.3

gensim=3.8.3

geographiclib=1.52

geopy=2.1.0

gettext=0.21.0

gevent=21.1.2

gflags=2.2.2

giflib=5.2.1

gitdb=4.0.7

gitpython=3.1.18

glib=2.68.3

glib-tools=2.68.3

glog=0.5.0

gobject-introspection=1.68.0

google-auth=1.32.1

google-auth-oauthlib=0.4.1

google-pasta=0.2.0

greenlet=1.1.0

grpc-cpp=1.37.1

grpcio=1.37.1

gst-plugins-base=1.18.4

gstreamer=1.18.4

h5py=2.10.0

hdf5=1.10.6

html5lib=1.1

hummingbird-ml=0.4.0

icu=68.1=h58526e2_0

idna=2.10

imagecodecs=2021.3.31

imageio=2.9.0

importlib-metadata=4.6.1

intel-openmp=2021.2.0

interpret=0.2.4

interpret-core=0.2.4

ipykernel=6.0.1

ipython=7.23.1

ipython_genutils=0.2.0

isodate=0.6.0

itsdangerous=2.0.1

jdcal=1.4.1

jedi=0.18.0

jinja2=3.0.1

joblib=1.0.1

jpeg=9d

jupyter_client=6.1.12

jupyter_core=4.7.1

jxrlib=1.1

keras-applications=1.0.8

keras-preprocessing=1.1.2

keras2onnx=1.6.5

kiwisolver=1.3.1

koalas=1.8.0

krb5=1.19.1

lcms2=2.12

ld_impl_linux-64=2.36.1

lerc=2.2.1

liac-arff=2.5.0

libaec=1.0.5

libblas=3.9.0

libbrotlicommon=1.0.9

libbrotlidec=1.0.9

libbrotlienc=1.0.9

libcblas=3.9.0

libclang=11.1.0

libcurl=7.77.0

libdeflate=1.7

libedit=3.1.20210216

libev=4.33

libevent=2.1.10

libffi=3.3

libgcc-ng=9.3.0

libgfortran-ng=9.3.0

libgfortran5=9.3.0

libglib=2.68.3

libiconv=1.16

liblapack=3.9.0

liblapacke=3.9.0

libllvm10=10.0.1

libllvm11=11.1.0

libnghttp2=1.43.0

libogg=1.3.5

libopus=1.3.1

libpng=1.6.37

libpq=13.3

libprotobuf=3.15.8

libsodium=1.0.18

libssh2=1.9.0

libstdcxx-ng=9.3.0

libthrift=0.14.1

libtiff=4.2.0

libutf8proc=2.6.1

libuuid=2.32.1000

libuv=1.41.1

libvorbis=1.3.7

libwebp-base=1.2.0

libxcb=1.14

libxgboost=1.4.0

libxkbcommon=1.0.3

libxml2=2.9.12

libzopfli=1.0.3

lightgbm=3.2.1

lime=0.2.0.1

llvm-openmp=11.1.0

llvmlite=0.36.0

locket=0.2.1

lz4-c=1.9.3

markdown=3.3.4

markupsafe=2.0.1

matplotlib=3.4.2

matplotlib-base=3.4.2

matplotlib-inline=0.1.2

mkl=2021.2.0

mkl-devel=2021.2.0

mkl-include=2021.2.0

mleap=0.17.0

mlflow-skinny=1.18.0

msal=2021.06.08

msal-extensions=2021.06.08

msrest=2021.06.01

multidict=5.1.0

mysql-common=8.0.25

mysql-libs=8.0.25

ncurses=6.2

networkx=2.5.1

ninja=1.10.2

nltk=3.6.2

nspr=4.30

nss=3.67

numba=0.53.1

numpy=1.19.4

oauthlib=3.1.1

olefile=0.46

onnx=1.9.0

onnxconverter-common=1.7.0

onnxmltools=1.7.0

onnxruntime=1.7.2

openjpeg=2.4.0

openpyxl=3.0.7

openssl=1.1.1k

opt_einsum=3.3.0

orc=1.6.7=heec2584_1

packaging=21.0

pandas=1.2.3

parquet-cpp=1.5.1

parso=0.8.2

partd=1.2.0

patsy=0.5.1

pcre=8.45

pexpect=4.8.0

pickleshare=0.7.5

pillow=8.2.0

pip=21.1.1

pixman=0.40.0

plotly=4.14.3

pmdarima=1.8.2

pooch=1.4.0

portalocker=1.7.1

prompt-toolkit=3.0.19

protobuf=3.15.8

psutil=5.8.0

ptyprocess=0.7.0

py-xgboost=1.4.0

py4j=0.10.9

pyarrow=3.0.0

pyasn1=0.4.8

pyasn1-modules=0.2.8

pycairo=1.20.1

pycosat=0.6.3

pycparser=2.20

pygments=2.9.0

pygobject=3.40.1

pyjwt=2.1.0

pyodbc=4.0.30

pyopenssl=20.0.1

pyparsing=2.4.7

pyqt=5.12.3

pyqt-impl=5.12.3

pyqt5-sip=4.19.18

pyqtchart=5.12

pyqtwebengine=5.12.1

pysocks=1.7.1

python=3.8.10

python-dateutil=2.8.1

python-flatbuffers=1.12

python_abi=3.8

pytorch=1.8.1

pytz=2021.1

pyu2f=0.1.5

pywavelets=1.1.1

pyyaml=5.4.1

pyzmq=22.1.0

qt=5.12.9

re2=2021.04.01

readline=8.1

regex=2021.7.6

requests=2.25.1

requests-oauthlib=1.3.0

retrying=1.3.3

rsa=4.7.2

ruamel_yaml=0.15.100

s2n=1.0.10

salib=1.3.11

scikit-image=0.18.1

scikit-learn=0.23.2

scipy=1.5.3

seaborn=0.11.1

seaborn-base=0.11.1

setuptools=49.6.0

shap=0.39.0

six=1.16.0

skl2onnx=1.8.0.1

sklearn-pandas=2.2.0

slicer=0.0.7

smart_open=5.1.0

smmap=3.0.5

snappy=1.1.8

soupsieve=2.2.1

sqlite=3.36.0

statsmodels=0.12.2

tabulate=0.8.9

tenacity=7.0.0

tensorboard=2.4.1

tensorboard-plugin-wit=1.8.0

tensorflow=2.4.1

tensorflow-base=2.4.1

tensorflow-estimator=2.4.0

termcolor=1.1.0

textblob=0.15.3

threadpoolctl=2.1.0

tifffile=2021.4.8

tk=8.6.10

toolz=0.11.1

tornado=6.1

tqdm=4.61.2

traitlets=5.0.5

typing-extensions=3.10.0.0

typing_extensions=3.10.0.0

unixodbc=2.3.9

urllib3=1.26.4

wcwidth=0.2.5

webencodings=0.5.1

werkzeug=2.0.1

wheel=0.36.2

wrapt=1.12.1

xgboost=1.4.0

xorg-kbproto=1.0.7002

xorg-libice=1.0.10

xorg-libsm=1.2.3

xorg-libx11=1.7.2

xorg-libxext=1.3.4

xorg-libxrender=0.9.10003

xorg-renderproto=0.11.1002

xorg-xextproto=7.3.0002

xorg-xproto=7.0.31007

xz=5.2.5

yaml=0.2.5

yarl=1.6.3

zeromq=4.3.4

zfp=0.5.5

zipp=3.5.0

zlib=1.2.11010

zope.event=4.5.0

zope.interface=5.4.0

zstd=1.4.9

azure-common==1.1.27

azure-core==1.16.0

azure-graphrbac==0.61.1

azure-mgmt-authorization==0.61.0

azure-mgmt-containerregistry==8.0.0

azure-mgmt-core==1.3.0

azure-mgmt-keyvault==2.2.0

azure-mgmt-resource==13.0.0

azure-mgmt-storage==11.2.0

azureml-core==1.34.0

azureml-mlflow==1.34.0

azureml-opendatasets==1.34.0

backports-tempfile==1.0

backports-weakref==1.0.post1

contextlib2==0.6.0.post1

docker==4.4.4

ipywidgets==7.6.3

jeepney==0.6.0

jmespath==0.10.0

jsonpickle==2.0.0

kqlmagiccustom==0.1.114.post8

lxml==4.6.5

msrestazure==0.6.4

mypy==0.780

mypy-extensions==0.4.3

ndg-httpsclient==0.5.1

pandasql==0.7.3

pathspec==0.8.1

prettytable==2.4.0

pyperclip==1.8.2

ruamel-yaml==0.17.4

ruamel-yaml-clib==0.2.6

secretstorage==3.3.1

sqlalchemy==1.4.20

typed-ast==1.4.3

torchvision==0.9.1

websocket-client==1.1.0

Python libraries (GPU Accelerated VMs)

_libgcc_mutex=0.1

_openmp_mutex=4.5

_py-xgboost-mutex=2.0

_tflow_select=2.3.0

abseil-cpp=20210324.1

absl-py=0.13.0

adal=1.2.7

adlfs=0.7.7

aiohttp=3.7.4.post0

appdirs=1.4.4

arrow-cpp=3.0.0

astor=0.8.1

astunparse=1.6.3

async-timeout=3.0.1

attrs=21.2.0

aws-c-cal=0.5.11

aws-c-common=0.6.2

aws-c-event-stream=0.2.7

aws-c-io=0.10.5

aws-checksums=0.1.11

aws-sdk-cpp=1.8.186

azure-datalake-store=0.0.51

azure-storage-blob=12.9.0

backcall=0.2.0

backports=1.0

backports.functools_lru_cache=1.6.4

beautifulsoup4=4.9.3

blas=2.111

blas-devel=3.9.0

blinker=1.4

bokeh=2.3.2

brotli=1.0.9

brotli-bin=1.0.9

brotli-python=1.0.9

brotlipy=0.7.0

bzip2=1.0.8

c-ares=1.17.2

ca-certificates=2021.5.30

cachetools=4.2.2

cairo=1.16.0

certifi=2021.5.30

cffi=1.14.6

chardet=4.0.0

click=8.0.1

colorama=0.4.4

conda=4.9.2

conda-package-handling=1.7.3

configparser=5.0.2

cryptography=3.4.7

cudatoolkit=11.1.1

cycler=0.10.0

cython=0.29.24

cytoolz=0.11.0

dash=1.21.0

dash-core-components=1.17.1

dash-html-components=1.1.4

dash-renderer=1.9.1

dash-table=4.12.0

dash_cytoscape=0.2.0

dask-core=2021.9.1

databricks-cli=0.12.1

dataclasses=0.8

dbus=1.13.18

debugpy=1.4.1

decorator=5.1.0

dill=0.3.4

entrypoints=0.3

et_xmlfile=1.0.1001

expat=2.4.1

ffmpeg=4.3

fire=0.4.0

flask=2.0.1

flask-compress=1.10.1

fontconfig=2.13.1

freetype=2.10.4

fsspec=2021.8.1

future=0.18.2

g-ir-build-tools=1.68.0

g-ir-host-tools=1.68.0

gensim=3.8.3

geographiclib=1.52

geopy=2.1.0

gettext=0.19.8.1

gevent=21.8.0

gflags=2.2.2

gitdb=4.0.7

gitpython=3.1.23

glib=2.68.4

glib-tools=2.68.4

glog=0.5.0

gmp=6.2.1

gnutls=3.6.13

gobject-introspection=1.68.0

google-auth=1.35.0

google-auth-oauthlib=0.4.6

google-pasta=0.2.0

greenlet=1.1.1

grpc-cpp=1.37.1

gst-plugins-base=1.14.0

gstreamer=1.14.0

h5py=2.10.0

hdf5=1.10.6

html5lib=1.1

hummingbird-ml=0.4.0

icu=58.2

idna=2.10

imagecodecs-lite=2019.12.3

imageio=2.9.0

importlib-metadata=4.8.1

interpret=0.2.4

interpret-core=0.2.4

ipykernel=6.4.1

ipython=7.23.1

ipython_genutils=0.2.0

isodate=0.6.0

itsdangerous=2.0.1

jdcal=1.4.1

jedi=0.18.0

jinja2=3.0.1

joblib=1.0.1

jpeg=9b

jupyter_client=7.0.3

jupyter_core=4.8.1

keras=2.4.3

keras-applications=1.0.8

keras-preprocessing=1.1.2

keras2onnx=1.6.5

kiwisolver=1.3.2

koalas=1.8.0

krb5=1.19.2

lame=3.100

ld_impl_linux-64=2.36.1

liac-arff=2.5.0

libblas=3.9.0

libbrotlicommon=1.0.9

libbrotlidec=1.0.9

libbrotlienc=1.0.9

libcblas=3.9.0

libcurl=7.79.1

libedit=3.1.20191231

libev=4.33

libevent=2.1.10

libffi=3.4.2

libgcc-ng=11.2.0

libgfortran-ng=11.2.0

libgfortran5=11.2.0

libgirepository=1.68.0

libglib=2.68.4

libiconv=1.16

liblapack=3.9.0

liblapacke=3.9.0

libllvm11=11.1.0

libnghttp2=1.43.0

libpng=1.6.37

libprotobuf=3.16.0

libsodium=1.0.18

libssh2=1.10.0

libstdcxx-ng=11.2.0

libthrift=0.14.1

libtiff=4.2.0

libutf8proc=2.6.1

libuuid=2.32.1

libuv=1.42.0

libwebp-base=1.2.1

libxcb=1.13

libxgboost=1.4.0

libxml2=2.9.9

lightgbm=3.2.1

lime=0.2.0.1

llvm-openmp=12.0.1

llvmlite=0.37.0

locket=0.2.0

lz4-c=1.9.3

markdown=3.3.4

markupsafe=2.0.1

matplotlib=3.4.2

matplotlib-base=3.4.2

matplotlib-inline=0.1.3

mkl=2021.3.0

mkl-devel=2021.3.0

mkl-include=2021.3.0

mleap=0.17.0

mlflow-skinny=1.18.0

msal=2021.09.01

msrest=2021.09.01

multidict=5.1.0

multiprocess=0.70.12.2

ncurses=6.2

nest-asyncio=1.5.1

nettle=3.6

networkx=2.5

ninja=1.10.2

nltk=3.6.2

numba=0.54.0

numpy=1.19.4

oauthlib=3.1.1

olefile=0.46

onnx=1.9.0

onnxconverter-common=1.7.0

onnxmltools=1.7.0

onnxruntime=1.7.2

openh264=2.1.1

openpyxl=3.0.7

openssl=1.1.1l

opt_einsum=3.3.0

orc=1.6.7

packaging=21.0

pandas=1.2.3

parquet-cpp=1.5.1

parso=0.8.2

partd=1.2.0

pathos=0.2.8

patsy=0.5.1

pcre=8.45

pexpect=4.8.0

pickleshare=0.7.5003

pillow=7.1.2

pip=21.1.1

pixman=0.38.0

pkg-config=0.29.2

plotly=4.14.3

pmdarima=1.8.2

pooch=1.5.1

portalocker=1.7.1

pox=0.3.0

ppft=1.6.6.4

prompt-toolkit=3.0.20

protobuf=3.16.0

psutil=5.8.0

pthread-stubs=0.4

ptyprocess=0.7.0

py-xgboost=1.4.0

py4j=0.10.9

pyarrow=3.0.0

pyasn1=0.4.8

pyasn1-modules=0.2.7

pycairo=1.20.1

pycosat=0.6.3

pycparser=2.20

pydeprecate=0.3.1

pygments=2.10.0

pygobject=3.40.1

pyjwt=2.1.0

pyodbc=4.0.30

pyopenssl=20.0.1

pyparsing=2.4.7

pyqt=5.9.2

pysocks=1.7.1

pyspark=3.1.2

python=3.8.12

python-dateutil=2.8.2

python_abi=3.8

pytorch=1.8.1

pytorch-lightning=1.4.2

pytz=2021.1

pyu2f=0.1.5

pywavelets=1.1.1

pyyaml=5.4.1

pyzmq=22.3.0

qt=5.9.7

re2=2021.04.01

readline=8.1

regex=2021.8.28

requests=2.25.1

requests-oauthlib=1.3.0

retrying=1.3.3

rsa=4.7.2

ruamel_yaml=0.15.80

s2n=1.0.10

salib=1.4.5

scikit-image=0.18.1

scikit-learn=0.23.2

scipy=1.7.1

seaborn=0.11.1

seaborn-base=0.11.1

setuptools=49.6.0

shap=0.39.0

sip=4.19.13

skl2onnx=1.8.0.1

sklearn-pandas=2.2.0

slicer=0.0.7

smart_open=5.2.1

smmap=3.0.5

snappy=1.1.8

soupsieve=2.0.1

sqlite=3.36.0

statsmodels=0.12.2

tabulate=0.8.9

tbb=2021.3.0

tenacity=8.0.1

tensorboard=2.6.0

tensorboard-data-server=0.6.0

tensorboard-plugin-wit=1.8.0

tensorflow=2.4.1

tensorflow-base=2.4.1

termcolor=1.1.0

textblob=0.15.3

threadpoolctl=2.2.0

tifffile=2020.6.3

tk=8.6.11

toolz=0.11.1

torchaudio=0.8.1

torchmetrics=0.5.1

torchvision=0.9.1

tornado=6.1

tqdm=4.62.3

traitlets=5.1.0

unixodbc=2.3.9

urllib3=1.26.4

wcwidth=0.2.5

webencodings=0.5.1

werkzeug=2.0.1

wheel=0.37.0

wrapt=1.12.1

xgboost=1.4.0

xorg-kbproto=1.0.7

xorg-libice=1.0.10

xorg-libsm=1.2.3

xorg-libx11=1.7.2

xorg-libxau=1.0.9

xorg-libxdmcp=1.1.3

xorg-libxext=1.3.4

xorg-libxrender=0.9.10

xorg-renderproto=0.11.1

xorg-xextproto=7.3.0

xorg-xproto=7.0.31

xz=5.2.5

yaml=0.2.5

yarl=1.6.3

zeromq=4.3.4

zipp=3.5.0

zlib=1.2.11

zope.event=4.5.0

zope.interface=5.4.0

zstd=1.4.9

applicationinsights==0.11.10

argon2-cffi==21.1.0

azure-common==1.1.27

azure-core==1.18.0

azure-graphrbac==0.61.1

azure-identity==1.4.1

azure-mgmt-authorization==0.61.0

azure-mgmt-containerregistry==8.1.0

azure-mgmt-core==1.3.0

azure-mgmt-keyvault==9.1.0

azure-mgmt-resource==13.0.0

azure-mgmt-storage==11.2.0

azureml-core==1.34.0

azureml-dataprep==2.22.2

azureml-dataprep-native==38.0.0

azureml-dataprep-rslex==1.20.2

azureml-dataset-runtime==1.34.0

azureml-mlflow==1.34.0

azureml-opendatasets==1.34.0

azureml-telemetry==1.34.0

backports-tempfile==1.0

backports-weakref==1.0.post1

bleach==4.1.0

cloudpickle==1.6.0

contextlib2==21.6.0

defusedxml==0.7.1

diskcache==5.2.1

distro==1.6.0

docker==5.0.2

dotnetcore2==2.1.21

flatbuffers==1.12

fusepy==3.0.1

gast==0.3.3

grpcio==1.32.0

ipywidgets==7.6.5

jeepney==0.7.1

jmespath==0.10.0

jsonpickle==2.0.0

jsonschema==4.1.2

jupyterlab-pygments==0.1.2

jupyterlab-widgets==1.0.2

kqlmagiccustom==0.1.114.post8

lxml==4.7.1

mistune==0.8.4

msal-extensions==0.2.2

msrestazure==0.6.4

mypy==0.780

mypy-extensions==0.4.3

nbclient==0.5.4

nbconvert==6.2.0

nbformat==5.1.3

ndg-httpsclient==0.5.1

notebook==6.4.5

opencv-python==4.5.3.56

pandasql==0.7.3

pandocfilters==1.5.0

pathspec==0.9.0

petastorm==0.11.3

prettytable==3.0.0

prometheus-client==0.12.0

pyperclip==1.8.2

pyrsistent==0.18.0

ruamel-yaml==0.17.4

ruamel-yaml-clib==0.2.6

secretstorage==3.3.1

send2trash==1.8.0

six==1.15.0

sqlalchemy==1.4.25

tensorflow-estimator==2.4.0

tensorflow-gpu==2.4.1

terminado==0.12.1

testpath==0.5.0

typed-ast==1.4.3

typing-extensions==3.7.4.3

websocket-client==1.2.1

widgetsnbextension==3.5.2

Next steps

- Azure Synapse Analytics
- Apache Spark Documentation
- Apache Spark Concepts

Azure Synapse Runtime for Apache Spark 3.1

8/19/2022 • 3 minutes to read • Edit Online

Azure Synapse Analytics supports multiple runtimes for Apache Spark. This document will cover the runtime components and versions for the Azure Synapse Runtime for Apache Spark 3.1.

Component versions

COMPONENT	VERSION
Apache Spark	3.1.2
Operating System	Ubuntu 18.04
Java	1.8.0_282
Scala	2.12.10
Hadoop	3.1.1
.NET Core	3.1
.NET	2.0.0
Delta Lake	1.0
Python	3.8

Scala and Java libraries

HikariCP-2.5.1.jar

JLargeArrays-1.5.jar

JTransforms-3.1.jar

RoaringBitmap-0.9.0.jar

ST4-4.0.4.jar

SparkCustomEvents_3.1.2-1.0.0.jar

TokenLibrary-assembly-1.0.jar

VegasConnector-1.0.25.1_2.12.jar

accessors-smart-1.2.jar

activation-1.1.1.jar

adal4j-1.6.3.jar

aircompressor-0.10.jar

algebra_2.12-2.0.0-M2.jar

animal-sniffer-annotations-1.17.jar

antlr-runtime-3.5.2.jar

antlr4-runtime-4.8-1.jar

aopalliance-1.0.jar

aopalliance-repackaged-2.6.1.jar

apache-log4j-extras-1.2.17.jar

arpack_combined_all-0.1.jar

arrow-format-2.0.0.jar

arrow-memory-core-2.0.0.jar

arrow-memory-netty-2.0.0.jar

arrow-vector-2.0.0.jar

avro-1.8.2.jar

avro-ipc-1.8.2.jar

avro-mapred-1.8.2-hadoop2.jar

aws-java-sdk-bundle-1.11.375.jar

azure-eventhubs-3.3.0.jar

azure-eventhubs-spark_2.12-2.3.21.jar

azure-keyvault-core-1.0.0.jar

azure-storage-7.0.1.jar

azure-synapse-ml-pandas_2.12-1.0.0.jar

azure-synapse-ml-predict_2.12-1.0.jar

bcpkix-jdk15on-1.60.jar

bcprov-jdk15on-1.60.jar

bonecp-0.8.0.RELEASE.jar

breeze-macros_2.12-1.0.jar

breeze_2.12-1.0.jar

cats-kernel_2.12-2.0.0-M4.jar

checker-qual-2.8.1.jar

chill-java-0.9.5.jar

chill_2.12-0.9.5.jar

client-sdk-1.14.0.jar

cntk-2.4.jar

commons-beanutils-1.9.4.jar

commons-cli-1.2.jar

commons-codec-1.10.jar

commons-collections-3.2.2.jar

commons-compiler-3.0.16.jar

commons-compress-1.20.jar

commons-configuration2-2.1.1.jar

commons-crypto-1.1.0.jar

commons-daemon-1.0.13.jar

commons-dbcp-1.4.jar

commons-httpclient-3.1.jar

commons-io-2.5.jar

commons-lang-2.6.jar

commons-lang3-3.10.jar

commons-logging-1.1.3.jar

commons-math3-3.4.1.jar

commons-net-3.1.jar

commons-pool-1.5.4.jar

commons-pool2-2.6.2.jar

commons-text-1.6.jar

compress-lzf-1.0.3.jar

config-1.3.4.jar

core-1.1.2.jar

cosmos-analytics-spark-connector_3-1_2-12-assembly-3.0.4.jar

cudf-21.10.0-cuda11.jar

curator-client-2.12.0.jar

curator-framework-2.12.0.jar

curator-recipes-2.12.0.jar

datanucleus-api-jdo-4.2.4.jar

datanucleus-core-4.1.6.jar

datanucleus-rdbms-4.1.19.jar

delta-core_2.12-1.0.0.2b.jar

derby-10.12.1.1.jar

dnsjava-2.1.7.jar

dropwizard-metrics-hadoop-metrics2-reporter-0.1.2.jar

ehcache-3.3.1.jar

error_prone_annotations-2.3.2.jar

failureaccess-1.0.1.jar

flatbuffers-java-1.9.0.jar

fluent-logger-jar-with-dependencies.jar

geronimo-jcache_1.0_spec-1.0-alpha-1.jar

gson-2.2.4.jar

guava-28.0-jre.jar

guice-4.0.jar

guice-servlet-4.0.jar

hadoop-annotations-3.1.1.5.0-50849917.jar

hadoop-auth-3.1.1.5.0-50849917.jar

hadoop-aws-3.1.1.5.0-50849917.jar

hadoop-azure-3.1.1.5.0-50849917.jar

hadoop-client-3.1.1.5.0-50849917.jar

hadoop-common-3.1.1.5.0-50849917.jar

hadoop-hdfs-client-3.1.1.5.0-50849917.jar

hadoop-mapreduce-client-common-3.1.1.5.0-50849917.jar

hadoop-mapreduce-client-core-3.1.1.5.0-50849917.jar

hadoop-mapreduce-client-jobclient-3.1.1.5.0-50849917.jar

hadoop-openstack-3.1.1.5.0-50849917.jar

hadoop-yarn-api-3.1.1.5.0-50849917.jar

hadoop-yarn-client-3.1.1.5.0-50849917.jar

hadoop-yarn-common-3.1.1.5.0-50849917.jar

hadoop-yarn-registry-3.1.1.5.0-50849917.jar

hadoop-yarn-server-common-3.1.1.5.0-50849917.jar

hadoop-yarn-server-web-proxy-3.1.1.5.0-50849917.jar

hdinsight-spark-metrics_3_1_2-1.0.0.jar

hive-beeline-2.3.7.jar

hive-cli-2.3.7.jar

hive-common-2.3.7.jar

hive-exec-2.3.7-core.jar

hive-jdbc-2.3.7.jar

hive-llap-common-2.3.7.jar

hive-metastore-2.3.7.jar

hive-serde-2.3.7.jar

hive-service-rpc-3.1.2.jar

hive-shims-0.23-2.3.7.jar

hive-shims-2.3.7.jar

hive-shims-common-2.3.7.jar

hive-shims-scheduler-2.3.7.jar

hive-storage-api-2.7.2.jar

hive-vector-code-gen-2.3.7.jar

hk2-api-2.6.1.jar

hk2-locator-2.6.1.jar

hk2-utils-2.6.1.jar

htrace-core4-4.1.0-incubating.jar

httpclient-4.5.6.jar

httpcore-4.4.12.jar

httpmime-4.5.6.jar

hyperspace-core-spark3.1_2.12-0.5.0-synapse.jar

isolation-forest_3.0.0_2.12-1.0.1.jar

istack-commons-runtime-3.0.8.jar

ivy-2.4.0.jar

j2objc-annotations-1.3.jar

jackson-annotations-2.10.0.jar

jackson-core-2.10.0.jar

jackson-core-asl-1.9.13.jar

jackson-databind-2.10.0.jar

jackson-dataformat-cbor-2.10.0.jar

jackson-jaxrs-base-2.10.0.jar

jackson-jaxrs-json-provider-2.10.0.jar

jackson-mapper-asl-1.9.13.jar

jackson-module-jaxb-annotations-2.10.0.jar

jackson-module-paranamer-2.10.0.jar

jackson-module-scala_2.12-2.10.0.jar

jakarta.activation-api-1.2.1.jar

jakarta.annotation-api-1.3.5.jar

jakarta.inject-2.6.1.jar

jakarta.servlet-api-4.0.3.jar

jakarta.validation-api-2.0.2.jar

jakarta.ws.rs-api-2.1.6.jar

jakarta.xml.bind-api-2.3.2.jar

janino-3.0.16.jar

javassist-3.25.0-GA.jar

javatuples-1.2.jar

javax.inject-1.jar

javax.jdo-3.2.0-m3.jar

javolution-5.5.1.jar

jaxb-api-2.2.11.jar

jaxb-runtime-2.3.2.jar

jcip-annotations-1.0-1.jar

jcl-over-slf4j-1.7.30.jar

jdo-api-3.0.1.jar

jersey-client-2.30.jar

jersey-common-2.30.jar

jersey-container-servlet-2.30.jar

jersey-container-servlet-core-2.30.jar

jersey-entity-filtering-2.30.jar

jersey-hk2-2.30.jar

jersey-media-jaxb-2.30.jar

jersey-media-json-jackson-2.30.jar

jersey-server-2.30.jar

jetty-util-9.4.40.v20210413.jar

jetty-util-ajax-9.4.40.v20210413.jar

jline-2.14.6.jar

joda-time-2.10.5.jar

jodd-core-3.5.2.jar

jpam-1.1.jar

jsch-0.1.54.jar

json-1.8.jar

json-20090211.jar

json-simple-1.1.jar

json-smart-2.3.jar

json4s-ast_2.12-3.7.0-M5.jar

json4s-core_2.12-3.7.0-M5.jar

json4s-jackson_2.12-3.7.0-M5.jar

json4s-scalap_2.12-3.7.0-M5.jar

jsp-api-2.1.jar

jsr305-3.0.0.jar

jta-1.1.jar

jul-to-slf4j-1.7.30.jar

kafka-clients-2.4.1.5.0-50849917.jar

kerb-admin-1.0.1.jar

kerb-client-1.0.1.jar

kerb-common-1.0.1.jar

kerb-core-1.0.1.jar

kerb-crypto-1.0.1.jar

kerb-identity-1.0.1.jar

kerb-server-1.0.1.jar

kerb-simplekdc-1.0.1.jar

kerb-util-1.0.1.jar

kerby-asn1-1.0.1.jar

kerby-config-1.0.1.jar

kerby-pkix-1.0.1.jar

kerby-util-1.0.1.jar

kerby-xdr-1.0.1.jar

kryo-shaded-4.0.2.jar

kusto-data-2.7.0.jar

kusto-ingest-2.7.0.jar

kusto-spark_3.0_2.12-2.7.5.jar

leveldbjni-all-1.8.jar

libfb303-0.9.3.jar

libthrift-0.12.0.jar

libvegasjni.so

lightgbmlib-3.2.110.jar

listenablefuture-9999.0-empty-to-avoid-conflict-with-guava.jar

log4j-1.2.17.jar

lz4-java-1.7.1.jar

machinist_2.12-0.6.8.jar

macro-compat_2.12-1.1.1.jar

mdsdclientdynamic-2.0.jar

metrics-core-4.1.1.jar

metrics-graphite-4.1.1.jar

metrics-jmx-4.1.1.jar

metrics-json-4.1.1.jar

metrics-jvm-4.1.1.jar

microsoft-catalog-metastore-client-1.0.57.jar

microsoft-log4j-etwappender-1.0.jar

microsoft-spark.jar

minlog-1.3.0.jar

mmlspark-1.0.0-rc3-179-327be83c-SNAPSHOT.jar

mmlspark-cognitive-1.0.0-rc3-179-327be83c-SNAPSHOT.jar

mmlspark-core-1.0.0-rc3-179-327be83c-SNAPSHOT.jar

mmlspark-deep-learning-1.0.0-rc3-179-327be83c-SNAPSHOT.jar

mmlspark-lightgbm-1.0.0-rc3-179-327be83c-SNAPSHOT.jar

mmlspark-opencv-1.0.0-rc3-179-327be83c-SNAPSHOT.jar

mmlspark-vw-1.0.0-rc3-179-327be83c-SNAPSHOT.jar

mssql-jdbc-8.4.1.jre8.jar

mysql-connector-java-8.0.18.jar

netty-all-4.1.51.Final.jar

nimbus-jose-jwt-4.41.1.jar

notebook-utils-3.0.0-20211110.6.jar

objenesis-2.6.jar

okhttp-2.7.5.jar

okio-1.14.0.jar

onnxruntime_gpu-1.8.1.jar

opencsv-2.3.jar

opencv-3.2.0-1.jar

orc-core-1.5.12.jar

orc-mapreduce-1.5.12.jar

orc-shims-1.5.12.jar

oro-2.0.8.jar

osgi-resource-locator-1.0.3.jar

paranamer-2.8.jar

parquet-column-1.10.1.jar

parquet-common-1.10.1.jar

parquet-encoding-1.10.1.jar

parquet-format-2.4.0.jar

parquet-hadoop-1.10.1.jar

parquet-jackson-1.10.1.jar

peregrine-spark-0.9.jar

postgresql-42.2.9.jar

protobuf-java-2.5.0.jar

proton-j-0.33.8.jar

py4j-0.10.9.jar

pyrolite-4.30.jar

qpid-proton-j-extensions-1.2.4.jar

rapids-4-spark_2.12-21.10.0.jar

re2j-1.1.jar

scala-collection-compat_2.12-2.1.1.jar

scala-compiler-2.12.10.jar

scala-java8-compat_2.12-0.9.0.jar

scala-library-2.12.10.jar

scala-parser-combinators_2.12-1.1.2.jar

scala-reflect-2.12.10.jar

scala-xml_2.12-1.2.0.jar

scalactic_2.12-3.0.5.jar

shapeless_2.12-2.3.3.jar

shims-0.9.0.jar

slf4j-api-1.7.30.jar

slf4j-log4j12-1.7.16.jar

snappy-java-1.1.8.2.jar

spark-3.1-rpc-history-server-app-listener_2.12-1.0.0.jar

spark-3.1-rpc-history-server-core_2.12-1.0.0.jar

spark-avro_2.12-3.1.2.5.0-50849917.jar

spark-catalyst_2.12-3.1.2.5.0-50849917.jar

spark-cdm-connector-assembly-1.19.2.jar

spark-core_2.12-3.1.2.5.0-50849917.jar

spark-enhancement_2.12-3.1.2.5.0-50849917.jar

spark-enhancementui_2.12-1.1.0.jar

spark-graphx_2.12-3.1.2.5.0-50849917.jar

spark-hadoop-cloud_2.12-3.1.2.5.0-50849917.jar

spark-hive-thriftserver_2.12-3.1.2.5.0-50849917.jar

spark-hive_2.12-3.1.2.5.0-50849917.jar

spark-kusto-synapse-connector_3.1_2.12-1.0.0.jar

spark-kvstore_2.12-3.1.2.5.0-50849917.jar

spark-launcher_2.12-3.1.2.5.0-50849917.jar

spark-microsoft-telemetry_2.12-3.1.2.5.0-50849917.jar

spark-microsoft-tools_2.12-3.1.2.5.0-50849917.jar

spark-mllib-local_2.12-3.1.2.5.0-50849917.jar

spark-mllib_2.12-3.1.2.5.0-50849917.jar

spark-mssql-connector-1.2.0.jar

spark-network-common_2.12-3.1.2.5.0-50849917.jar

spark-network-shuffle_2.12-3.1.2.5.0-50849917.jar

spark-repl_2.12-3.1.2.5.0-50849917.jar

spark-sketch_2.12-3.1.2.5.0-50849917.jar

spark-sql-kafka-0-10_2.12-3.1.2.5.0-50849917.jar

spark-sql_2.12-3.1.2.5.0-50849917.jar

spark-streaming-kafka-0-10-assembly_2.12-3.1.2.5.0-50849917.jar

spark-streaming-kafka-0-10_2.12-3.1.2.5.0-50849917.jar

spark-streaming_2.12-3.1.2.5.0-50849917.jar

spark-tags_2.12-3.1.2.5.0-50849917.jar

spark-token-provider-kafka-0-10_2.12-3.1.2.5.0-50849917.jar

spark-unsafe_2.12-3.1.2.5.0-50849917.jar

spark-yarn_2.12-3.1.2.5.0-50849917.jar

spark_diagnostic_cli-1.0.10_spark-3.1.2.jar

spire-macros_2.12-0.17.0-M1.jar

spire-platform_2.12-0.17.0-M1.jar

spire-util_2.12-0.17.0-M1.jar

spire_2.12-0.17.0-M1.jar

spray-json_2.12-1.3.2.jar

sqlanalyticsconnector_3.1.2-1.0.1.jar

stax-api-1.0.1.jar

stax2-api-3.1.4.jar

stream-2.9.6.jar

structuredstreamforspark_2.12-3.0.1-2.1.3.jar

super-csv-2.2.0.jar

synapse-spark-telemetry_2.12-0.0.6.jar

synfs-3.0.0-20211110.6.jar

threeten-extra-1.5.0.jar

token-provider-1.0.1.jar

transaction-api-1.1.jar

univocity-parsers-2.9.1.jar

velocity-1.5.jar

vw-jni-8.9.1.jar

wildfly-openssl-1.0.7.Final.jar

woodstox-core-5.0.3.jar

xbean-asm7-shaded-4.15.jar

xz-1.5.jar

zookeeper-3.4.6.5.0-50849917.jar

zstd-jni-1.4.8-1.jar

Python libraries

_openmp_mutex=4.5

_py-xgboost-mutex=2.0

abseil-cpp=20210324.0

absl-py=0.13.0

adal=1.2.7

adlfs=0.7.7

aiohttp=3.7.4.post0

alsa-lib=1.2.3

appdirs=1.4.4

arrow-cpp=3.0.0

astor=0.8.1

astunparse=1.6.3

async-timeout=3.0.1

attrs=21.2.0

azure-datalake-store=0.0.51

azure-identity=2021.03.15b1

azure-storage-blob=12.8.1

backcall=0.2.0

backports=1.0

backports.functools_lru_cache=1.6.4

beautifulsoup4=4.9.3

blas=2.109

blas-devel=3.9.0

blinker=1.4

blosc=1.21.0

bokeh=2.3.2

brotli=1.0.9

brotli-bin=1.0.9

brotli-python=1.0.9

brotlipy=0.7.0

brunsli=0.1

bzip2=1.0.8

c-ares=1.17.1

ca-certificates=2021.7.5

cachetools=4.2.2

cairo=1.16.0

certifi=2021.5.30

cffi=1.14.5

chardet=4.0.0

charls=2.2.0

click=8.0.1

cloudpickle=1.6.0

conda=4.9.2

conda-package-handling=1.7.3

configparser=5.0.2

cryptography=3.4.7

cudatoolkit=11.1.1

cycler=0.10.0

cython=0.29.23

cytoolz=0.11.0

dash=1.20.0

dash-core-components=1.16.0

dash-html-components=1.1.3

dash-renderer=1.9.1

dash-table=4.11.3

dash_cytoscape=0.2.0

dask-core=2021.6.2

databricks-cli=0.12.1

dataclasses=0.8

dbus=1.13.18

debugpy=1.3.0

decorator=4.4.2

dill=0.3.4

entrypoints=0.3

et_xmlfile=1.1.0

expat=2.4.1

fire=0.4.0

flask=2.0.1

flask-compress=1.10.1

fontconfig=2.13.1

freetype=2.10.4

fsspec=2021.6.1

future=0.18.2

gast=0.3.3

gensim=3.8.3

geographiclib=1.52

geopy=2.1.0

gettext=0.21.0

gevent=21.1.2

gflags=2.2.2

giflib=5.2.1

gitdb=4.0.7

gitpython=3.1.18

glib=2.68.3

glib-tools=2.68.3

glog=0.5.0

gobject-introspection=1.68.0

google-auth=1.32.1

google-auth-oauthlib=0.4.1

google-pasta=0.2.0

greenlet=1.1.0

grpc-cpp=1.37.1

grpcio=1.37.1

gst-plugins-base=1.18.4

gstreamer=1.18.4

h5py=2.10.0

hdf5=1.10.6

html5lib=1.1

hummingbird-ml=0.4.0

icu=68.1

idna=2.10

imagecodecs=2021.3.31

imageio=2.9.0

importlib-metadata=4.6.1

intel-openmp=2021.2.0

interpret=0.2.4

interpret-core=0.2.4

ipykernel=6.0.1

ipython=7.23.1

ipython_genutils=0.2.0

isodate=0.6.0

itsdangerous=2.0.1

jdcal=1.4.1

jedi=0.18.0

jinja2=3.0.1

joblib=1.0.1

jpeg=9d

jupyter_client=6.1.12

jupyter_core=4.7.1

jxrlib=1.1

keras-applications=1.0.8

keras-preprocessing=1.1.2

keras2onnx=1.6.5

kiwisolver=1.3.1

koalas=1.8.0

krb5=1.19.1

lcms2=2.12

ld_impl_linux-64=2.36.1

lerc=2.2.1

liac-arff=2.5.0

libaec=1.0.5

libblas=3.9.0

libbrotlicommon=1.0.9

libbrotlidec=1.0.9

libbrotlienc=1.0.9

libcblas=3.9.0

libclang=11.1.0

libcurl=7.77.0

libdeflate=1.7

libedit=3.1.20210216

libev=4.33

libevent=2.1.10

libffi=3.3

libgcc-ng=9.3.0

libgfortran-ng=9.3.0

libgfortran5=9.3.0

libglib=2.68.3

libiconv=1.16

liblapack=3.9.0

liblapacke=3.9.0

libllvm10=10.0.1

libllvm11=11.1.0

libnghttp2=1.43.0

libogg=1.3.5

libopus=1.3.1

libpng=1.6.37

libpq=13.3

libprotobuf=3.15.8

libsodium=1.0.18

libssh2=1.9.0

libstdcxx-ng=9.3.0

libthrift=0.14.1

libtiff=4.2.0

libutf8proc=2.6.1

libuuid=2.32.1

libuv=1.41.1

libvorbis=1.3.7

libwebp-base=1.2.0

libxcb=1.14

libxgboost=1.4.0

libxkbcommon=1.0.3

libxml2=2.9.12

libzopfli=1.0.3

lightgbm=3.2.1

lime=0.2.0.1

llvm-openmp=11.1.0

llvmlite=0.36.0

locket=0.2.1

lz4-c=1.9.3

markdown=3.3.4

markupsafe=2.0.1

matplotlib=3.4.2

matplotlib-base=3.4.2

matplotlib-inline=0.1.2

mkl=2021.2.0

mkl-devel=2021.2.0

mkl-include=2021.2.0

mleap=0.17.0

mlflow-skinny=1.18.0

msal=2021.06.08

msal-extensions=2021.06.08

msrest=2021.06.01

multidict=5.1.0

mysql-common=8.0.25

mysql-libs=8.0.25

ncurses=6.2

networkx=2.5.1

ninja=1.10.2

nltk=3.6.2

nspr=4.30

nss=3.67

numba=0.53.1

numpy=1.19.4

oauthlib=3.1.1

olefile=0.46

onnx=1.9.0

onnxconverter-common=1.7.0

onnxmltools=1.7.0

onnxruntime=1.7.2

openjpeg=2.4.0

openpyxl=3.0.7

openssl=1.1.1k

opt_einsum=3.3.0

orc=1.6.7

packaging=21.0

pandas=1.2.3

parquet-cpp=1.5.1

parso=0.8.2

partd=1.2.0

patsy=0.5.1

pcre=8.45

pexpect=4.8.0

pickleshare=0.7.5

pillow=8.2.0

pip=21.1.1

pixman=0.40.0

plotly=4.14.3

pmdarima=1.8.2

pooch=1.4.0

portalocker=1.7.1

prompt-toolkit=3.0.19

protobuf=3.15.8

psutil=5.8.0

ptyprocess=0.7.0

py-xgboost=1.4.0

py4j=0.10.9

pyarrow=3.0.0

pyasn1=0.4.8

pyasn1-modules=0.2.8

pycairo=1.20.1

pycosat=0.6.3

pycparser=2.20

pygments=2.9.0

pygobject=3.40.1

pyjwt=2.1.0

pyodbc=4.0.30

pyopenssl=20.0.1

pyparsing=2.4.7

pyqt=5.12.3

pyqt-impl=5.12.3

pyqt5-sip=4.19.18

pyqtchart=5.12

pyqtwebengine=5.12.1

pysocks=1.7.1

pyspark=3.1.2

python=3.8.10

python-dateutil=2.8.1

python-flatbuffers=1.12

python_abi=3.8

pytorch=1.8.1

pytz=2021.1

pyu2f=0.1.5

pywavelets=1.1.1

pyyaml=5.4.1

pyzmq=22.1.0

qt=5.12.9

re2=2021.04.01

readline=8.1

regex=2021.7.6

requests=2.25.1

requests-oauthlib=1.3.0

retrying=1.3.3

rsa=4.7.2

ruamel_yaml=0.15.100

s2n=1.0.10

salib=1.3.11

scikit-image=0.18.1

scikit-learn=0.23.2

scipy=1.5.3

seaborn=0.11.1

seaborn-base=0.11.1

setuptools=49.6.0

shap=0.39.0

six=1.16.0

skl2onnx=1.8.0.1

sklearn-pandas=2.2.0

slicer=0.0.7

smart_open=5.1.0

smmap=3.0.5

snappy=1.1.8

soupsieve=2.2.1

sqlite=3.36.0

statsmodels=0.12.2

tabulate=0.8.9

tenacity=7.0.0

tensorboard=2.4.1

tensorboard-plugin-wit=1.8.0

tensorflow=2.4.1

tensorflow-base=2.4.1

tensorflow-estimator=2.4.0

termcolor=1.1.0

textblob=0.15.3

threadpoolctl=2.1.0

tifffile=2021.4.8

tk=8.6.10

toolz=0.11.1

tornado=6.1

tqdm=4.61.2

traitlets=5.0.5

typing-extensions=3.10.0.0

typing_extensions=3.10.0.0

unixodbc=2.3.9

urllib3=1.26.4

wcwidth=0.2.5

webencodings=0.5.1

werkzeug=2.0.1

wheel=0.36.2

wrapt=1.12.1

xgboost=1.4.0

xorg-kbproto=1.0.7

xorg-libice=1.0.10

xorg-libsm=1.2.3

xorg-libx11=1.7.2

xorg-libxext=1.3.4

xorg-libxrender=0.9.10

xorg-renderproto=0.11.1

xorg-xextproto=7.3.0

xorg-xproto=7.0.31

xz=5.2.5

yaml=0.2.5

yarl=1.6.3

zeromq=4.3.4

zfp=0.5.5

zipp=3.5.0

zlib=1.2.11

zope.event=4.5.0

zope.interface=5.4.0

zstd=1.4.9

azure-common=1.1.27

azure-core=1.16.0

azure-graphrbac=0.61.1

azure-mgmt-authorization=0.61.0

azure-mgmt-containerregistry=8.0.0

azure-mgmt-core=1.3.0

azure-mgmt-keyvault=2.2.0

azure-mgmt-resource=13.0.0

azure-mgmt-storage=11.2.0

azureml-core=1.34.0

azureml-mlflow=1.34.0

azureml-opendatasets=1.34.0

backports-tempfile=1.0

backports-weakref=1.0.post1

contextlib2=0.6.0.post1

docker=4.4.4

ipywidgets=7.6.3

jeepney=0.6.0

jmespath=0.10.0

jsonpickle=2.0.0

msrestazure=0.6.4

mypy=0.780

mypy-extensions=0.4.3

ndg-httpsclient=0.5.1

pandasql=0.7.3

pathspec=0.8.1

ruamel-yaml=0.17.4

ruamel-yaml-clib=0.2.6

secretstorage=3.3.1

sqlalchemy=1.4.20

typed-ast=1.4.3

torchvision=0.9.1

websocket-client=1.1.0

Next steps

- Azure Synapse Analytics
- Apache Spark Documentation
- Apache Spark Concepts

Azure Synapse Runtime for Apache Spark 2.4

8/19/2022 • 3 minutes to read • Edit Online

Azure Synapse Analytics supports multiple runtimes for Apache Spark. This document will cover the runtime components and versions for the Azure Synapse Runtime for Apache Spark 2.4.

Component versions

COMPONENT	VERSION
Apache Spark	2.4.8
Operating System	Ubuntu 16.04
Java	1.8.0_272
Scala	2.11
.NET Core	3.1
.NET	1.0.0
Delta Lake	0.6
Python	3.6

Scala and Java libraries

HikariCP-java7-2.4.12.jar

JavaEWAH-0.3.2.jar

RoaringBitmap-0.7.45.jar

ST4-4.0.4.jar

SparkCustomEvents-2.0.0.jar

TokenLibrary-assembly-1.0.jar

VegasConnector-1.0.21.1.jar

accessors-smart-1.2.jar

activation-1.1.1.jar

aircompressor-0.10.jar

antlr-2.7.7.jar

antlr-runtime-3.4.jar

antlr4-runtime-4.7.jar

aopalliance-1.0.jar

aopalliance-repackaged-2.4.0-b34.jar

apache-log4j-extras-1.2.17.jar

apacheds-i18n-2.0.0-M15.jar

apacheds-kerberos-codec-2.0.0-M15.jar

api-asn1-api-1.0.0-M20.jar

api-util-1.0.0-M20.jar

arpack_combined_all-0.1.jar

arrow-format-0.10.0.jar

arrow-memory-0.10.0.jar

arrow-vector-0.10.0.jar

audience-annotations-0.5.0.jar

avro-1.8.2.jar

avro-ipc-1.8.2.jar

avro-mapred-1.8.2-hadoop2.jar

aws-java-sdk-bundle-1.11.271.jar

azure-eventhubs-3.2.2.jar

azure-eventhubs-spark_2.11-2.3.18.jar

azure-keyvault-core-1.0.0.jar

azure-storage-7.0.1.jar

bonecp-0.8.0.RELEASE.jar

breeze-macros_2.11-0.13.2.jar

breeze_2.11-0.13.2.jar

calcite-avatica-1.2.0-incubating.jar

calcite-core-1.2.0-incubating.jar

calcite-linq4j-1.2.0-incubating.jar

cardinalestimate_2.11-0.1-uber.jar

chill-java-0.9.3.jar

chill_2.11-0.9.3.jar

client-sdk-1.11.0.jar

cntk-2.4.jar

commons-beanutils-1.9.4.jar

commons-cli-1.2.jar

commons-codec-1.10.jar

commons-collections-3.2.2.jar

commons-compiler-3.0.9.jar

commons-compress-1.8.1.jar

commons-configuration-1.6.jar

commons-crypto-1.0.0.jar

commons-dbcp-1.4.jar

commons-digester-1.8.jar

commons-httpclient-3.1.jar

commons-io-2.4.jar

commons-lang-2.6.jar

commons-lang3-3.5.jar

commons-logging-1.1.3.jar

commons-math3-3.4.1.jar

commons-net-3.1.jar

commons-pool-1.5.4.jar

compress-lzf-1.0.3.jar

config-1.3.4.jar

core-1.1.2.jar

cosmos-analytics-spark-connector-assembly-1.4.5.jar

curator-client-2.7.1.jar

curator-framework-2.7.1.jar

curator-recipes-2.7.1.jar

datanucleus-api-jdo-3.2.6.jar

datanucleus-core-3.2.10.jar

datanucleus-rdbms-3.2.9.jar

delta-core_2.11-0.6.1.1.jar

derby-10.12.1.1.jar

ehcache-3.3.1.jar

eigenbase-properties-1.1.5.jar

flatbuffers-1.2.0-3f79e055.jar

fluent-logger-jar-with-dependencies.jar

geronimo-jcache_1.0_spec-1.0-alpha-1.jar

gson-2.2.4.jar

guava-14.0.1.jar

guice-3.0.jar

guice-servlet-3.0.jar

hadoop-annotations-2.9.1.2.6.99.201-34744923.jar

hadoop-auth-2.9.1.2.6.99.201-34744923.jar

hadoop-aws-2.9.1.2.6.99.201-34744923.jar

hadoop-azure-2.9.1.2.6.99.201-34744923.jar

hadoop-client-2.9.1.2.6.99.201-34744923.jar

hadoop-common-2.9.1.2.6.99.201-34744923.jar

hadoop-hdfs-client-2.9.1.2.6.99.201-34744923.jar

hadoop-mapreduce-client-app-2.9.1.2.6.99.201-34744923.jar

hadoop-mapreduce-client-common-2.9.1.2.6.99.201-34744923.jar

hadoop-mapreduce-client-core-2.9.1.2.6.99.201-34744923.jar

hadoop-mapreduce-client-jobclient-2.9.1.2.6.99.201-34744923.jar

hadoop-mapreduce-client-shuffle-2.9.1.2.6.99.201-34744923.jar

hadoop-openstack-2.9.1.2.6.99.201-34744923.jar

hadoop-yarn-api-2.9.1.2.6.99.201-34744923.jar

hadoop-yarn-client-2.9.1.2.6.99.201-34744923.jar

hadoop-yarn-common-2.9.1.2.6.99.201-34744923.jar

hadoop-yarn-registry-2.9.1.2.6.99.201-34744923.jar

hadoop-yarn-server-common-2.9.1.2.6.99.201-34744923.jar

hadoop-yarn-server-web-proxy-2.9.1.2.6.99.201-34744923.jar

hdinsight-spark-metrics_2_4-2.1.jar

hive-beeline-1.2.1.spark2.jar

hive-cli-1.2.1.spark2.jar

hive-exec-1.2.1.spark2.jar

hive-jdbc-1.2.1.spark2.jar

hive-metastore-1.2.1.spark2.jar

hk2-api-2.4.0-b34.jar

hk2-locator-2.4.0-b34.jar

hk2-utils-2.4.0-b34.jar

hppc-0.7.2.jar

htrace-core4-4.1.0-incubating.jar

httpclient-4.5.6.jar

httpcore-4.4.10.jar

hyperspace-core_2.11-0.4.0.1.jar

impulse-core_2.11-0.1.2.jar

isolation-forest_2.4.3_2.11-0.3.2.jar

ivy-2.4.0.jar

jackson-annotations-2.6.7.jar

jackson-core-2.6.7.jar

jackson-core-asl-1.9.13.jar

jackson-databind-2.6.7.3.jar

jackson-jaxrs-1.9.13.jar

jackson-mapper-asl-1.9.13.jar

jackson-module-paranamer-2.7.9.jar

jackson-module-scala_2.11-2.6.7.1.jar

jackson-xc-1.9.13.jar

janino-3.0.9.jar

javassist-3.18.1-GA.jar

javax.annotation-api-1.2.jar

javax.inject-1.jar

javax.inject-2.4.0-b34.jar

javax.servlet-api-3.1.0.jar

javax.ws.rs-api-2.0.1.jar

javolution-5.5.1.jar

jaxb-api-2.2.2.jar

jcip-annotations-1.0-1.jar

jcl-over-slf4j-1.7.16.jar

jdo-api-3.0.1.jar

jersey-client-2.22.2.jar

jersey-common-2.22.2.jar

jersey-container-servlet-2.22.2.jar

jersey-container-servlet-core-2.22.2.jar

jersey-guava-2.22.2.jar

jersey-media-jaxb-2.22.2.jar

jersey-server-2.22.2.jar

jetty-6.1.26.jar

jetty-sslengine-6.1.26.jar

jetty-util-6.1.26.jar

jline-2.14.6.jar

joda-time-2.9.3.jar

jodd-core-3.5.2.jar

jpam-1.1.jar

jsch-0.1.54.jar

json-20090211.jar

json-simple-1.1.jar

json-smart-2.3.jar

json4s-ast_2.11-3.5.3.jar

json4s-core_2.11-3.5.3.jar

json4s-jackson_2.11-3.5.3.jar

json4s-scalap_2.11-3.5.3.jar

jsp-api-2.1.jar

jsr305-1.3.9.jar

jta-1.1.jar

jtransforms-2.4.0.jar

jul-to-slf4j-1.7.16.jar

kryo-shaded-4.0.2.jar

kusto-data-2.4.1.jar

kusto-ingest-2.4.1.jar

kusto-spark_2.4_2.11-2.5.1.jar

leveldbjni-all-1.8.jar

libfb303-0.9.3.jar

libthrift-0.9.3.jar

libuxsock.so

lightgbmlib-2.3.180.jar

log4j-1.2.17.jar

lz4-java-1.4.0.jar

machinist_2.11-0.6.1.jar

macro-compat_2.11-1.1.1.jar

mdsdclientdynamic-2.0.jar

metrics-core-3.1.5.jar

metrics-graphite-3.1.5.jar

metrics-json-3.1.5.jar

metrics-jvm-3.1.5.jar

microsoft-catalog-metastore-client-1.0.44.jar

microsoft-log4j-etwappender-1.0.jar

microsoft-spark.jar

minlog-1.3.0.jar

mmlspark_2.11-1.0.0-rc3-6-0a30d1ae-SNAPSHOT.jar

mssql-jdbc-8.4.1.jre8.jar

mysql-connector-java-8.0.18.jar

netty-3.9.9.Final.jar

netty-all-4.1.42.Final.jar

nimbus-jose-jwt-4.41.1.jar

notebook-utils-2.4.4-20210430.2.jar

objenesis-2.5.1.jar

okhttp-2.7.5.jar

okio-1.6.0.jar

opencsv-2.3.jar

opencv-3.2.0-1.jar

orc-core-1.5.5-nohive.jar

orc-mapreduce-1.5.5-nohive.jar

orc-shims-1.5.5.jar

oro-2.0.8.jar

osgi-resource-locator-1.0.1.jar

paranamer-2.8.jar

parquet-column-1.10.1.jar

parquet-common-1.10.1.jar

parquet-encoding-1.10.1.jar

parquet-format-2.4.0.jar

parquet-hadoop-1.10.1.jar

parquet-hadoop-bundle-1.6.0.jar

parquet-jackson-1.10.1.jar

peregrine-spark-0.5.jar

postgresql-42.2.9.jar

protobuf-java-2.5.0.jar

proton-j-0.33.4.jar

py4j-0.10.7.jar

pyrolite-4.13.jar

qpid-proton-j-extensions-1.2.3.jar

scala-compiler-2.11.12.jar

scala-java8-compat_2.11-0.9.0.jar

scala-library-2.11.12.jar

scala-parser-combinators_2.11-1.1.0.jar

scala-reflect-2.11.12.jar

scala-xml_2.11-1.0.5.jar

shapeless_2.11-2.3.2.jar

shims-0.7.45.jar

slf4j-api-1.7.16.jar

slf4j-log4j12-1.7.16.jar

snappy-0.2.jar

snappy-java-1.1.7.3.jar

spark-avro_2.11-2.4.4.2.6.99.201-34744923.jar

spark-catalyst_2.11-2.4.4.2.6.99.201-34744923.jar

spark-cdm-connector-assembly-0.19.1.jar

spark-core_2.11-2.4.4.2.6.99.201-34744923.jar

spark-enhancement_2.11-2.4.4.2.6.99.201-34744923.jar

spark-enhancementui_2.11-1.0.0.jar

spark-graphx_2.11-2.4.4.2.6.99.201-34744923.jar

spark-hive-thriftserver_2.11-2.4.4.2.6.99.201-34744923.jar

spark-hive_2.11-2.4.4.2.6.99.201-34744923.jar

spark-kusto-synapse-connector-0.11.0.jar

spark-kvstore_2.11-2.4.4.2.6.99.201-34744923.jar

spark-launcher_2.11-2.4.4.2.6.99.201-34744923.jar

spark-microsoft-telemetry_2.11-2.4.4.2.6.99.201-34744923.jar

spark-microsoft-tools_2.11-2.4.4.2.6.99.201-34744923.jar

spark-mllib-local_2.11-2.4.4.2.6.99.201-34744923.jar

spark-mllib_2.11-2.4.4.2.6.99.201-34744923.jar

spark-mssql-connector-1.0.0.jar

spark-network-common_2.11-2.4.4.2.6.99.201-34744923.jar

spark-network-shuffle_2.11-2.4.4.2.6.99.201-34744923.jar

spark-repl_2.11-2.4.4.2.6.99.201-34744923.jar

spark-sketch_2.11-2.4.4.2.6.99.201-34744923.jar

spark-sql_2.11-2.4.4.2.6.99.201-34744923.jar

spark-streaming_2.11-2.4.4.2.6.99.201-34744923.jar

spark-tags_2.11-2.4.4.2.6.99.201-34744923.jar

spark-unsafe_2.11-2.4.4.2.6.99.201-34744923.jar

spark-yarn_2.11-2.4.4.2.6.99.201-34744923.jar

spark_diagnostic_cli-1.0.3_spark-2.4.5.jar

spire-macros_2.11-0.13.0.jar

spire_2.11-0.13.0.jar

spray-json_2.11-1.3.2.jar

sqlanalyticsconnector-1.0.9.2.6.99.201-34744923.jar

stax-api-1.0-2.jar

stax-api-1.0.1.jar

stax2-api-3.1.4.jar

stream-2.7.0.jar

stringtemplate-3.2.1.jar

structuredstreamforspark_2.11-1.3.2.jar

super-csv-2.2.0.jar

synapse-spark-telemetry_2.11-0.0.4.jar

univocity-parsers-2.7.3.jar

validation-api-1.1.0.Final.jar

vw-jni-8.8.1.jar

wildfly-openssl-1.0.7.Final.jar

woodstox-core-5.0.3.jar

xbean-asm6-shaded-4.8.jar

xmlenc-0.52.jar

xz-1.5.jar

zookeeper-3.4.11.2.6.99.201-34744923.jar

zstd-jni-1.3.2-2.jar

Python libraries

_libgcc_mutex=0.1

_openmp_mutex=4.5

c-ares=1.16.1

ca-certificates=2020.6.20

certifi=2020.6.20

cffi=1.14.3

chardet=3.0.4

cryptography=3.1.1

conda=4.3.21

conda-env=2.6.0

cytoolz=0.8.2

gperftools=2.7

h5py=2.10.0

hdf5=1.10.6

jpeg=9d

krb5=1.17.1

ld_impl_linux-64=2.35

libblas=3.9.0

libcblas=3.9.0

libcurl=7.71.1

libedit=3.1.20191231

libev=4.33

libffi=3.2.1

libgcc-ng=9.3.0

libgfortran-ng=9.3.0

libgfortran4=7.5.0

libgfortran5=9.3.0

libgomp=9.3.0

libiconv=1.16

liblapack=3.9.0

libnghttp2=1.41.0

libopenblas=0.3.12

libssh2=1.9.0

libstdcxx-ng=9.3.0

numpy=1.18.5

ncurses=6.2

openssl=1.1.1

perl=5.32.0

pip=20.2.4

pygments=2.7.3

pyopenssl=19.1.0

python=3.6.11

python_abi=3.6

readline=8.0

requests=2.24.0

sentencepiece=0.1.9

setuptools=41.4.0

six=1.15.0

sqlite=3.33.0

tk=8.6.1

toolz=0.11.1

urllib3=1.25.10

unixodbc=2.3.9

xz=5.2.5

wheel=0.30.0

yaml=0.2.5

zlib=1.2.11

absl-py=0.11.0

adal=1.2.4

adlfs=0.5.5

aiohttp=3.7.2

alembic=1.4.1

altair=4.1.0

appdirs=1.4.4

applicationinsights=0.11.9

asn1crypto=1.4.0

astor=0.8.1

astroid=2.4.2

astunparse=1.6.3

async-timeout=3.0.1

attrs=20.3.0

azure-common=1.1.25

azure-core=1.8.2

azure-datalake-store=0.0.51

azure-graphrbac=0.61.1

azure-identity=1.4.1

azure-mgmt-authorization=0.61.0

azure-mgmt-containerregistry=2.8.0

azure-mgmt-keyvault=2.2.0

azure-mgmt-resource=12.0.0

azure-mgmt-storage=11.2.0

azure-storage-blob=12.5.0

azure-storage-common=2.1.0

azure-storage-queue=12.1.5

azureml-automl-core=1.22.0

azureml-automl-runtime=1.22.0

azureml-core=1.22.0

azureml-dataprep=2.9.0

azureml-dataprep-native=29.0.0

azureml-dataprep-rslex=1.7.0

azureml-dataset-runtime=1.22.0

azureml-defaults=1.22.0

azureml-interpret=1.22.0

azureml-mlflow=1.16.0

azureml-model-management-sdk=1.0.1b6.post1

azureml-opendatasets=1.18.0

azureml-pipeline=1.22.0

azureml-pipeline-core=1.22.0

azureml-pipeline-steps=1.22.0

azureml-sdk=1.22.0

azure-storage-blob=12.5.0

azureml-telemetry=1.22.0

azureml-train=1.22.0

azureml-train-automl=1.22.0

azureml-train-automl-client=1.22.0.post1

azureml-train-automl-runtime=1.22.0.post1

azureml-train-core=1.22.0

azureml-train-restclients-hyperdrive=1.22.0

backports.tempfile=1.0

backports.weakref=1.0.post1

beautifulsoup4=4.9.3

bitarray=1.6.1

bokeh=2.2.3

boto=2.49.0

boto3=1.15.14

botocore=1.18.14

Bottleneck=1.3.2

bpemb=0.3.2

cachetools=4.1.1

certifi=2020.6.20

click=7.1.2

cloudpickle=1.6.0

configparser=3.7.4

contextlib2=0.6.0.post1

cycler=0.10.0

cython=0.29.21

cytoolz=0.8.2

databricks-cli=0.14.0

dataclasses=0.8

datashape=0.5.2

decorator=4.4.2

Deprecated=1.2.10

dill=0.3.2

distro=1.5.0

docker=4.3.1

docutils=0.16

dotnetcore2=2.1.17

entrypoints=0.3

et-xmlfile=1.0.1

filelock=3.0.12

fire=0.3.1

flair=0.5

Flask=1.0.3

fsspec=0.8.4

fusepy=3.0.1

future=0.18.2

gast=0.3.3

gensim=3.8.3

geographiclib=1.50

geopy=2.0.0

gitdb=4.0.5

GitPython=3.1.11

google-auth=1.23.0

google-auth-oauthlib=0.4.2

google-pasta=0.2.0

gorilla=0.3.0

grpcio=1.33.2

gunicorn=19.9.0

html5lib=1.1

hummingbird-ml=0.0.6

hyperopt=0.2.5

idna=2.10

idna-ssl=1.1.0

imageio=2.9.0

importlib-metadata=1.7.0

interpret-community=0.16.0

interpret-core=0.2.1

ipykernel=5.5.3

ipython=7.8.0

ipython-genutils=0.2.0

ipywidgets=7.6.3

isodate=0.6.0

isort=5.6.4

itsdangerous=1.1.0

jdcal=1.4.1

jeepney=0.4.3

Jinja2=2.11.2

jmespath=0.10.0

joblib=0.14.1

json-logging-py=0.2

jsonpickle=1.4.1

jsonschema=3.2.0

Keras-Applications=1.0.8

Keras-Preprocessing=1.1.2

keras2onnx=1.6.0

kiwisolver=1.3.1

koalas=1.2.0

langdetect=1.0.8

lazy-object-proxy=1.4.3

liac-arff=2.5.0

lightgbm=2.3.0

Mako=1.1.3

Markdown=3.3.3

MarkupSafe=1.1.1

matplotlib=3.2.2

mccabe=0.6.1

mistune=0.8.4

mleap=0.16.1

mlflow=1.11.0

more-itertools=8.6.0

mpld3=0.3

mpmath=1.1.0

msal=1.5.0

msal-extensions=0.2.2

msrest=0.6.19

msrestazure=0.6.4

multidict=5.0.0

multipledispatch=0.6.0

mypy=0.780

mypy-extensions=0.4.3

ndg-httpsclient=0.5.1

networkx=2.5

nimbusml=1.7.1

nltk=3.5

nose=1.3.7

oauthlib=3.1.0

odo=0.5.0

olefile=0.46

onnx=1.6.0

onnxconverter-common=1.6.0

onnxmltools=1.4.1

onnxruntime=1.3.0

openpyxl=3.0.5

opt-einsum=3.3.0

packaging=20.4

pandas=0.25.3

pandasql=0.7.3

pathspec=0.8.0

patsy=0.5.1

pickleshare=0.7.5

Pillow=8.0.1

plotly=4.12.0

pluggy=0.13.1

pmdarima=1.1.1

portalocker=1.7.1

prometheus-client=0.8.0

prometheus-flask-exporter=0.18.1

protobuf=3.13.0

psutil=5.7.2

py=1.9.0

py-cpuinfo=5.0.0

py4j=0.10.7

pyarrow=1.0.1

pyasn1=0.4.8

pyasn1-modules=0.2.8

pycrypto=2.6.1

pycparser=2.20

PyJWT=1.7.1

pylint=2.6.0

pymssql=2.1.5

pyodbc=4.0.30

pyopencl=2020.1

pyparsing=2.4.7

pyrsistent=0.17.3

pyspark=2.4.5

pytest=5.3.2

python-dateutil=2.8.1

python-editor=1.0.4

pytools=2021.1

pytz=2020.1

PyWavelets=1.1.1

PyYAML=5.3.1

querystring-parser=1.2.4

regex=2020.10.28

requests-oauthlib=1.3.0

retrying=1.3.3

rsa=4.6

ruamel.yaml=0.16.12

ruamel.yaml.clib=0.2.2

s3transfer=0.3.3

sacremoses=0.0.43

scikit-image=0.17.2

scikit-learn=0.22.2.post1

scipy=1.5.2

seaborn=0.11.0

SecretStorage=3.1.2

segtok=1.5.10

shap=0.34.0

skl2onnx=1.4.9

sklearn-pandas=1.7.0

smart-open=1.9.0

smmap=3.0.4

soupsieve=2.0.1

SQLAlchemy=1.3.13

sqlitedict=1.7.0

sqlparse=0.4.1

statsmodels=0.10.2

tabulate=0.8.7

tb-nightly=1.14.0a20190603

tensorboard=2.3.0

tensorboard-plugin-wit=1.7.0

tensorflow=2.0.0b1

tensorflow-estimator=2.3.0

termcolor=1.1.0

textblob=0.15.3

tf-estimator-nightly=1.14.0.dev2019060501

tf2onnx=1.7.2

tifffile=2020.9.3

tokenizers=0.9.2

toml=0.10.2

torch=1.7.0

tornado=6.1

tqdm=4.48.2

transformers=3.4.0

typed-ast=1.4.1

typing-extensions=3.7.4.3

urllib3=1.25.10

wcwidth=0.2.5

webencodings=0.5.1

websocket-client=0.57.0

Werkzeug=1.0.1

wheel=0.30.0

wrapt=1.11.2

xgboost=0.90

zict=1.0.0

zipp=0.6.0

Next steps

- Azure Synapse Analytics
- Apache Spark Documentation
- Apache Spark Concepts

Synapse runtime for Apache Spark lifecycle and supportability

8/19/2022 • 3 minutes to read • Edit Online

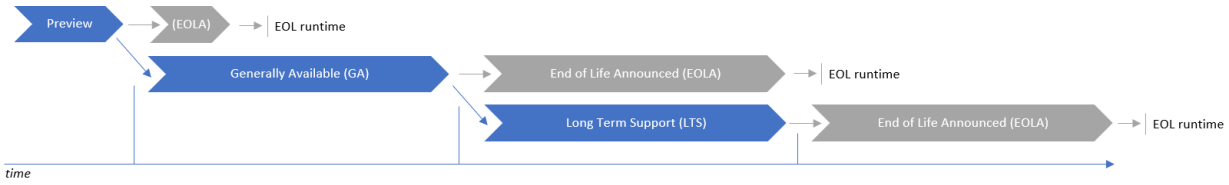
Apache Spark pools in Azure Synapse use runtimes to tie together essential component versions such as Azure Synapse optimizations, packages, and connectors with a specific Apache Spark version. Each runtime will be upgraded periodically to include new improvements, features, and patches.

Release cadence

The Apache Spark project usually releases minor versions about **every 6 months**. Once released, the Azure Synapse team aims to provide a **preview runtime within approximately 90 days**, if possible.

Runtime lifecycle

The following chart captures a typical lifecycle path for a Synapse runtime for Apache Spark.



RUNTIME RELEASE STAGE	TYPICAL LIFECYCLE*	NOTES
Preview	3 months*	Microsoft Azure Preview terms apply. See here for details: Preview Terms Of Use | Microsoft Azure
Generally Available (GA)	12 months*	Generally available (GA) runtimes are open to all eligible customers and are ready for production use. A GA runtime may not be elected to move into an LTS stage at Microsoft discretion.
Long Term Support (LTS)	12 months*	Long term support (LTS) runtimes are open to all eligible customers and are ready for production use, but customers are encouraged to expedite validation and workload migration to latest GA runtimes.
End of Life announced (EOLA)	12 months* for GA or LTS runtimes. 1 month* for Preview runtimes.	Prior to the end of a given runtime's lifecycle, we aim to provide 12 months notice to customers as an exit ramp to migrate workloads to a GA runtime.
End of Life (EOL)	-	At this stage, the runtime is retired and no longer supported.

** Expected duration of a runtime in each stage. These timelines are provided as an example for a given runtime, and may vary depending on various factors. Lifecycle timelines are subject to change at Microsoft discretion.*

*** Your use of runtimes is governed by the terms applicable to your Azure subscription.*

> **IMPORTANT**
>
> - The above timelines are provided as examples based on current Apache Spark releases. If the Apache Spark project changes the lifecycle of a specific version affecting a Synapse runtime, changes to the stage dates will be noted on the release notes.
> - Both GA and LTS runtimes may be moved into EOL stage faster based on outstanding security risks and usage rates criteria at Microsoft discretion.
> - Please refer to Lifecycle FAQ - Microsoft Azure for information about Azure lifecycle policies.

Release stages and support

Preview runtimes

Azure Synapse Analytics provides previews to give you a chance to evaluate and share feedback on features before they become generally available (GA).

At the end of the Preview lifecycle for the runtime, Microsoft will assess if the runtime will move into a Generally Availability (GA) based on customer usage, security and stability criteria.

If not eligible for GA stage, the Preview runtime will move into the retirement cycle.

Generally available runtimes

Once a runtime is Generally Available, only security fixes will be backported. In addition, new components or features will be introduced if they do not change underlying dependencies or component versions.

At the end of the GA lifecycle for the runtime, Microsoft will assess if the runtime will have an extended lifecycle (LTS) based on customer usage, security and stability criteria.

If not eligible for LTS stage, the GA runtime will move into the retirement cycle.

Long term support runtimes

For runtimes that are covered by Long term support (LTS) customers are encouraged to expedite validation and migration of code base and workloads to the latest GA runtimes. We recommend that customers do not onboard new workloads using an LTS runtime. Security fixes and stability improvements may be backported, but no new components or features will be introduced into the runtime at this stage.

End of life announcement

Prior to the end of the runtime lifecycle at any stage, an end of life announcement (EOLA) is performed.

Support SLAs are applicable for EOL announced runtimes, but all customers must migrate to a GA stage runtime no later than the EOL date.

Existing pools will work as expected, but **no new Synapse Spark pools of such a version can be created, as the runtime version will not be listed on Azure Synapse Studio, Synapse API, or Azure Portal.**

If necessary due to outstanding security issues, runtime usage, or other factors, Microsoft may expedite moving a runtime into the final EOL stage at any time, at Microsoft's discretion.

End of life and retirement

As of the applicable EOL date, runtimes are considered retired.

- Existing Spark Pools definitions and metadata will still exist in the workspace for a defined period, yet **all**

pipelines, jobs and notebooks will not be able to execute.

- Spark Pools definitions will be deleted from the Synapse workspace in 90 days after the applicable EOL date.

Machine learning with Apache Spark

8/19/2022 • 3 minutes to read • Edit Online

Apache Spark in Azure Synapse Analytics enables machine learning with big data, providing the ability to obtain valuable insight from large amounts of structured, unstructured, and fast-moving data.

This section includes an overview and tutorials for machine learning workflows, including exploratory data analysis, feature engineering, model training, model scoring, and deployment.

Synapse Runtime

The Synapse Runtime is a curated environment optimized for data science and machine learning. The Synapse Runtime offers a range of popular open-source libraries and builds in the Azure Machine Learning SDK by default. The Synapse Runtime also includes many external libraries, including PyTorch, Scikit-Learn, XGBoost, and more.

Learn more about the available libraries and related versions by viewing the published Azure Synapse Analytics runtime.

Exploratory data analysis

When using Apache Spark in Azure Synapse Analytics, there are various built-in options to help you visualize your data, including Synapse notebook chart options, access to popular open-source libraries like Seaborn and Matplotlib, as well as integration with Synapse SQL and Power BI.

Learn more about the data visualization and data analysis options by viewing the article on how to visualize data using Azure Synapse Notebooks.

Feature engineering

By default, the Synapse Runtime includes a set of libraries that are commonly used for feature engineering. For large datasets, you can use Spark SQL, MLlib, and Koalas for feature engineering. For smaller datasets, third-party libraries like Numpy, Pandas, and Scikit-learn also provide useful methods for these scenarios.

Train models

There are several options when training machine learning models using Azure Spark in Azure Synapse Analytics: Apache Spark MLlib, Azure Machine Learning, and various other open-source libraries.

Learn more about the machine learning capabilities by viewing the article on how to train models in Azure Synapse Analytics.

SparkML and MLlib

Spark's in-memory distributed computation capabilities make it a good choice for the iterative algorithms used in machine learning and graph computations. `spark.ml` provides a uniform set of high-level APIs that help users create and tune machine learning pipelines.To learn more about `spark.ml`, you can visit the Apache Spark ML programming guide.

Azure Machine Learning automated ML

Azure Machine Learning automated ML (automated machine learning) helps automate the process of developing machine learning models. It allows data scientists, analysts, and developers to build ML models with

high scale, efficiency, and productivity all while sustaining model quality. The components to run the Azure Machine Learning automated ML SDK is built directly into the Synapse Runtime.

Open-source libraries

Every Apache Spark pool in Azure Synapse Analytics comes with a set of pre-loaded and popular machine learning libraries. Some of the relevant machine learning libraries that are included by default include:

- Scikit-learn is one of the most popular single-node machine learning libraries for classical ML algorithms. Scikit-learn supports most of the supervised and unsupervised learning algorithms and can also be used for data-mining and data-analysis.
- XGBoost is a popular machine learning library that contains optimized algorithms for training decision trees and random forests.
- PyTorch & TensorFlow are powerful Python deep learning libraries. Within an Apache Spark pool in Azure Synapse Analytics, you can use these libraries to build single-machine models by setting the number of executors on your pool to zero. Even though Apache Spark is not functional under this configuration, it is a simple and cost-effective way to create single-machine models.

Track model development

MLFlow is an open-source library for managing the life cycle of your machine learning experiments. MLFlow Tracking is a component of MLflow that logs and tracks your training run metrics and model artifacts. To learn more about how you can use MLFlow Tracking through Azure Synapse Analytics and Azure Machine Learning, visit this tutorial on how to use MLFlow.

Model scoring

Model scoring, or inferencing, is the phase where a model is used to make predictions. For model scoring with SparkML or MLLib, you can leverage the native Spark methods to perform inferencing directly on a Spark DataFrame. For other open-source libraries and model types, you can also create a Spark UDF to scale out inference on large datasets. For smaller datasets, you can also use the native model inference methods provided by the library.

Register and serve models

Registering a model allows you to store, version, and track metadata about models in your workspace. After you have finished training your model, you can register your model to the Azure Machine Learning model registry. Once registered, ONNX models can also be used to enrich the data stored in dedicated SQL pools.

Next steps

To get started with machine learning in Azure Synapse Analytics, be sure to check out the following tutorials:

- Analyze data with Azure Synapse Notebooks
- Train a machine learning model with automated ML
- Train a machine learning model with Apache Spark MLlib

Azure Synapse Analytics notebooks

8/19/2022 • 2 minutes to read • Edit Online

A Synapse Studio notebook is a web interface for you to create files that contain live code, visualizations, and narrative text. Notebooks are a good place to validate ideas and use quick experiments to get insights from your data.

With an Synapse Studio notebook, you can:

- Get started with zero setup effort.
- Keep data secure with built-in enterprise security features.
- Analyze data across raw formats (CSV, txt, JSON, etc.), processed file formats (parquet, Delta Lake, ORC, etc.), and SQL tabular data files against Spark and SQL.
- Be productive with enhanced authoring capabilities and built-in data visualization.

This section contains articles on mixing languages, creating data visualizations, parameterizing notebooks, building pipelines, and more. It also contains references and tutorials on how you can get started with your notebook development.

Create, manage, and use notebooks

You can manage notebooks using the Synapse Studio UI.

To learn more on how you can create and manage notebooks, see the following articles:

- Manage Notebooks
  - Create notebooks
  - Develop notebooks
  - Bring data to a notebook
  - Use multiple languages using magic commands and temporary tables
  - Use cell magic commands
- Development
  - Configure Spark session settings
  - Use Microsoft Spark utilities
  - Visualize data using notebooks and libraries
  - Integrate a notebook into pipelines

Next steps

Notebooks are also widely used in data preparation, data visualization, machine learning, and other big data scenarios. To learn more about how you can use notebooks for your data analysis and big data scenarios, please visit the following tutorials:

- Create a notebook
- Create visualizations using Synapse Studio notebooks
- Build machine learning models with Apache Spark MLlib
- Build machine learning models with Azure automated ML

Manage libraries for Apache Spark in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

Libraries provide reusable code that you might want to include in your programs or projects.

You might need to update your serverless Apache Spark pool environment for various reasons. For example, you might find that:

- One of your core dependencies released a new version.
- You need an extra package for training your machine learning model or preparing your data.
- You have found a better package and no longer need the older package.
- Your team has built a custom package that you need available in your Apache Spark pool.

To make third party or locally built code available to your applications, install a library onto one of your serverless Apache Spark pools or notebook session.

> **IMPORTANT**
>
> - There are three levels of package installing on Synapse Analytics -- default level, Spark pool level and session level.
> - Apache Spark in Azure Synapse Analytics has a full Anaconda install plus extra libraries served as the default level installation which is fully managed by Synapse. The Spark pool level packages can be used by all running Artifacts, e.g., Notebook and Spark job definition attaching the corresponding Spark pool. The session level installation will create an environment for the specific Notebook session, the change of session level libraries will not be persisted between sessions.
> - You can upload custom libraries and a specific version of an open-source library that you would like to use in your Azure Synapse Analytics Workspace. The workspace packages can be installed in your Spark pools.
> - To be noted, the pool level library management can take certain amount of time depending on the size of packages and the complexity of required dependencies. The session level installation is suggested with experimental and quick iterative scenarios.

Default Installation

Default packages include a full Anaconda install plus extra commonly used libraries. The full libraries list can be found at Apache Spark version support.

When a Spark instance starts, these libraries are included automatically. More packages can be added at the Spark pool level or session level.

Workspace packages

When your team develops custom applications or models, you might develop various code artifacts like *.whl* or *.jar* files to package your code.

In Synapse, workspace packages can be custom or private *.whl* or *.jar* files. You can upload these packages to your workspace and later assign them to a specific serverless Apache Spark pool. Once assigned, these workspace packages are installed automatically on all Spark pool sessions.

To learn more about how to manage workspace libraries, see the following article:

- Manage workspace packages

Pool packages

In some cases, you might want to standardize the packages that are used on an Apache Spark pool. This standardization can be useful if the same packages are commonly installed by multiple people on your team.

Using the Azure Synapse Analytics pool management capabilities, you can configure the default set of libraries to install on a given serverless Apache Spark pool. These libraries are installed on top of the base runtime.

Currently, pool management is only supported for Python. For Python, Synapse Spark pools use Conda to install and manage Python package dependencies. When specifying your pool-level libraries, you can now provide a *requirements.txt* or an *environment.yml* file. This environment configuration file is used every time a Spark instance is created from that Spark pool.

To learn more about these capabilities, see Manage Spark pool packages.

> **IMPORTANT**
>
> - If the package you are installing is large or takes a long time to install, this fact affects the Spark instance start up time.
> - Altering the PySpark, Python, Scala/Java, .NET, or Spark version is not supported.
> - Installing packages from PyPI is not supported within DEP-enabled workspaces.

Session-scoped packages

Often, when doing interactive data analysis or machine learning, you might try newer packages or you might need packages that are currently unavailable on your Apache Spark pool. Instead of updating the pool configuration, users can now use session-scoped packages to add, manage, and update session dependencies.

Session-scoped packages allow users to define package dependencies at the start of their session. When you install a session-scoped package, only the current session has access to the specified packages. As a result, these session-scoped packages don't affect other sessions or jobs using the same Apache Spark pool. In addition, these libraries are installed on top of the base runtime and pool level packages.

To learn more about how to manage session-scoped packages, see the following articles:

- Python session packages: At the start of a session, provide a Conda *environment.yml* to install more Python packages from popular repositories.
- Scala/Java session packages: At the start of your session, provide a list of *.jar* files to install using `%%configure`.

Manage your packages outside Synapse Analytics UI

If your team want to manage the libraries without visiting the package management UIs, you have the options to manage the workspace packages and pool level package updates through Azure PowerShell cmdlets or REST APIs for Synapse Analytics.

To learn more about Azure PowerShell cmdlets and package management REST APIs, see the following articles:

- Azure PowerShell cmdlets for Synapse Analytics: Manage your Spark pool libraries through Azure PowerShell cmdlets
- Package management REST APIs: Manage your Spark pool libraries through REST APIs

Next steps

- View the default libraries: Apache Spark version support
- Troubleshoot library installation errors: Troubleshoot library errors

Monitor your Apache Spark applications in Synapse Studio

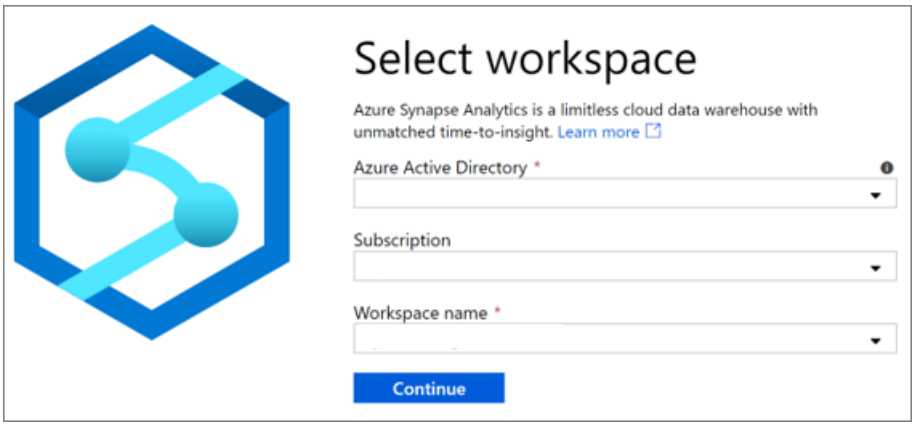
8/19/2022 • 2 minutes to read • Edit Online

With Azure Synapse Analytics, you can use Apache Spark to run notebooks, jobs, and other kinds of applications on your Apache Spark pools in your workspace.

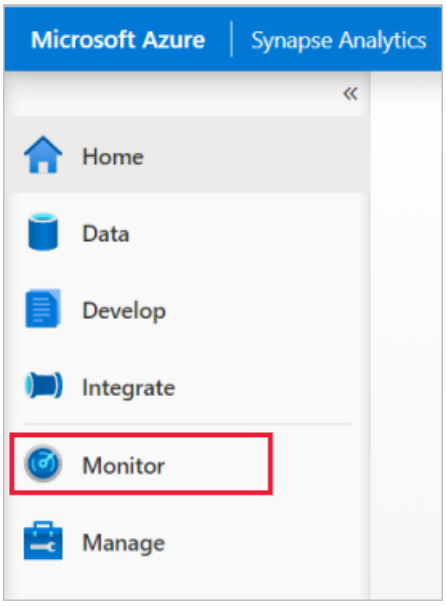
This article explains how to monitor your Apache Spark applications, allowing you to keep an eye on the latest status, issues, and progress.

Access Apache Spark applications list

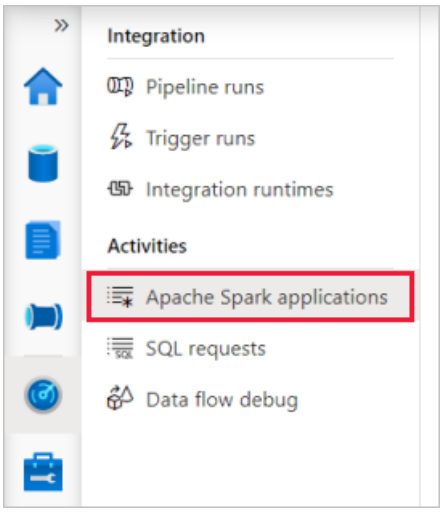
To see the list of Apache Spark applications in your workspace, first open the Synapse Studio and select your workspace.



Once you've opened your workspace, select the **Monitor** section on the left.



Select **Apache Spark applications** to view the list of Apache Spark applications.



Filter your Apache Spark applications

You can filter the list of Apache Spark applications to the ones you're interested in. The filters at the top of the screen allow you to specify a field on which you'd like to filter.

For example, you can filter the view to see only the Apache Spark applications that contain the name "sales":

View details about a specific Apache Spark application

To view the details about one of your Apache Spark applications, select the Apache Spark application and view the details. If the Apache Spark application is still running, you can monitor the progress. Read more.

Next steps

For more information on monitoring pipeline runs, see the Monitor pipeline runs Synapse Studio article.

For more information on debugging Apache Spark application, see the Monitor Apache Spark applications on Synapse Studio article.

Linux Foundation Delta Lake overview

8/19/2022 • 10 minutes to read • Edit Online

This article has been adapted for more clarity from its original counterpart here. This article helps you quickly explore the main features of Delta Lake. The article provides code snippets that show how to read from and write to Delta Lake tables from interactive, batch, and streaming queries. The code snippets are also available in a set of notebooks PySpark here, Scala here, and C# here

Here's what we will cover:

- Create a table
- Read data
- Update table data
- Overwrite table data
- Conditional update without overwrite
- Read older versions of data using Time Travel
- Write a stream of data to a table
- Read a stream of changes from a table
- SQL Support

Configuration

Make sure you modify the below as appropriate for your environment.

```
import random

session_id = random.randint(0,1000000)
delta_table_path = "/delta/delta-table-{0}".format(session_id)

delta_table_path
```

```
var sessionId = (new Random()).Next(10000000);
var deltaTablePath = $"/delta/delta-table-{sessionId}";

deltaTablePath
```

```
val sessionId = scala.util.Random.nextInt(1000000)
val deltaTablePath = s"/delta/delta-table-$sessionId";
```

Results in:

'/delta/delta-table-335323'

Create a table

To create a Delta Lake table, write a DataFrame out a DataFrame in the delta format. You can change the format from Parquet, CSV, JSON, and so on, to delta.

The code that follows shows you how to create a new Delta Lake table using the schema inferred from your DataFrame.

```
data = spark.range(0,5)
data.show()
data.write.format("delta").save(delta_table_path)
```

```
var data = spark.Range(0,5);
data.Show();
data.Write().Format("delta").Save(deltaTablePath);
```

```
val data = spark.range(0, 5)
data.show
data.write.format("delta").save(deltaTablePath)
```

Results in:

ID
0
1
2
3
4

Read data

You read data in your Delta Lake table by specifying the path to the files and the delta format.

```
df = spark.read.format("delta").load(delta_table_path)
df.show()
```

```
var df = spark.Read().Format("delta").Load(deltaTablePath);
df.Show()
```

```
val df = spark.read.format("delta").load(deltaTablePath)
df.show()
```

Results in:

ID
1
3
4

ID
0
2

The order of the results is different from above as there was no order explicitly specified before outputting the results.

Update table data

Delta Lake supports several operations to modify tables using standard DataFrame APIs, this is one of the big enhancements that delta format adds. The following example runs a batch job to overwrite the data in the table.

```
data = spark.range(5,10)
data.write.format("delta").mode("overwrite").save(delta_table_path)
df.show()
```

```
var data = spark.Range(5,10);
data.Write().Format("delta").Mode("overwrite").Save(deltaTablePath);
df.Show();
```

```
val data = spark.range(5, 10)
data.write.format("delta").mode("overwrite").save(deltaTablePath)
df.show()
```

Results in:

ID
7
8
5
9
6

Here you can see that all five records have been updated to hold new values.

Save as catalog tables

Delta Lake can write to managed or external catalog tables.

```
data.write.format("delta").saveAsTable("ManagedDeltaTable")
spark.sql("CREATE TABLE ExternalDeltaTable USING DELTA LOCATION '{0}'".format(delta_table_path))
spark.sql("SHOW TABLES").show()
```

```
data.Write().Format("delta").SaveAsTable("ManagedDeltaTable");
spark.Sql($"CREATE TABLE ExternalDeltaTable USING DELTA LOCATION '{deltaTablePath}'");
spark.Sql("SHOW TABLES").Show();
```

```
data.write.format("delta").saveAsTable("ManagedDeltaTable")
spark.sql(s"CREATE TABLE ExternalDeltaTable USING DELTA LOCATION '$deltaTablePath'")
spark.sql("SHOW TABLES").show
```

Results in:

DATABASE	TABLENAME	ISTEMPORARY
default	externaldeltatable	false
default	manageddeltatable	false

With this code, you created a new table in the catalog from an existing dataframe, referred to as a managed table. Then you defined a new external table in the catalog that uses an existing location, referred to as an external table. In the output you can see both tables, no matter how they were created, are listed in the catalog.

Now you can look at the extended properties of both of these tables

```
spark.sql("DESCRIBE EXTENDED ManagedDeltaTable").show(truncate=False)
```

```
spark.Sql("DESCRIBE EXTENDED ManagedDeltaTable").Show(truncate: 0);
```

```
spark.sql("DESCRIBE EXTENDED ManagedDeltaTable").show(truncate=false)
```

Results in:

COL_NAME	DATA_TYPE	COMMENT
id	bigint	null
Detailed Table Information		
Database	default	
Table	manageddeltatable	
Owner	trusted-service-user	
Created Time	Sat Apr 25 00:35:34 UTC 2020	
Last Access	Thu Jan 01 00:00:00 UTC 1970	
Created By	Spark 2.4.4.2.6.99.201-11401300	
Type	MANAGED	

COL_NAME	DATA_TYPE	COMMENT
Provider	delta	
Table Properties	[transient_lastDdlTime=1587774934]	
Statistics	2407 bytes	
Location	abfss://data@<data lake>.dfs.core.windows.net/synapse/workspaces/<workspace name>/warehouse/manageddeltatable	
Serde Library	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe	
InputFormat	org.apache.hadoop.mapred.SequenceFileInputFormat	
OutputFormat	org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat	
Storage Properties	[serialization.format=1]	

```
spark.sql("DESCRIBE EXTENDED ExternalDeltaTable").show(truncate=False)
```

```
spark.Sql("DESCRIBE EXTENDED ExternalDeltaTable").Show(truncate: 0);
```

```
spark.sql("DESCRIBE EXTENDED ExternalDeltaTable").show(truncate=false)
```

Results in:

COL_NAME	DATA_TYPE	COMMENT
id	bigint	null
Detailed Table Information		
Database	default	
Table	externaldeltatable	
Owner	trusted-service-user	
Created Time	Sat Apr 25 00:35:38 UTC 2020	
Last Access	Thu Jan 01 00:00:00 UTC 1970	
Created By	Spark 2.4.4.2.6.99.201-11401300	

COL_NAME	DATA_TYPE	COMMENT
Type	EXTERNAL	
Provider	DELTA	
Table Properties	[transient_lastDdlTime=1587774938]	
Location	abfss://data@<data lake>.dfs.core.windows.net/delta/delta-table-587152	
Serde Library	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe	
InputFormat	org.apache.hadoop.mapred.SequenceFileInputFormat	
OutputFormat	org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat	
Storage Properties	[serialization.format=1]	

Conditional update without overwrite

Delta Lake provides programmatic APIs to conditional update, delete, and merge (this is commonly referred to as an upsert) data into tables.

```
from delta.tables import *
from pyspark.sql.functions import *

delta_table = DeltaTable.forPath(spark, delta_table_path)

delta_table.update(
  condition = expr("id % 2 == 0"),
  set = { "id": expr("id + 100") })
delta_table.toDF().show()
```

```
using Microsoft.Spark.Extensions.Delta;
using Microsoft.Spark.Extensions.Delta.Tables;
using Microsoft.Spark.Sql;
using static Microsoft.Spark.Sql.Functions;

var deltaTable = DeltaTable.ForPath(deltaTablePath);

deltaTable.Update(
  condition: Expr("id % 2 == 0"),
  set: new Dictionary<string, Column>(){{ "id", Expr("id + 100") }});
deltaTable.ToDF().Show();
```

```
import io.delta.tables._
import org.apache.spark.sql.functions._

val deltaTable = DeltaTable.forPath(deltaTablePath)

// Update every even value by adding 100 to it
deltaTable.update(
  condition = expr("id % 2 == 0"),
  set = Map("id" -> expr("id + 100")))
deltaTable.toDF.show
```

Results in:

ID
106
108
5
7
9

Here you just added 100 to every even ID.

```
delta_table.delete("id % 2 == 0")
delta_table.toDF().show()
```

```
deltaTable.Delete(condition: Expr("id % 2 == 0"));
deltaTable.ToDF().Show();
```

```
deltaTable.delete(condition = expr("id % 2 == 0"))
deltaTable.toDF.show
```

Results in:

ID
5
7
9

Notice that every even row has been deleted.

```
new_data = spark.range(0,20).alias("newData")

delta_table.alias("oldData")\
    .merge(new_data.alias("newData"), "oldData.id = newData.id")\
    .whenMatchedUpdate(set = { "id": lit("-1")})\
    .whenNotMatchedInsert(values = { "id": col("newData.id") })\
    .execute()

delta_table.toDF().show(100)
```

```
var newData = spark.Range(20).As("newData");

deltaTable
    .As("oldData")
    .Merge(newData, "oldData.id = newData.id")
    .WhenMatched()
        .Update(new Dictionary<string, Column>() {{"id", Lit("-1")}})
    .WhenNotMatched()
        .Insert(new Dictionary<string, Column>() {{"id", Col("newData.id")}})
    .Execute();

deltaTable.ToDF().Show(100);
```

```
val newData = spark.range(0, 20).toDF

deltaTable.as("oldData").
  merge(
    newData.as("newData"),
    "oldData.id = newData.id").
  whenMatched.
  update(Map("id" -> lit(-1))).
  whenNotMatched.
  insert(Map("id" -> col("newData.id"))).
  execute()

deltaTable.toDF.show()
```

Results in:

ID
18
15
19
2
1
6
8
3

ID
-1
10
13
0
16
4
-1
12
11
14
-1
17

Here you have a combination of the existing data. The existing data has been assigned the value -1 in the update(WhenMatched) code path. The new data that was created at the top of the snippet and was added via the insert code path (WhenNotMatched), was also added.

History

Delta Lake's has the ability to allow looking into history of a table. That is, the changes that were made to the underlying Delta Table. The cell below shows how simple it is to inspect the history.

```
delta_table.history().show(20, 1000, False)
```

```
deltaTable.History().Show(20, 1000, false);
```

```
deltaTable.history.show(false)
```

Results in:

VERSION	TIMESTAMP	USERID	USERNAME	OPERATION	OPERATIONPARAMETERS	JOB	NOTEBOOK	CLUSTERID	READVERSION	ISOLATIONLEVEL	ISBLINDAPPEND

VERSION	TIMESTAMP	USERID	USERNAME	OPERATION	OPERATIONPARAMETERS	JOB	NOTEBOOK	CLUSTERID	READVERSION	ISOLATIONLEVEL	ISBLINDAPPEND
4	2020-04-25 00:36:27	null	null	MERGE	[predicate -> (oldData.`ID` = newData.`ID`)]	null	null	null	3	null	false
3	2020-04-25 00:36:08	null	null	DELETE	[predicate -> ["((`ID` % CAST(2 AS BIGINT)) = CAST(0 AS BIGINT))"]]	null	null	null	2	null	false
2	2020-04-25 00:35:51	null	null	UPDATE	[predicate -> ((ID#744L % cast(2 as bigint)) = cast(0 as bigint))]	null	null	null	1	null	false
1	2020-04-25 00:35:05	null	null	WRITE	[mode -> Overwrite, partitionBy -> []]	null	null	null	0	null	false
0	2020-04-25 00:34:34	null	null	WRITE	[mode -> ErrorIfExists, partitionBy -> []]	null	null	null	null	null	true

Here you can see all of the modifications made over the above code snippets.

Read older versions of data using Time Travel

It's possible to query previous snapshots of your Delta Lake table by using a feature called Time Travel. If you want to access the data that you overwrote, you can query a snapshot of the table before you overwrote the first set of data using the versionAsOf option.

Once you run the cell below, you should see the first set of data from before you overwrote it. Time Travel is an extremely powerful feature that takes advantage of the power of the Delta Lake transaction log to access data that is no longer in the table. Removing the version 0 option (or specifying version 1) would let you see the newer data again. For more information, see Query an older snapshot of a table.

```
df = spark.read.format("delta").option("versionAsOf", 0).load(delta_table_path)
df.show()
```

```
var df = spark.Read().Format("delta").Option("versionAsOf", 0).Load(deltaTablePath);
df.Show();
```

```
val df = spark.read.format("delta").option("versionAsOf", 0).load(deltaTablePath)
df.show()
```

Results in:

ID
0
1
4
3
2

Here you can see you have gone back to the earliest version of the data.

Write a stream of data to a table

You can also write to a Delta Lake table using Spark's Structured Streaming. The Delta Lake transaction log guarantees exactly once processing, even when there are other streams or batch queries running concurrently against the table. By default, streams run in append mode, which adds new records to the table.

For more information about Delta Lake integration with Structured Streaming, see Table Streaming Reads and Writes.

In the cells below, here's what we are doing:

- Cell 30 Show the newly appended data
- Cell 31 Inspect history
- Cell 32 Stop the structured streaming job
- Cell 33 Inspect history <--You'll notice appends have stopped

First you are going to set up a simple Spark Streaming job to generate a sequence and make the job write to

your Delta Table.

```
streaming_df = spark.readStream.format("rate").load()
stream = streaming_df\
    .selectExpr("value as id")\
    .writeStream\
    .format("delta")\
    .option("checkpointLocation", "/tmp/checkpoint-{0}".format(session_id))\
    .start(delta_table_path)
```

```
var streamingDf = spark.ReadStream().Format("rate").Load();
var stream = streamingDf.SelectExpr("value as
id").WriteStream().Format("delta").Option("checkpointLocation", $"/tmp/checkpoint-
{sessionId}").Start(deltaTablePath);
```

```
val streamingDf = spark.readStream.format("rate").load()
val stream = streamingDf.select($"value" as "id").writeStream.format("delta").option("checkpointLocation",
s"/tmp/checkpoint-$sessionId").start(deltaTablePath)
```

Read a stream of changes from a table

While the stream is writing to the Delta Lake table, you can also read from that table as a streaming source. For example, you can start another streaming query that prints all the changes made to the Delta Lake table.

```
delta_table.toDF().sort(col("id").desc()).show(100)
```

```
deltaTable.ToDF().Sort(Col("id").Desc()).Show(100);
```

```
deltaTable.toDF.sort($"id".desc).show
```

Results in:

ID
19
18
17
16
15
14
13
12

ID
11
10
8
6
4
3
2
1
0
-1
-1
-1

```
delta_table.history().drop("userId", "userName", "job", "notebook", "clusterId", "isolationLevel",
"isBlindAppend").show(20, 1000, False)
```

```
deltaTable.History().Drop("userId", "userName", "job", "notebook", "clusterId", "isolationLevel",
"isBlindAppend").Show(20, 1000, false);
```

```
deltaTable.history.show
```

Results in:

VERSION	TIMESTAMP	OPERATION	OPERATIONPARAMETERS	READVERSION
5	2020-04-25 00:37:09	STREAMING UPDATE	[outputMode -> Append, queryId -> d26b4f8a-7e5a-44f2-a5fb-23a7bd02aef7, epochId -> 0]	4
4	2020-04-25 00:36:27	MERGE	[predicate -> (oldData.`id` = newData.`id`)]	3

VERSION	TIMESTAMP	OPERATION	OPERATIONPARAMETERS	READVERSION
3	2020-04-25 00:36:08	DELETE	[predicate -> ["((`id` % CAST(2 AS BIGINT)) = CAST(0 AS BIGINT))"]]	2
2	2020-04-25 00:35:51	UPDATE	[predicate -> ((id#744L % cast(2 as bigint)) = cast(0 as bigint))]	1
1	2020-04-25 00:35:05	WRITE	[mode -> Overwrite, partitionBy -> []]	0
0	2020-04-25 00:34:34	WRITE	[mode -> ErrorIfExists, partitionBy -> []]	null

Here you are dropping some of the less interesting columns to simplify the viewing experience of the history view.

```
stream.stop()
delta_table.history().drop("userId", "userName", "job", "notebook", "clusterId", "isolationLevel",
"isBlindAppend").show(100, 1000, False)
```

```
stream.Stop();
deltaTable.History().Drop("userId", "userName", "job", "notebook", "clusterId", "isolationLevel",
"isBlindAppend").Show(100, 1000, false);
```

```
stream.stop
deltaTable.history.show
```

Results in:

VERSION	TIMESTAMP	OPERATION	OPERATIONPARAMETERS	READVERSION
5	2020-04-25 00:37:09	STREAMING UPDATE	[outputMode -> Append, queryId -> d26b4f8a-7e5a-44f2-a5fb-23a7bd02aef7, epochId -> 0]	4
4	2020-04-25 00:36:27	MERGE	[predicate -> (oldData.`id` = newData.`id`)]	3
3	2020-04-25 00:36:08	DELETE	[predicate -> ["((`id` % CAST(2 AS BIGINT)) = CAST(0 AS BIGINT))"]]	2

VERSION	TIMESTAMP	OPERATION	OPERATIONPARAMETERS	READVERSION
2	2020-04-25 00:35:51	UPDATE	[predicate -> ((id#744L % cast(2 as bigint)) = cast(0 as bigint))]	1
1	2020-04-25 00:35:05	WRITE	[mode -> Overwrite, partitionBy -> []]	0
0	2020-04-25 00:34:34	WRITE	[mode -> ErrorIfExists, partitionBy -> []]	null

Convert Parquet to Delta

You can do an in-place conversion from the Parquet format to Delta.

Here you are going to test if the existing table is in delta format or not.

```
parquet_path = "/parquet/parquet-table-{0}".format(session_id)
data = spark.range(0,5)
data.write.parquet(parquet_path)
DeltaTable.isDeltaTable(spark, parquet_path)
```

```
var parquetPath = $"/parquet/parquet-table-{sessionId}";
var data = spark.Range(0,5);
data.Write().Parquet(parquetPath);
DeltaTable.IsDeltaTable(parquetPath)
```

```
val parquetPath = s"/parquet/parquet-table-$sessionId"
val data = spark.range(0,5)
data.write.parquet(parquetPath)
DeltaTable.isDeltaTable(parquetPath)
```

Results in:

False

Now you are going to convert the data to delta format and verify it worked.

```
DeltaTable.convertToDelta(spark, "parquet.`{0}`".format(parquet_path))
DeltaTable.isDeltaTable(spark, parquet_path)
```

```
DeltaTable.ConvertToDelta(spark, $"parquet.`{parquetPath}`");
DeltaTable.IsDeltaTable(parquetPath)
```

```
DeltaTable.convertToDelta(spark, s"parquet.`$parquetPath`")
DeltaTable.isDeltaTable(parquetPath)
```

Results in:

True

SQL support

Delta supports table utility commands through SQL. You can use SQL to:

- Get a DeltaTable's history
- Vacuum a DeltaTable
- Convert a Parquet file to Delta

```
spark.sql("DESCRIBE HISTORY delta.`{0}`".format(delta_table_path)).show()
```

```
spark.Sql($"DESCRIBE HISTORY delta.`{deltaTablePath}`").Show();
```

```
spark.sql(s"DESCRIBE HISTORY delta.`$deltaTablePath`").show()
```

Results in:

VERSION	TIMESTAMP	USERID	USERNAME	OPERATION	OPERATIONPARAMETERS	JOB	NOTEBOOK	CLUSTERID	READVERSION	ISOLATIONLEVEL	ISBLINDAPPEND
5	2020-04-25 00:37:09	null	null	STREAMING UPDATE	[outputMode -> Ap...	null	null	null	4	null	true
4	2020-04-25 00:36:27	null	null	MERGE	[predicate -> (ol...	null	null	null	3	null	false
3	2020-04-25 00:36:08	null	null	DELETE	[predicate -> ["(...	null	null	null	2	null	false
2	2020-04-25 00:35:51	null	null	UPDATE	[predicate -> ((i...	null	null	null	1	null	false
1	2020-04-25 00:35:05	null	null	WRITE	[mode -> Overwrit...	null	null	null	0	null	false
0	2020-04-25 00:34:34	null	null	WRITE	[mode -> ErrorIfE...	null	null	null	null	null	true

```
spark.sql("VACUUM delta.`{0}`".format(delta_table_path)).show()
```

```
spark.Sql($"VACUUM delta.`{deltaTablePath}`").Show();
```

```
spark.sql(s"VACUUM delta.`$deltaTablePath`").show()
```

Results in:

PATH
abfss://data@arca...

Now you are going to verify that a table is not a delta format table, then convert it to delta format using Spark SQL and confirm it was converted correctly.

```
parquet_id = random.randint(0,1000)
parquet_path = "/parquet/parquet-table-{0}-{1}".format(session_id, parquet_id)
data = spark.range(0,5)
data.write.parquet(parquet_path)
DeltaTable.isDeltaTable(spark, parquet_path)
spark.sql("CONVERT TO DELTA parquet.`{0}`".format(parquet_path))
DeltaTable.isDeltaTable(spark, parquet_path)
```

```
var parquetId =  (new Random()).Next(10000000);
var parquetPath = $"/parquet/parquet-table-{sessionId}-{parquetId}";
var data = spark.Range(0,5);
data.Write().Parquet(parquetPath);
DeltaTable.IsDeltaTable(parquetPath);
spark.Sql($"CONVERT TO DELTA parquet.`{parquetPath}`");
DeltaTable.IsDeltaTable(parquetPath);
```

```
val parquetId = scala.util.Random.nextInt(1000)
val parquetPath = s"/parquet/parquet-table-$sessionId-$parquetId"
val data = spark.range(0,5)
data.write.parquet(parquetPath)
DeltaTable.isDeltaTable(parquetPath)
spark.sql(s"CONVERT TO DELTA parquet.`$parquetPath`")
DeltaTable.isDeltaTable(parquetPath)
```

Results in:

True

For full documentation, see the Delta Lake Documentation Page

For more information, see Delta Lake Project.

Next steps

- .NET for Apache Spark documentation
- Azure Synapse Analytics

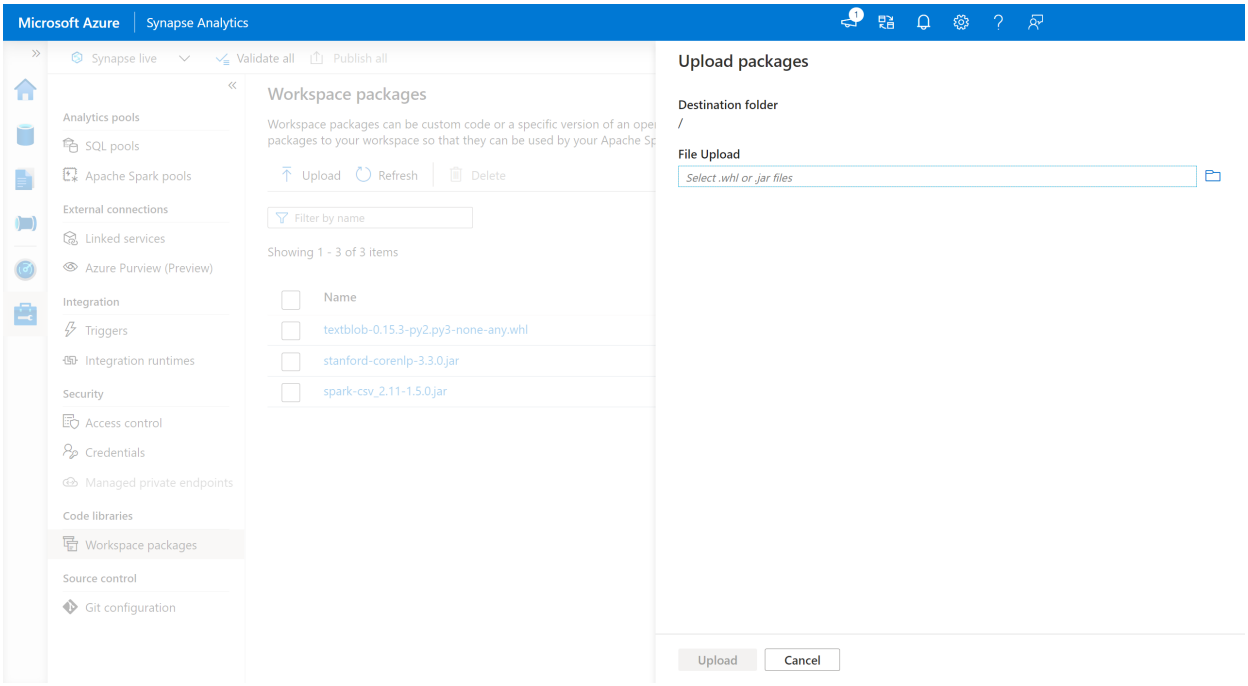
Workspace packages

8/19/2022 • 2 minutes to read • Edit Online

Workspace packages can be custom or private wheel or jar files. You can upload these packages to your workspace and later assign them to a specific Spark pool.

To add workspace packages:

1. Navigate to the **Manage** > **Workspace packages** tab.
2. Upload your wheel files by using the file selector.
3. Once the files have been uploaded to the Azure Synapse workspace, you can add these packages to a given Apache Spark pool.



WARNING

- Within Azure Synapse, an Apache Spark pool can leverage custom libraries that are either uploaded as Workspace Packages or uploaded within a well-known Azure Data Lake Storage path. However, both of these options cannot be used simultaneously within the same Apache Spark pool. If packages are provided using both methods, only the wheel files specified in the Workspace packages list will be installed.
- Once Workspace Packages are used to install packages on a given Apache Spark pool, there is a limitation that you can no longer specify packages using the Storage account path on the same pool.

Storage account

Custom-built wheel packages can be installed on the Apache Spark pool by uploading all the wheel files into the Azure Data Lake Storage (Gen2) account that is linked with the Synapse workspace.

The files should be uploaded to the following path in the storage account's default container:

```
abfss://<file_system>@<account_name>.dfs.core.windows.net/synapse/workspaces/<workspace_name>/sparkpools/<pool_name>/libraries/python/
```

WARNING

- In some cases, you may need to create the file path based on the structure above if it does not already exist. For example, you may need to add the `python` folder within the `libraries` folder if it does not already exist.
- This method of managing custom wheel files will not be supported on the Azure Synapse Runtime for Apache Spark 3.0. Please refer to the Workspace packages feature to manage custom wheel files.

IMPORTANT

To install custom libraries using the Azure DataLake Storage method, you must have the **Storage Blob Data Contributor** or **Storage Blob Data Owner** permissions on the primary Gen2 Storage account that is linked to the Azure Synapse Analytics workspace.

Next steps

- View the default libraries: Apache Spark version support
- Troubleshoot library installation errors: Troubleshoot library errors
- Create a private Conda channel using your Azure Data Lake Storage Account: Conda private channels

Manage libraries for Apache Spark pools in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

Once you have identified the Scala, Java or Python packages that you would like to use or update for your Spark application, you can install or remove them into a Spark pool. Pool-level libraries are available to all notebooks and jobs running on the pool.

There are two primary ways to install a library on a Spark pool:

- Install a workspace library that has been uploaded as a workspace package.
- For updating Python libraries, provide a *requirements.txt* or *Conda environment.yml* environment specification to install packages from repositories like PyPI, Conda-Forge, and more. Read the section about environment specification for further information.

After the changes are saved, a Spark job will run the installation and cache the resulting environment for later reuse. Once the job is complete, new Spark jobs or notebook sessions will use the updated pool libraries.

> **IMPORTANT**
>
> - If the package you are installing is large or takes a long time to install, this affects the Spark instance start up time.
> - Altering the PySpark, Python, Scala/Java, .NET, or Spark version is not supported.
> - Installing packages from external repositories like PyPI, Conda-Forge, or the default Conda channels is not supported within data exfiltration protection enabled workspaces.

Manage packages from Synapse Studio or Azure portal

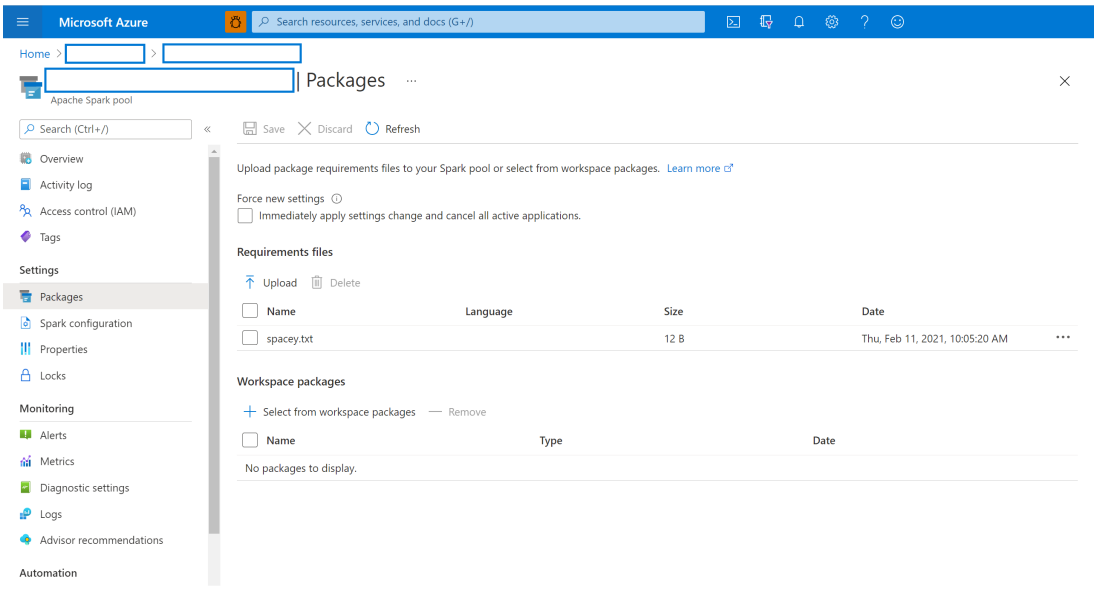
Spark pool libraries can be managed either from the Synapse Studio or Azure portal.

To update or add libraries to a Spark pool:

1. Navigate to your Azure Synapse Analytics workspace from the Azure portal.

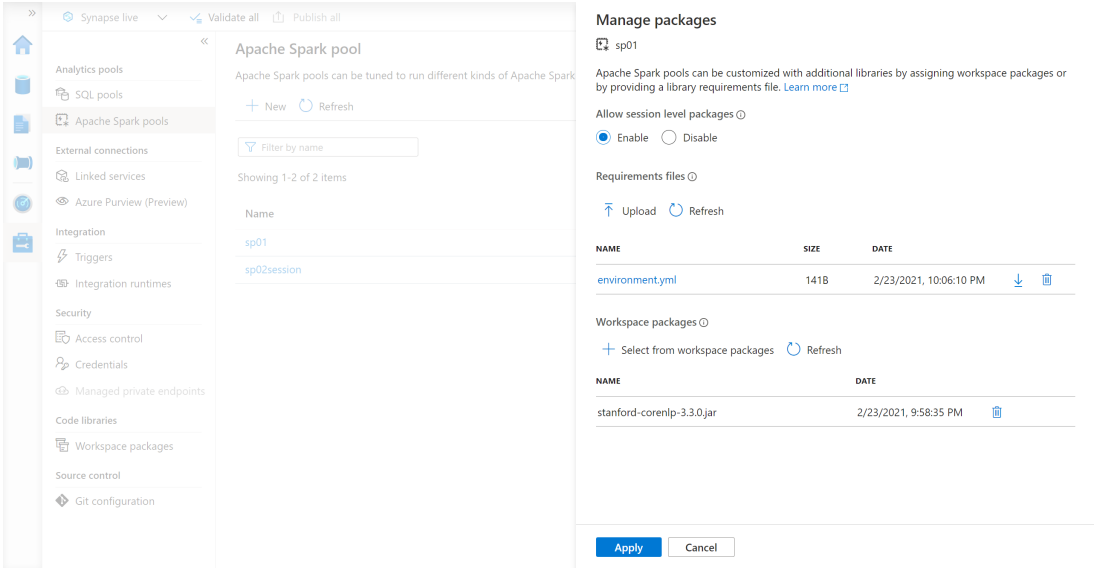
   If you are updating from the **Azure portal**:

   - Under the **Synapse resources** section, select the **Apache Spark pools** tab and select a Spark pool from the list.
   - Select the **Packages** from the **Settings** section of the Spark pool.



If you are updating from the **Synapse Studio**:

- Select **Manage** from the main navigation panel and then select **Apache Spark pools**.
- Select the **Packages** section for a specific Spark pool.



2. For Python feed libraries, upload the environment configuration file using the file selector in the **Packages** section of the page.
3. You can also select additional **workspace packages** to add Jar or Wheel files to your pool.
4. You can also remove the deprecated packages from **Workspace packages** section, your pool will no longer attach these packages.
5. Once you save your changes, a system job will be triggered to install and cache the specified libraries. This process helps reduce overall session startup time.
6. Once the job has successfully completed, all new sessions will pick up the updated pool libraries.

> **IMPORTANT**
>
> By selecting the option to **Force new settings**, you will end the all current sessions for the selected Spark pool. Once the sessions are ended, you will have to wait for the pool to restart.
>
> If this setting is unchecked, then you will have to wait for the current Spark session to end or stop it manually. Once the session has ended, you will need to let the pool restart.

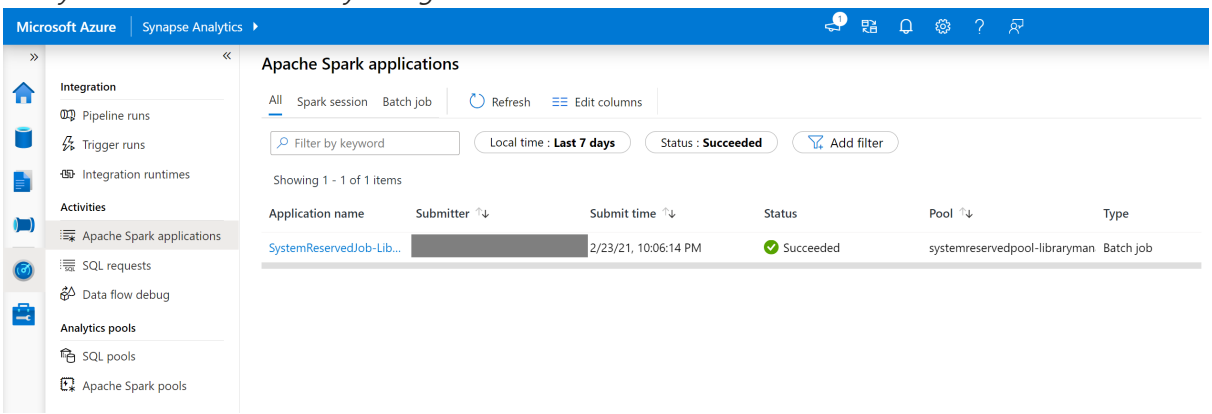
Track installation progress

A system reserved Spark job is initiated each time a pool is updated with a new set of libraries. This Spark job helps monitor the status of the library installation. If the installation fails due to library conflicts or other issues, the Spark pool will revert to its previous or default state.

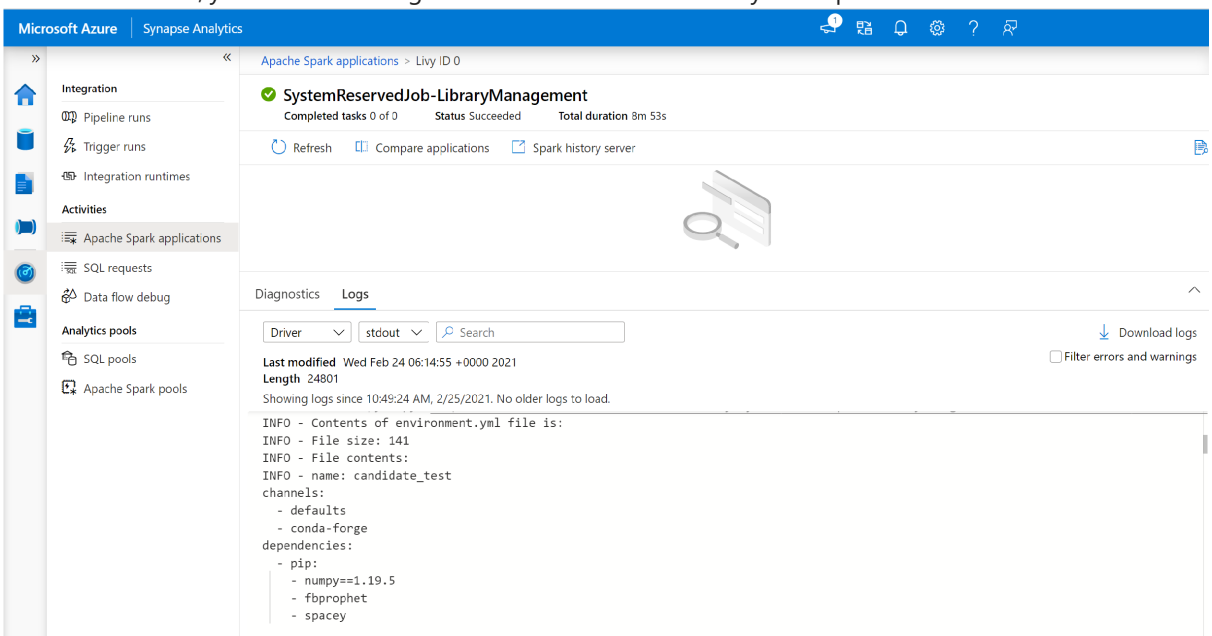
In addition, users can also inspect the installation logs to identify dependency conflicts or see which libraries were installed during the pool update.

To view these logs:

1. Navigate to the Spark applications list in the **Monitor** tab.
2. Select the system Spark application job that corresponds to your pool update. These system jobs run under the *SystemReservedJob-LibraryManagement* title.



3. Switch to view the **driver** and **stdout** logs.
4. Within the results, you will see the logs related to the installation of your dependencies.



Environment specification formats

PIP requirements.txt

A *requirements.txt* file (output from the `pip freeze` command) can be used to upgrade the environment. When a pool is updated, the packages listed in this file are downloaded from PyPI. The full dependencies are then cached and saved for later reuse of the pool.

The following snippet shows the format for the requirements file. The PyPI package name is listed along with an exact version. This file follows the format described in the pip freeze reference documentation.

This example pins a specific version.

```
absl-py==0.7.0
adal==1.2.1
alabaster==0.7.10
```

YML format

In addition, you can also provide an *environment.yml* file to update the pool environment. The packages listed in this file are downloaded from the default Conda channels, Conda-Forge, and PyPI. You can specify other channels or remove the default channels by using the configuration options.

This example specifies the channels and Conda/PyPI dependencies.

```
name: stats2
channels:
- defaults
dependencies:
- bokeh
- numpy
- pip:
  - matplotlib
  - koalas==1.7.0
```

For details on creating an environment from this environment.yml file, see Creating an environment from an environment.yml file.

Next steps

- View the default libraries: Apache Spark version support
- Troubleshoot library installation errors: Troubleshoot library errors

Manage session-scoped packages

8/19/2022 • 2 minutes to read • Edit Online

In addition to pool level packages, you can also specify session-scoped libraries at the beginning of a notebook session. Session-scoped libraries let you specify and use custom Python environments or jar packages within a notebook session.

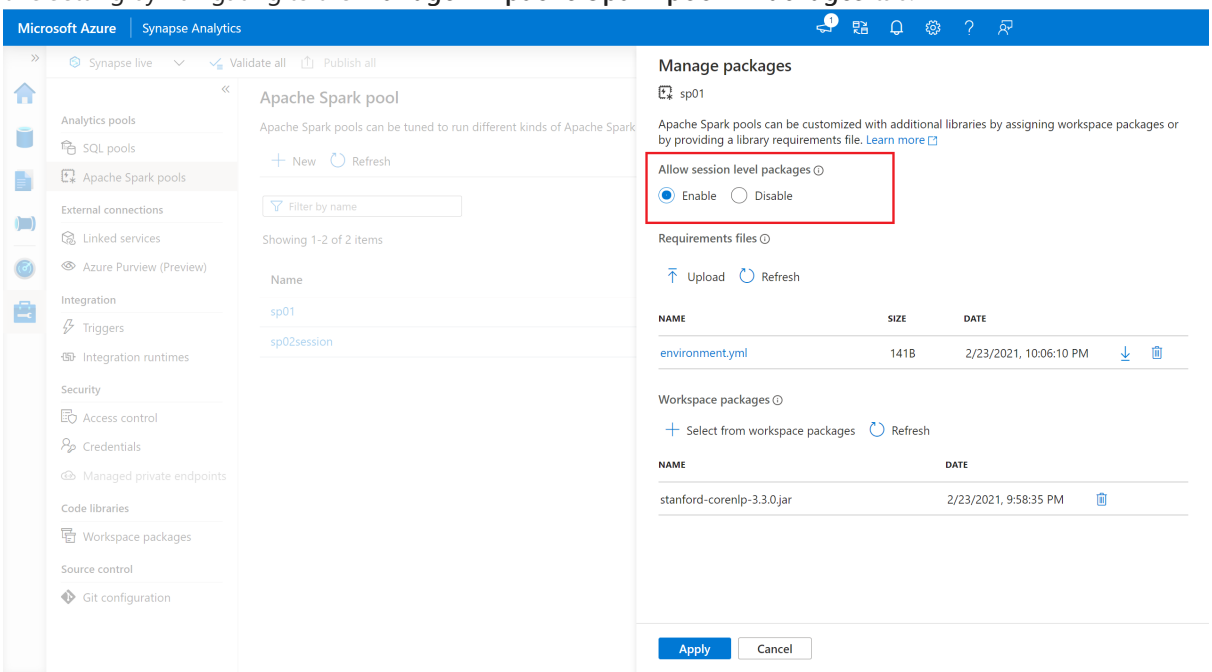
When using session-scoped libraries, it is important to keep the following points in mind:

- When you install session-scoped libraries, only the current notebook has access to the specified libraries.
- These libraries will not impact other sessions or jobs using the same Spark pool.
- These libraries are installed on top of the base runtime and pool level libraries.
- Notebook libraries will take the highest precedence.

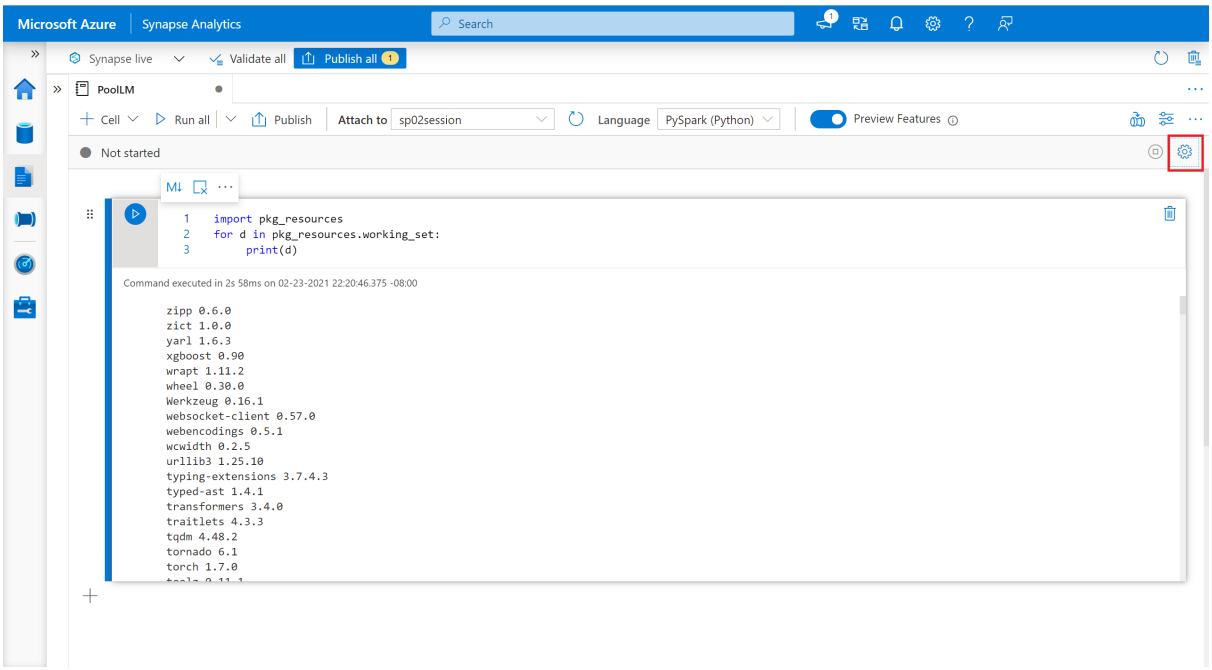
Session-scoped Python packages

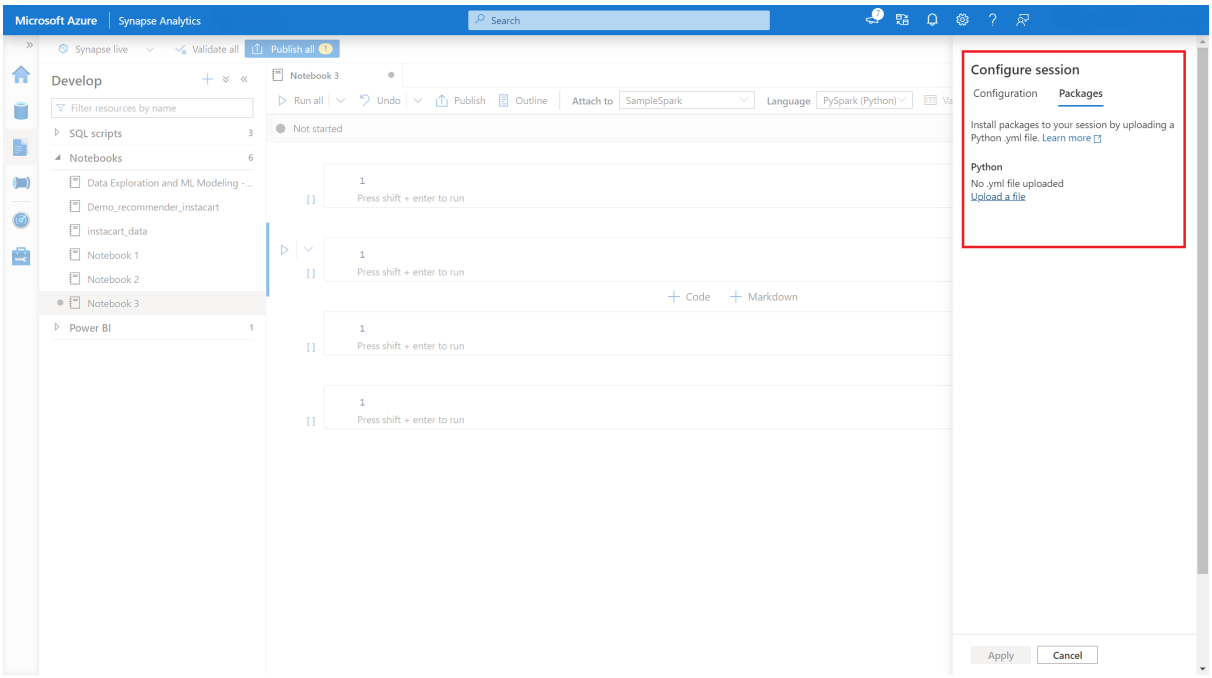
To specify session-scoped Python packages:

1. Navigate to the selected Spark pool and ensure that you have enabled session-level libraries. You can enable this setting by navigating to the **Manage** > **Apache Spark pool** > **Packages** tab.



2. Once the setting has been applied, you can open a notebook and select **Configure Session** > **Packages**.





3. Here, you can upload a Conda *environment.yml* file to install or upgrade packages within a session. Once you start your session, the specified libraries will be installed. Once your session ends, these libraries will no longer be available as they are specific to your session.

Verify installed libraries

To verify if the correct versions of the correct libraries are installed from PyPI, run the following code:

```
import pkg_resources
for d in pkg_resources.working_set:
     print(d)
```

In some cases, to view the package versions from Conda, you may need to inspect the package version individually.

Session-scoped Java or Scala packages

To specify session-scoped Java or Scala packages, you can use the `%%configure` option:

```
%%configure -f
{
    "conf": {
        "spark.jars": "abfss://<<file system>>@<<storage account>.dfs.core.windows.net/<<path to JAR
file>>",
    }
}
```

We recommend you to run the %%configure at the beginning of your notebook. You can refer to this document for the full list of valid parameters.

Next steps

- View the default libraries: Apache Spark version support
- Manage the packages outside Synapse Studio portal: Manage packages through Az commands and REST APIs

Troubleshoot library installation errors

8/19/2022 • 5 minutes to read • Edit Online

To make third party or locally built code available to your applications, you can install a library onto one of your serverless Apache Spark pools. The packages listed in the requirements.txt file are downloaded from PyPi at the time of pool startup. This requirements file is used every time a Spark instance is created from that Spark pool. Once a library is installed for a Spark pool, it's available for all sessions using the same pool.

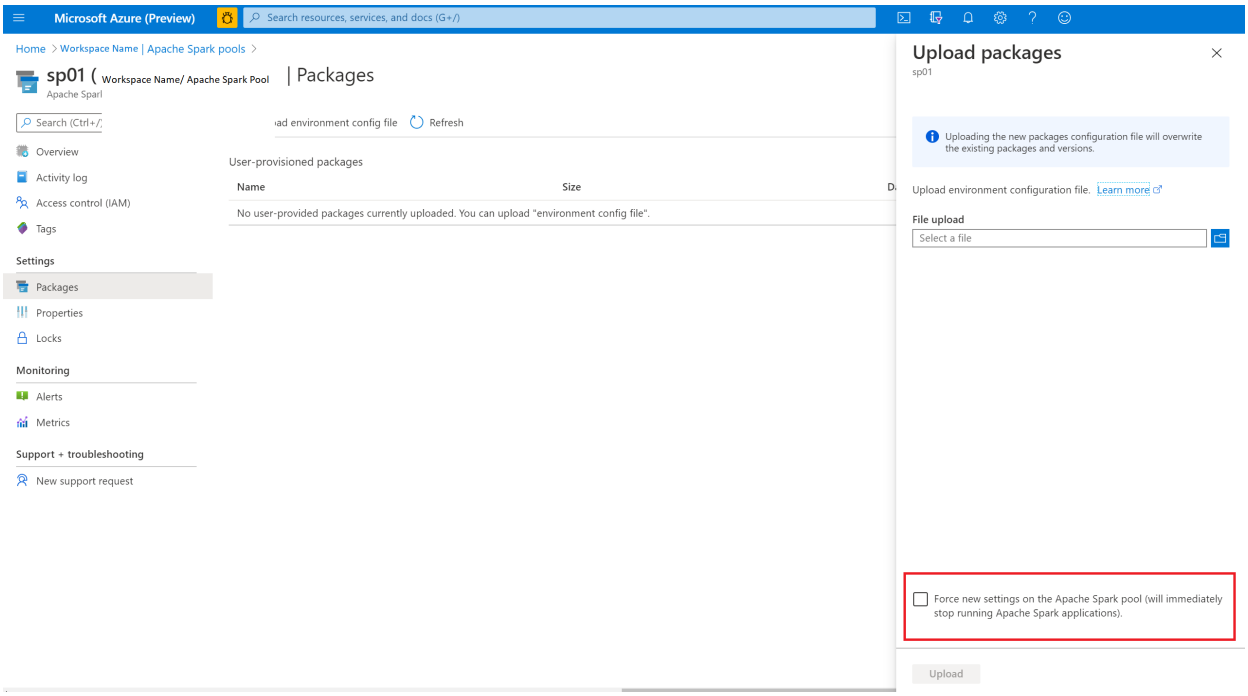
In some cases, you may find that a library isn't appearing in your Apache Spark pool. This case often occurs when there's an error in the provided requirements.txt or specified libraries. When an error occurs in the library installation process, the Apache Spark pool will revert back to libraries specified in the Synapse base runtime.

The goal of this document is to provide common issues and to help you debug library installation errors.

Force update your Apache Spark pool

When you update the libraries in your Apache Spark pool, these changes will be picked up once the pool has restarted. If you have active jobs, these jobs will continue to run on the original version of the spark pool.

You can force the changes to apply by selecting the option to **Force new settings**. This setting will end the all current sessions for the selected Spark pool. Once the sessions are ended, you'll have to wait for the pool to restart.



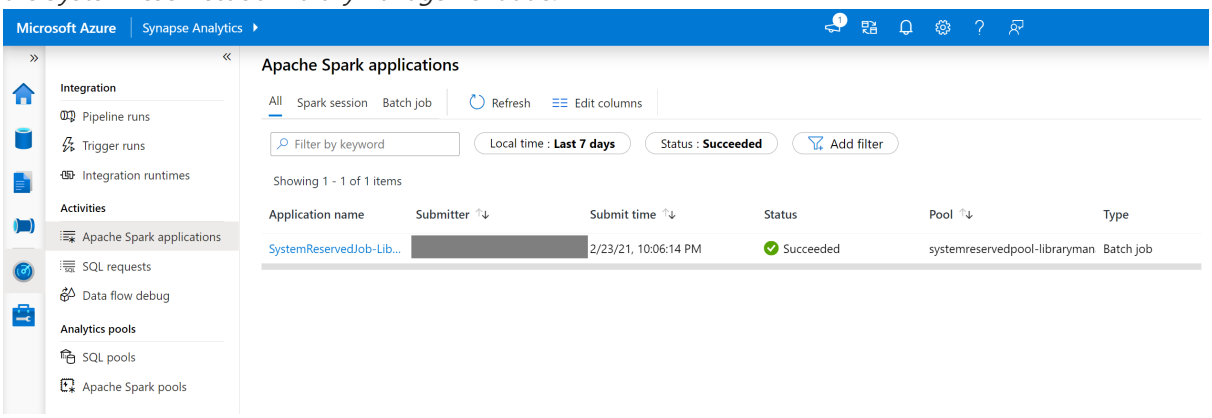
Track installation progress

A system reserved Spark job is started each time a pool is updated with a new set of libraries. This Spark job helps monitor the status of the library installation. If the installation fails because of library conflicts or other issues, the Spark pool will revert to its previous or default state.

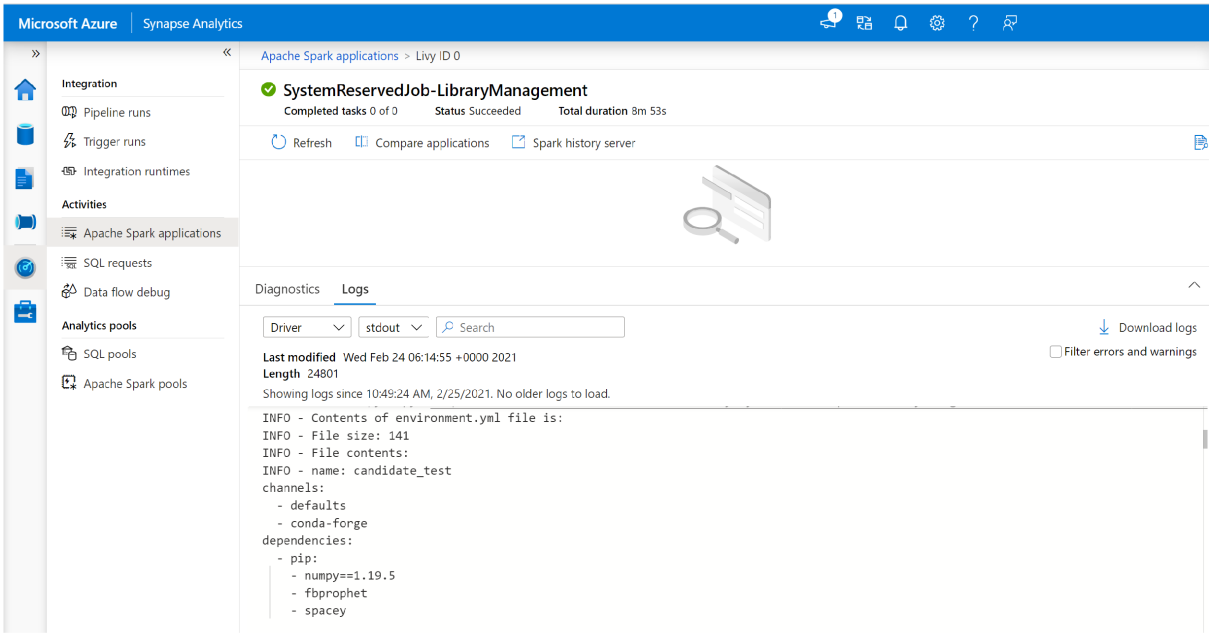
In addition, users can also inspect the installation logs to identify dependency conflicts or see which libraries were installed during the pool update.

To view these logs:

1. Navigate to the Spark applications list in the **Monitor** tab.
2. Select the system Spark application job that corresponds to your pool update. These system jobs run under the *SystemReservedJob-LibraryManagement* title.



3. Switch to view the **driver** and **stdout** logs.
4. Within the results, you'll see the logs related to the installation of your packages.

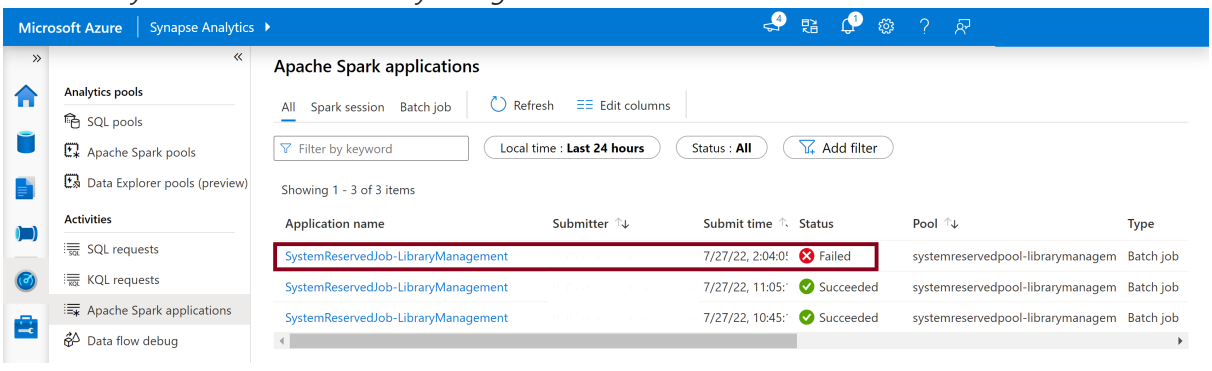


Track installation failures

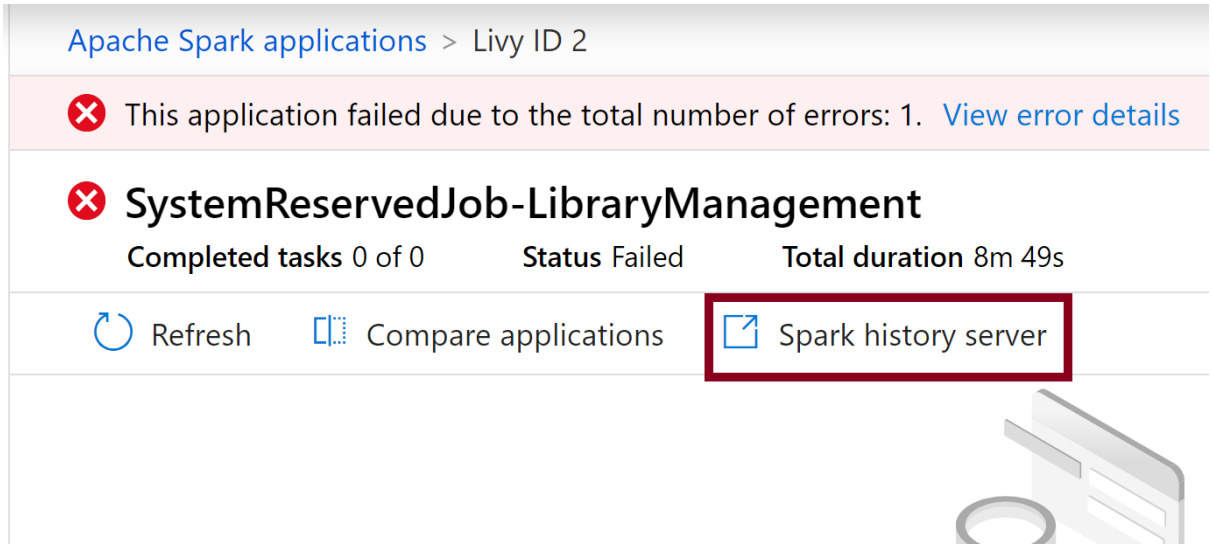
In certain cases, users can also inspect the full installation logs available in the Spark History Server to identify complicated dependency conflicts. The logs available through the Spark UI could be truncated and accessing the full installation logs through the Spark History Server would be useful in complex library installation scenarios.

To view the full installation logs:

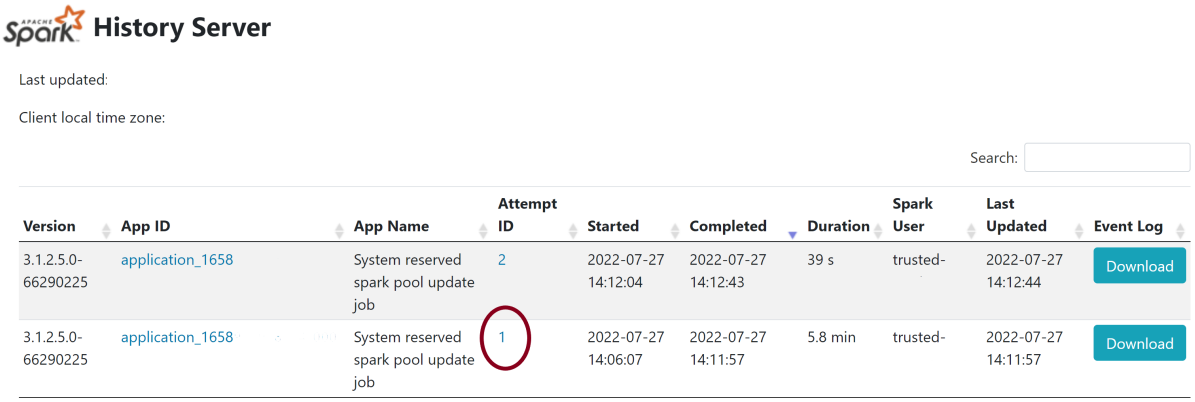
1. Navigate to the Spark applications list in the **Monitor** tab.
2. Select the system Spark application job that corresponds to the failed pool update. These system jobs run under the *SystemReservedJob-LibraryManagement* title.



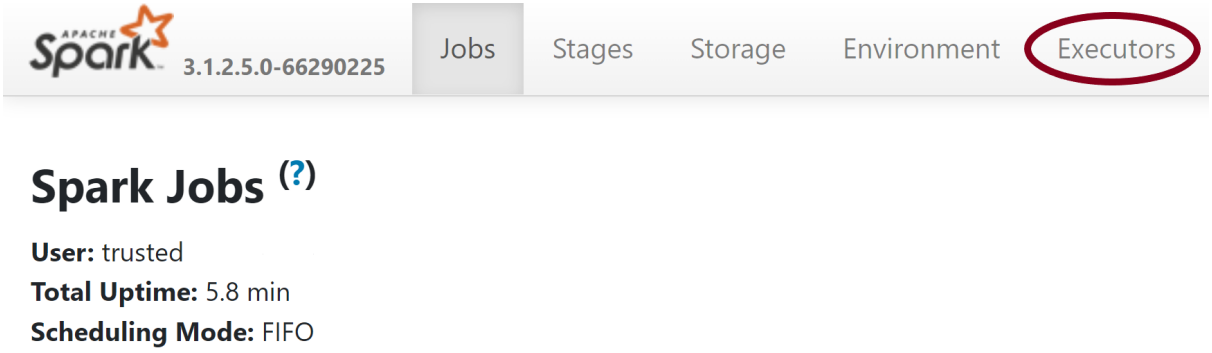
3. Select the highlighted **Spark history server** option which would open the Spark history server details page in a new tab.



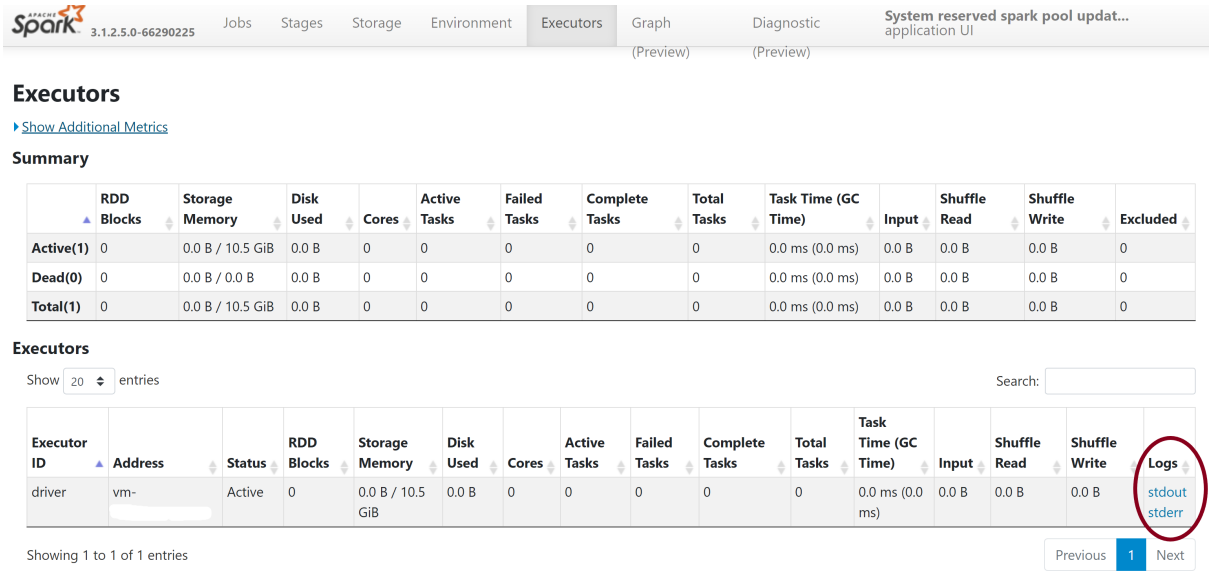
4. In this page, you will see 2 attempts, select **Attempt 1** as shown below.



5. On the top navigation bar in the Spark history server page, switch to the **Executors** tab.



6. Download the **stdout** and **stderr** log files to access the full library management output and error logs.



Validate your permissions

To install and update libraries, you must have the **Storage Blob Data Contributor** or **Storage Blob Data Owner** permissions on the primary Azure Data Lake Storage Gen2 Storage account that is linked to the Azure Synapse Analytics workspace.

To validate that you have these permissions, you can run the following code:

```
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
data2 = [("James","Smith","Joe","4355","M",3000),
    ("Michael","Rose","Edward","40288","F",4000)
  ]

schema = StructType([ \
    StructField("firstname",StringType(),True), \
    StructField("middlename",StringType(),True), \
    StructField("lastname",StringType(),True), \
    StructField("id", StringType(), True), \
    StructField("gender", StringType(), True), \
    StructField("salary", IntegerType(), True) \
  ])

df = spark.createDataFrame(data=data2,schema=schema)

df.write.csv("abfss://<<ENTER NAME OF FILE SYSTEM>>@<<ENTER NAME OF PRIMARY STORAGE
ACCOUNT>>.dfs.core.windows.net/validate_permissions.csv")
```

If you receive an error, you are likely missing the required permissions. To learn how to obtain the required permissions, visit this document: Assign Storage Blob Data Contributor or Storage Blob Data Owner permissions.

In addition, if you are running a Pipeline, then the Workspace MSI must have Storage Blob Data Owner or Storage Blob Data Contributor permissions as well. To learn how to grant your workspace identity this permission, visit: Grant permissions to workspace managed identity.

Check the environment configuration file

An environment configuration file can be used to upgrade the Conda environment. This acceptable file formats for Python pool management are listed as Environment Specifications.

It is important to note the following restrictions:

- The contents of the requirements file must not include extra blank lines or characters.
- The Synapse Runtime includes a set of libraries that are pre-installed onto every serverless Apache Spark pool. Packages that come pre-installed onto the base runtime cannot be removed or uninstalled.
- Altering the PySpark, Python, Scala/Java, .NET, or Spark version is not supported.
- Python session-scoped libraries only accepts files with a YML extension.

Validate wheel files

The Synapse serverless Apache Spark pools are based off the Linux distribution. When downloading and installing Wheel files directly from PyPI, be sure to select the version that is built on Linux and runs on the same Python version as the Spark pool.

> **IMPORTANT**
> Custom packages can be added or modified between sessions. However, you'll need to wait for the pool and session to restart to see the updated package.

Check for dependency conflicts

In general, Python dependency resolution can be tricky to manage. To help debug dependency conflicts locally, you can create your own virtual environment based off the Synapse Runtime and validate your changes.

To recreate the environment and validate your updates:

1. Download the template to locally recreate the Synapse runtime. There may be slight differences between the template and the actual Synapse environment.
2. Create a virtual environment using the following instructions. This environment allows you to create an isolated Python installation with the specified list of libraries.

   ```
   conda myenv create -f environment.yml
   conda activate myenv
   ```

3. Use `pip install -r <provide your req.txt file>` to update the virtual environment with your specified packages. If the installation results in an error, then there may be a conflict between what is pre-installed in the Synapse base runtime and what is specified in the provided requirements file. These dependency conflicts must be resolved in order to get the updated libraries on your serverless Apache Spark pool.

> **IMPORTANT**
> Issues may arrise when using pip and conda together. When combining pip and conda, it's best to follow these recommended best practices.

Next steps

- View the default libraries: Apache Spark version support

Create a custom Conda channel for package management

8/19/2022 • 3 minutes to read • Edit Online

When installing Python packages, the Conda package manager uses channels to look for packages. You may need to create a custom Conda channel for various reasons. For example, you may find that:

- your workspace is data exfiltration protected and outbound connections are blocked.
- you have packages that you don't want to upload to public repositories.
- you want to set up an alternate repository for the users within your workspace.

In this article, we'll provide a step-by-step guide to help you create your custom Conda channel within your Azure Data Lake Storage account.

Set up your local machine

1. Install Conda on your local machine.You can refer to the Azure Synapse Spark runtime to identify the Conda version that is used on the same runtime.

2. To create a custom channel, install conda-build.

```
conda install conda-build
```

3. Organize all the packages in for the platform you want to serve. In this example, we will install Anaconda archive on your local machine.

```
sudo wget https://repo.continuum.io/archive/Anaconda3-4.4.0-Linux-x86_64.sh
sudo chmod +x Anaconda3-4.4.0-Linux-x86_64.sh
sudo bash Anaconda3-4.4.0-Linux-x86_64.sh -b -p /usr/lib/anaconda3
export PATH="/usr/lib/anaconda3/bin:$PATH"
sudo chmod 777 -R /usr/lib/anaconda3
```

4. To create a similar environment to what is created available in the Azure Synapse runtime, you may download this template. There may be slight differences between the template and the actual Azure Synapse Environment. Once downloaded, you can run the following command:

```
apt-get -yq install gcc g++
conda env update --prune -f base_environment.yml
```

Mount the storage account onto your machine

Next, we will mount the Azure Data Lake Storage Gen2 account onto your local machine. This process can also be done with a WASB account; however, we will go through an example for the ADLSg2 account

For more information on how to mount the storage account on your local machine, you can visit this page.

1. You can install blobfuse from the Linux Software Repository for Microsoft products.

```
wget https://packages.microsoft.com/config/ubuntu/16.04/packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
sudo apt-get update
sudo apt-get install blobfuse fuse
export AZURE_STORAGE_ACCOUNT=<storage-account-name>
export AZURE_STORAGE_SAS_TOKEN="<SAS>"
export AZURE_STORAGE_BLOB_ENDPOINT=*.dfs.core.windows.net
```

2. Create your mountpoint (`mkdir /path/to/mount`) and mount a Blob container with blobfuse. In this example, let's use the value **privatechannel** for the **mycontainer** variable.

```
sudo mkdir /home/trusted-service-user/privatechannel
sudo mkdir -p /mnt/blobfusetmp
blobfuse /home/trusted-service-user/privatechannel --container-name=privatechannel --tmp-
path=/mnt/blobfusetmp --use-adls=true --log-level=LOG_DEBUG
sudo chown trusted-service-user /mnt/blobfusetmp
```

Create the channel

In the next set of steps, we will create a custom Conda channel.

1. On your local machine, create a directory to organize all the packages for your custom channel. Organize all the `tar.bz2` packages from https://repo.anaconda.com/pkgs/main/linux-64/ into the subdirectory. Be sure to also include all dependent tar.bz2 packages as well.

```
cd ~/privatechannel/
mkdir -p channel/linux64

<Add all .tar.bz2 from https://repo.anaconda.com/pkgs/main/linux-64/>
// Note: Add all dependent .tar.bz2 as well

cd channel
mkdir noarch
echo '{}' > noarch/repodata.json
bzip2 -k noarch/repodata.json

// Create channel
conda index channel/noarch
conda index channel/linux-64
conda index channel
```

2. Now, you may check the storage account where your `privatechannel/channel` directory would have been created.

> **NOTE**
> Conda does not honor the SAS token associated to a container. Therefore, you must mark the container "privatechannel" as public access.

For more information, you can also visit the Conda user guide to creating custom channels.

Storage account permissions

Now, we will need to validate the permissions on the storage account. To set these permissions, navigate to the path where custom channel will be created. Then, create a SAS token for `privatechannel` that has read, list, and

execute permissions.

The channel name will now be the blob SAS URL that is generated from this process.

Create a sample Conda environment configuration file

Last, verify the installation process by creating a sample Conda `environment.yml` file. If you have in a data exfiltration protection enabled workspace, you must specify the `nodefaults` channel in your environment file.

Here is an example Conda configuration file:

```
name: sample
channels:
  - https://<<storage account name>>.blob.core.windows.net/privatechannel/channel?<<SAS Token>
  - nodefaults
dependencies:
  - openssl
  - ncurses
```

Once you've created the sample Conda file, you can create a virtual Conda environment. You can verify this locally by running the following commands:

```
conda env create --file sample.yml
source activate env
conda list
```

Now that you've verified your custom channel, you can use the Python pool management process to update the libraries on your Apache Spark pool.

Next steps

- View the default libraries: Apache Spark version support
- Manage Session level Python packages: Python package management on Notebook Session

Manage packages outside Synapse Analytics Studio UIs

8/19/2022 • 2 minutes to read • Edit Online

You may want to manage your libraries for your serverless Apache Spark pools without going into the Synapse Analytics UI pages. For example, you may find that:

- you develop a custom package and want to upload it to your workspace and use it in your Spark pool. And you want to finish the steps on your local tools without visiting the package management UIs.
- you are updating your packages through the CI/CD process

In this article, we'll provide a general guide to help you managing libraries through Azure PowerShell cmdlets or REST APIs.

Manage packages through Azure PowerShell cmdlets

Add new libraries

1. New-AzSynapseWorkspacePackage command can be used to **upload new libraries to workspace**.

```
New-AzSynapseWorkspacePackage -WorkspaceName ContosoWorkspace -Package ".\ContosoPackage.whl"
```

2. The combination of New-AzSynapseWorkspacePackage and Update-AzSynapseSparkPool commands can be used to **upload new libraries to workspace** and **attach the library to a Spark pool**.

```
$package = New-AzSynapseWorkspacePackage -WorkspaceName ContosoWorkspace -Package
".\ContosoPackage.whl"
Update-AzSynapseSparkPool -WorkspaceName ContosoWorkspace -Name ContosoSparkPool -PackageAction Add -
Package $package
```

3. If you want to attach an **existing workspace library** to your Spark pool, please refer to the command combination of Get-AzSynapseWorkspacePackage and Update-AzSynapseSparkPool.

```
$packages = Get-AzSynapseWorkspacePackage -WorkspaceName ContosoWorkspace
Update-AzSynapseSparkPool -WorkspaceName ContosoWorkspace -Name ContosoSparkPool -PackageAction Add -
Package $packages
```

Remove libraries

1. In order to **remove a installed package** from your Spark pool, please refer to the command combination of Get-AzSynapseWorkspacePackage and Update-AzSynapseSparkPool.

```
$package = Get-AzSynapseWorkspacePackage -WorkspaceName ContosoWorkspace -Name ContosoPackage
Update-AzSynapseSparkPool -WorkspaceName ContosoWorkspace -Name ContosoSparkPool -PackageAction
Remove -Package $package
```

2. You can also retrieve a Spark pool and **remove all attached workspace libraries** from the pool by calling Get-AzSynapseSparkPool and Update-AzSynapseSparkPool commands.

```
$pool = Get-AzSynapseSparkPool -ResourceGroupName ContosoResourceGroup -WorkspaceName
ContosoWorkspace -Name ContosoSparkPool
$pool | Update-AzSynapseSparkPool -PackageAction Remove -Package $pool.WorkspacePackages
```

For more Azure PowerShell cmdlets capabilities, please refer to Azure PowerShell cmdlets for Azure Synapse Analytics.

Manage packages through REST APIs

Manage the workspace packages

With the ability of REST APIs, you can add/delete packages or list all uploaded files of your workspace. See the full supported APIs, please refer to Overview of workspace library APIs.

Manage the Spark pool packages

You can leverage the Spark pool REST API to attach or remove your custom or open source libraries to your Spark pools.

1. For custom libraries, please specify the list of custom files as the **customLibraries** property in request body.

```
"customLibraries": [
    {
        "name": "samplejartestfile.jar",
        "path": "<workspace-name>/libraries/<jar-name>.jar",
        "containerName": "prep",
        "uploadedTimestamp": "1970-01-01T00:00:00Z",
        "type": "jar"
    }
]
```

2. You can also update your Spark pool libraries by specifying the **libraryRequirements** property in request body.

```
"libraryRequirements": {
      "content": "",
      "filename": "requirements.txt"
}
```

Next steps

- View the default libraries: Apache Spark version support
- Manage Spark pool level packages through Synapse Studio portal: Python package management on Notebook Session

Create, develop, and maintain Synapse notebooks in Azure Synapse Analytics

8/19/2022 • 19 minutes to read • Edit Online

A Synapse notebook is a web interface for you to create files that contain live code, visualizations, and narrative text. Notebooks are a good place to validate ideas and use quick experiments to get insights from your data. Notebooks are also widely used in data preparation, data visualization, machine learning, and other Big Data scenarios.

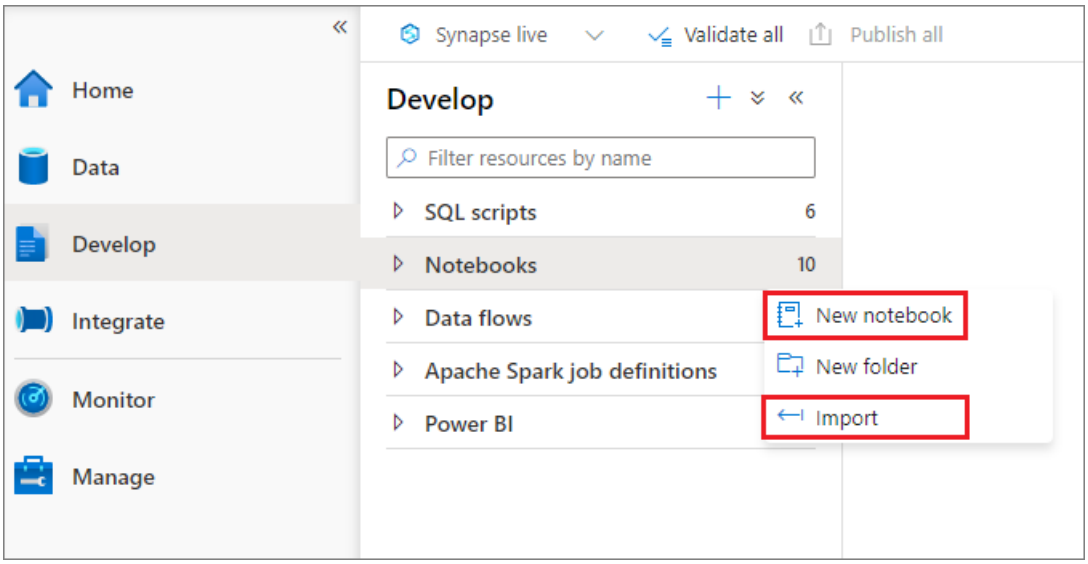
With a Synapse notebook, you can:

- Get started with zero setup effort.
- Keep data secure with built-in enterprise security features.
- Analyze data across raw formats (CSV, txt, JSON, etc.), processed file formats (parquet, Delta Lake, ORC, etc.), and SQL tabular data files against Spark and SQL.
- Be productive with enhanced authoring capabilities and built-in data visualization.

This article describes how to use notebooks in Synapse Studio.

Create a notebook

There are two ways to create a notebook. You can create a new notebook or import an existing notebook to a Synapse workspace from the **Object Explorer**. Synapse notebooks recognize standard Jupyter Notebook IPYNB files.



Develop notebooks

Notebooks consist of cells, which are individual blocks of code or text that can be run independently or as a group.

We provide rich operations to develop notebooks:

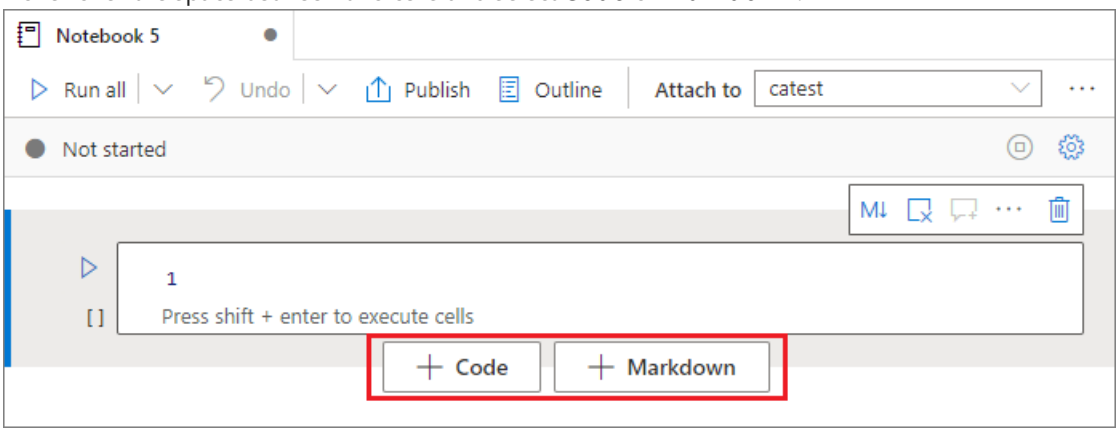
- Add a cell
- Set a primary language
- Use multiple languages

- Use temp tables to reference data across languages
- IDE-style IntelliSense
- Code Snippets
- Format text cell with toolbar buttons
- Undo/Redo cell operation
- Code cell commenting
- Move a cell
- Delete a cell
- Collapse a cell input
- Collapse a cell output
- Notebook outline

Add a cell

There are multiple ways to add a new cell to your notebook.

1. Hover over the space between two cells and select **Code** or **Markdown**.



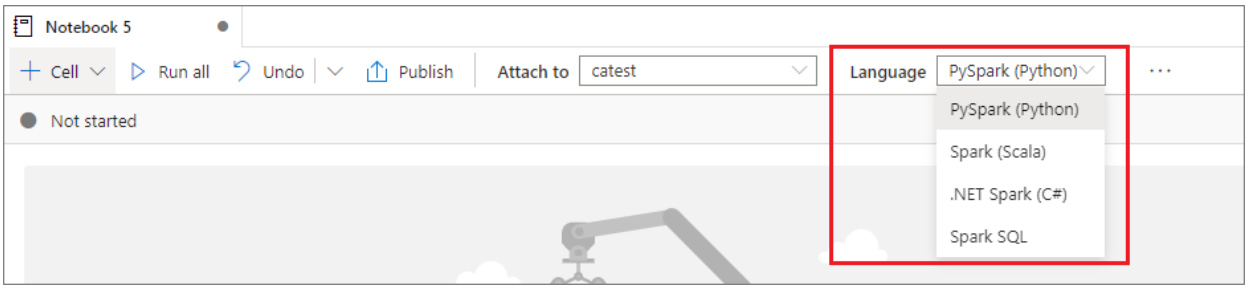
2. Use aznb Shortcut keys under command mode. Press **A** to insert a cell above the current cell. Press **B** to insert a cell below the current cell.

Set a primary language

Synapse notebooks support four Apache Spark languages:

- PySpark (Python)
- Spark (Scala)
- Spark SQL
- .NET Spark (C#)

You can set the primary language for new added cells from the dropdown list in the top command bar.

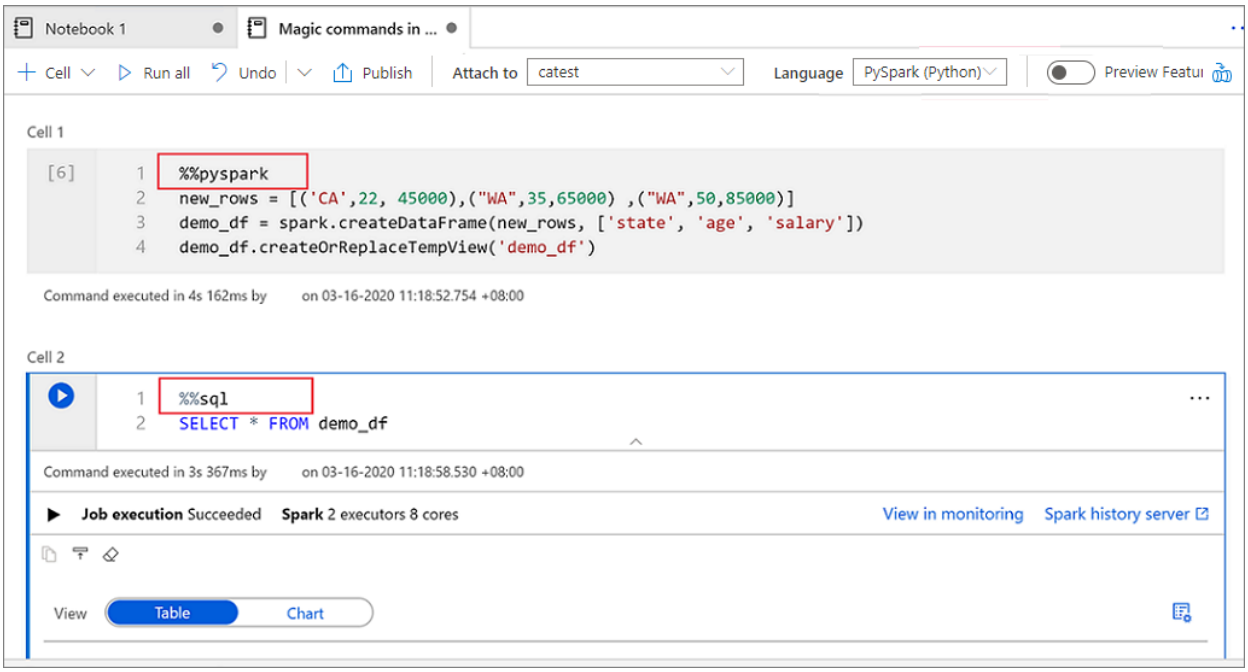


Use multiple languages

You can use multiple languages in one notebook by specifying the correct language magic command at the beginning of a cell. The following table lists the magic commands to switch cell languages.

MAGIC COMMAND	LANGUAGE	DESCRIPTION
%%pyspark	Python	Execute a **Python** query against Spark Context.
%%spark	Scala	Execute a **Scala** query against Spark Context.
%%sql	SparkSQL	Execute a **SparkSQL** query against Spark Context.
%%csharp	.NET for Spark C#	Execute a **.NET for Spark C#** query against Spark Context.

The following image is an example of how you can write a PySpark query using the **%%pyspark** magic command or a SparkSQL query with the **%%sql** magic command in a **Spark(Scala)** notebook. Notice that the primary language for the notebook is set to pySpark.



Use temp tables to reference data across languages

You cannot reference data or variables directly across different languages in a Synapse notebook. In Spark, a temporary table can be referenced across languages. Here is an example of how to read a `Scala` DataFrame in `PySpark` and `SparkSQL` using a Spark temp table as a workaround.

1. In Cell 1, read a DataFrame from a SQL pool connector using Scala and create a temporary table.

```
%%spark
val scalaDataFrame = spark.read.sqlanalytics("mySQLPoolDatabase.dbo.mySQLPoolTable")
scalaDataFrame.createOrReplaceTempView( "mydataframetable" )
```

2. In Cell 2, query the data using Spark SQL.

```
%%sql
SELECT * FROM mydataframetable
```

3. In Cell 3, use the data in PySpark.

```
%%pyspark
myNewPythonDataFrame = spark.sql("SELECT * FROM mydataframetable")
```

IDE-style IntelliSense

Synapse notebooks are integrated with the Monaco editor to bring IDE-style IntelliSense to the cell editor. Syntax highlight, error marker, and automatic code completions help you to write code and identify issues quicker.

The IntelliSense features are at different levels of maturity for different languages. Use the following table to see what's supported.

LANGUAGES	SYNTAX HIGHLIGHT	SYNTAX ERROR MARKER	SYNTAX CODE COMPLETION	VARIABLE CODE COMPLETION	SYSTEM FUNCTION CODE COMPLETION	USER FUNCTION CODE COMPLETION	SMART INDENT	CODE FOLDING
PySpark (Python)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Spark (Scala)	Yes	Yes	Yes	Yes	Yes	Yes	-	Yes
SparkSQL	Yes	Yes	Yes	Yes	Yes	-	-	-
.NET for Spark (C#)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

> **NOTE**
> An active Spark session is required to benefit the Variable Code Completion, System Function Code Completion, User Function Code Completion for .NET for Spark (C#).

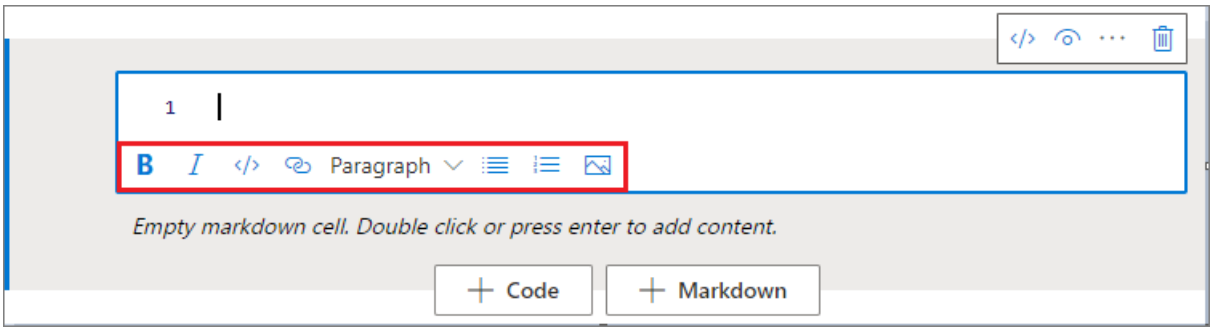
Code Snippets

Synapse notebooks provide code snippets that make it easier to enter common used code patterns, such as configuring your Spark session, reading data as a Spark DataFrame, or drawing charts with matplotlib etc.

Snippets appear in Shortcut keys of IDE style IntelliSense mixed with other suggestions. The code snippets contents align with the code cell language. You can see available snippets by typing **Snippet** or any keywords appear in the snippet title in the code cell editor. For example, by typing **read** you can see the list of snippets to read data from various data sources.

Format text cell with toolbar buttons

You can use the format buttons in the text cells toolbar to do common markdown actions. It includes bolding text, italicizing text, paragraph/headers through a dropdown, inserting code, inserting unordered list, inserting ordered list, inserting hyperlink and inserting image from URL.



Undo/Redo cell operation

Select the **Undo** / **Redo** button or press **Z** / **Shift+Z** to revoke the most recent cell operations. Now you can undo/redo up to the latest 10 historical cell operations.

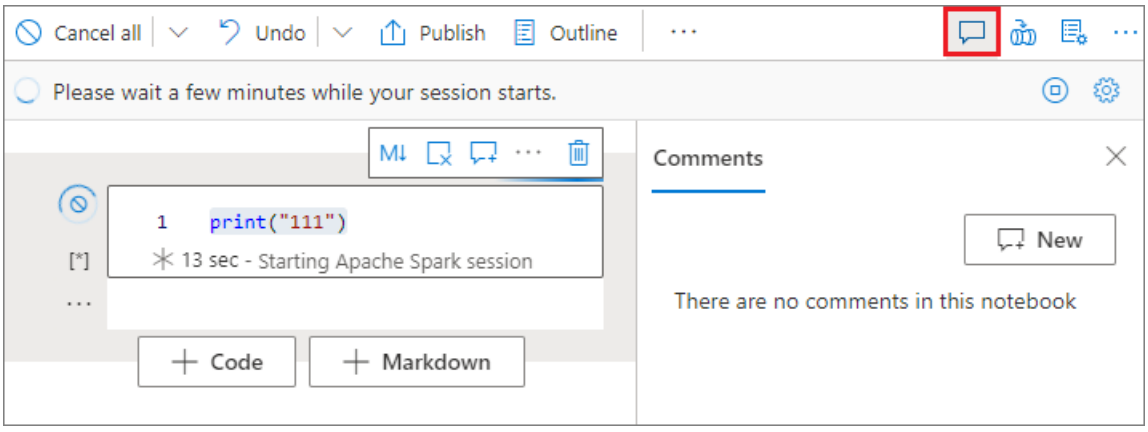
Supported undo cell operations:

- Insert/Delete cell: You could revoke the delete operations by selecting **Undo**, the text content will be kept along with the cell.
- Reorder cell.
- Toggle parameter.
- Convert between Code cell and Markdown cell.

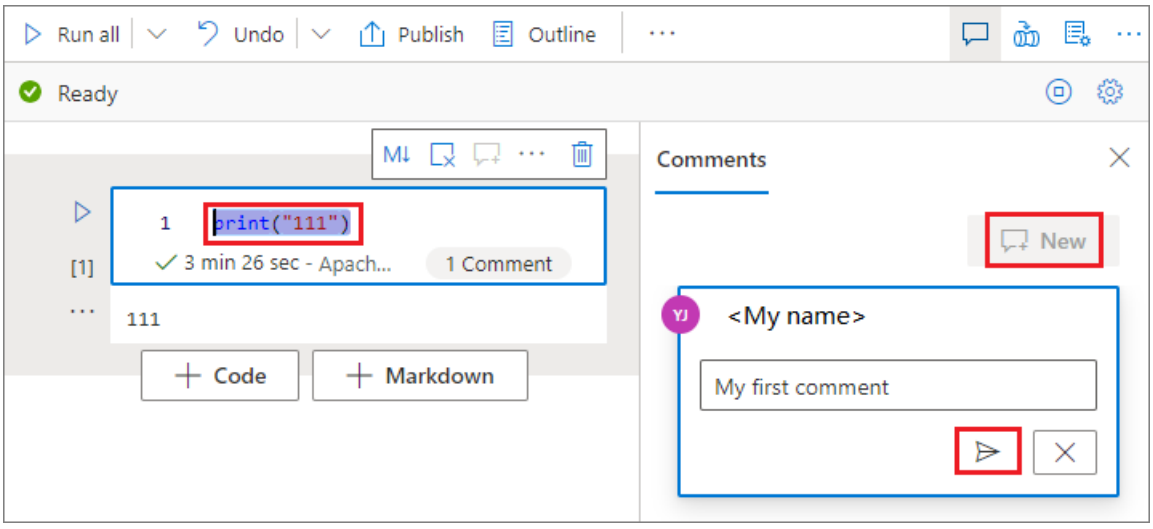
> **NOTE**
> In-cell text operations and code cell commenting operations are not undoable. Now you can undo/redo up to the latest 10 historical cell operations.

Code cell commenting

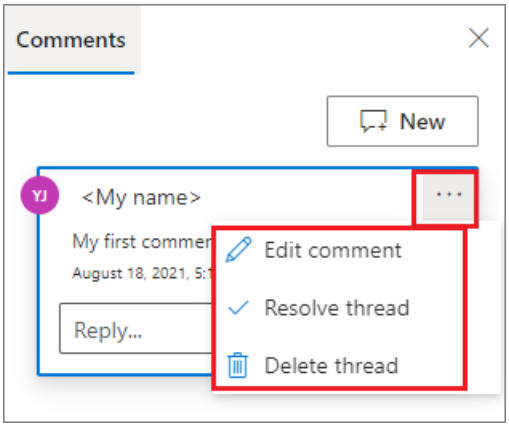
1. Select **Comments** button on the notebook toolbar to open **Comments** pane.



2. Select code in the code cell, click **New** in the **Comments** pane, add comments then click **Post comment** button to save.

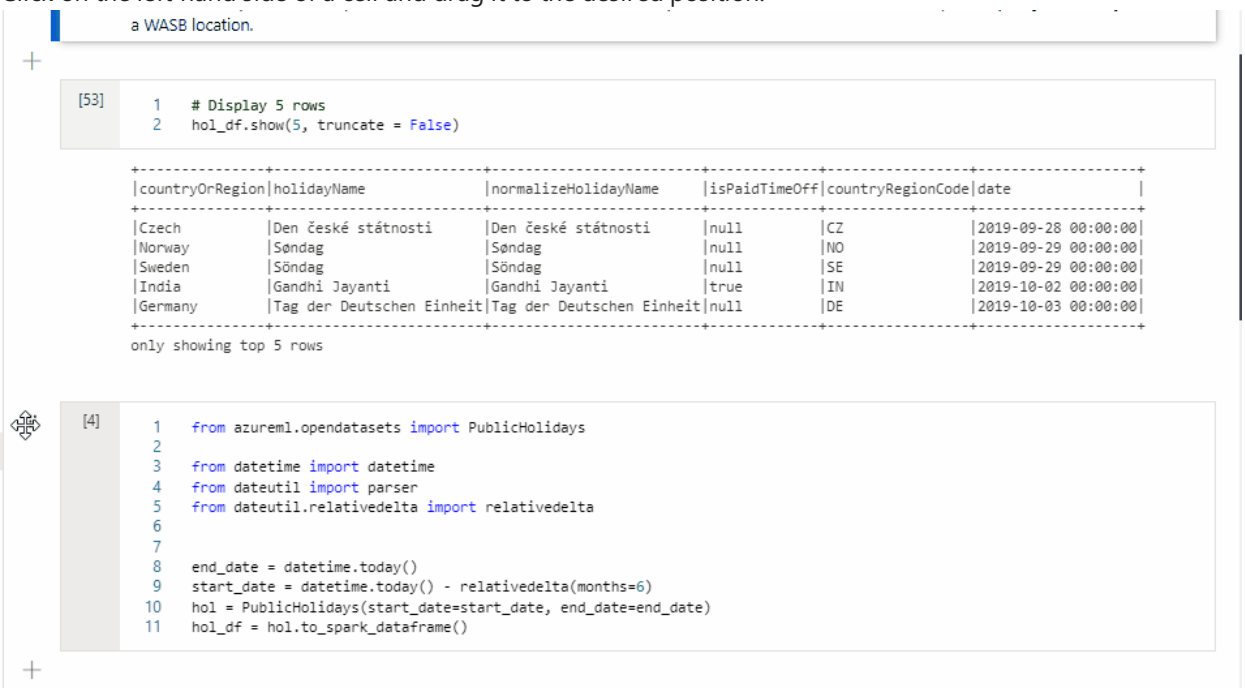


3. You could perform **Edit comment**, **Resolve thread**, or **Delete thread** by clicking the **More** button besides your comment.



Move a cell

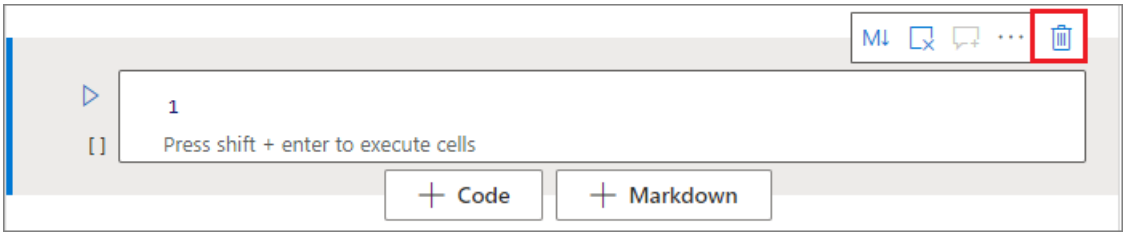
Click on the left-hand side of a cell and drag it to the desired position.



Delete a cell

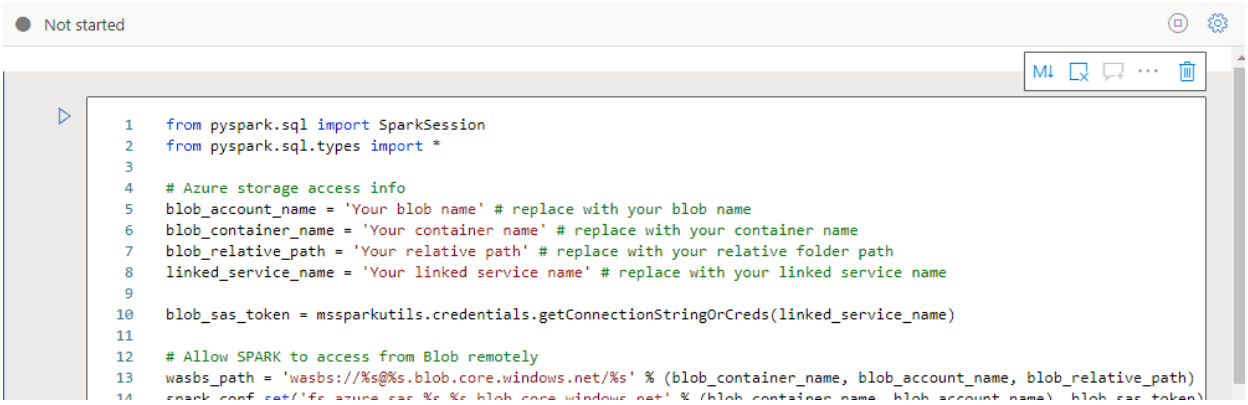
To delete a cell, select the delete button at the right hand of the cell.

You can also use shortcut keys under command mode. Press **Shift+D** to delete the current cell.



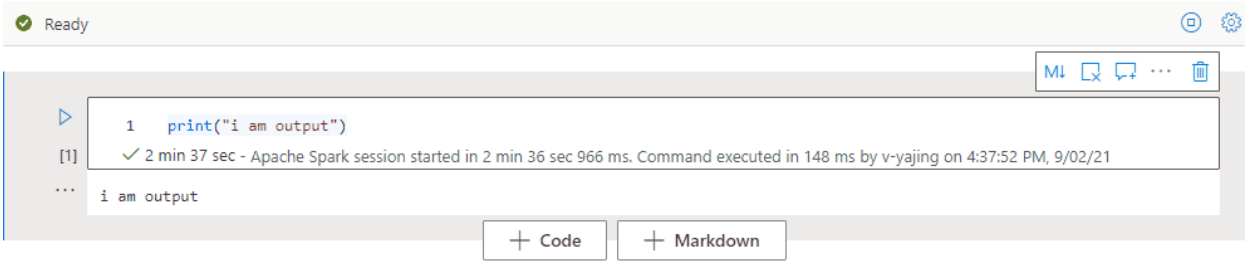
Collapse a cell input

Select the **More commands** ellipses (...) on the cell toolbar and **Hide input** to collapse current cell's input. To expand it, Select the **Show input** while the cell is collapsed.



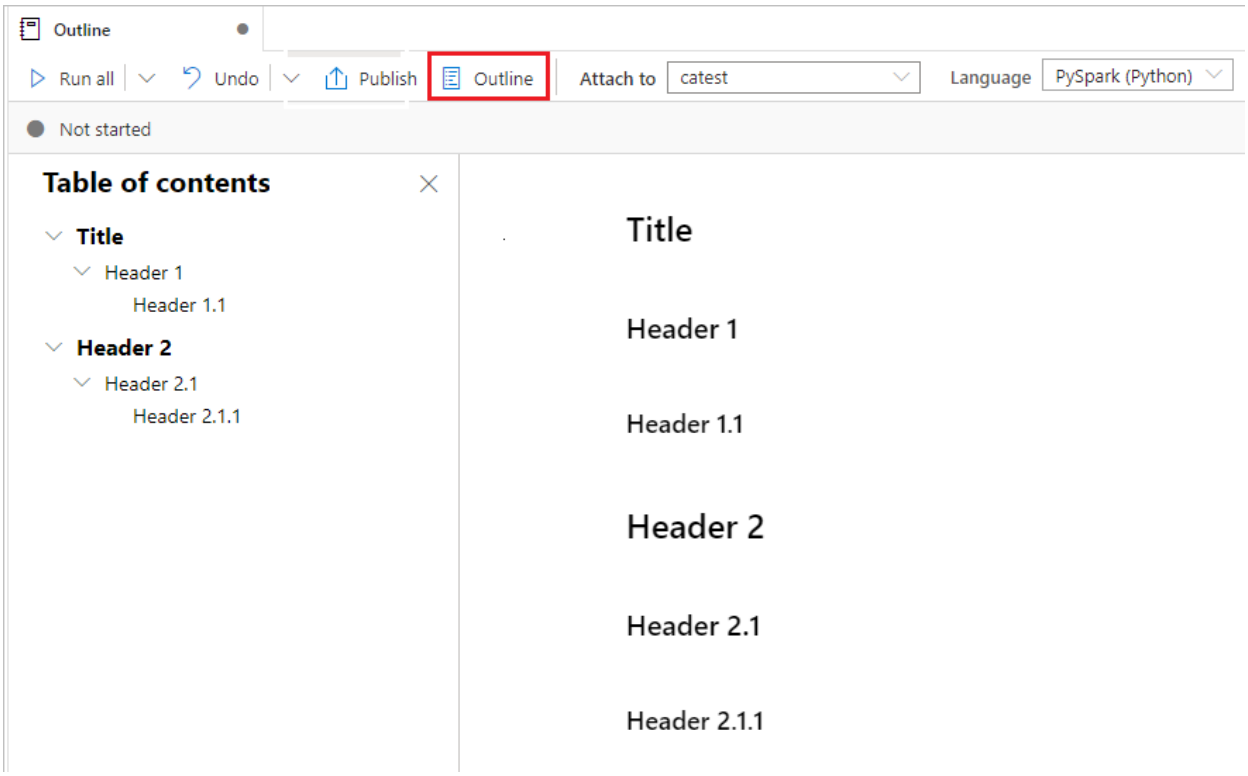
Collapse a cell output

Select the **More commands** ellipses (...) on the cell toolbar and **Hide output** to collapse current cell's output. To expand it, select the **Show output** while the cell's output is hidden.



Notebook outline

The Outlines (Table of Contents) presents the first markdown header of any markdown cell in a sidebar window for quick navigation. The Outlines sidebar is resizable and collapsible to fit the screen in the best ways possible. You can select the **Outline** button on the notebook command bar to open or hide sidebar



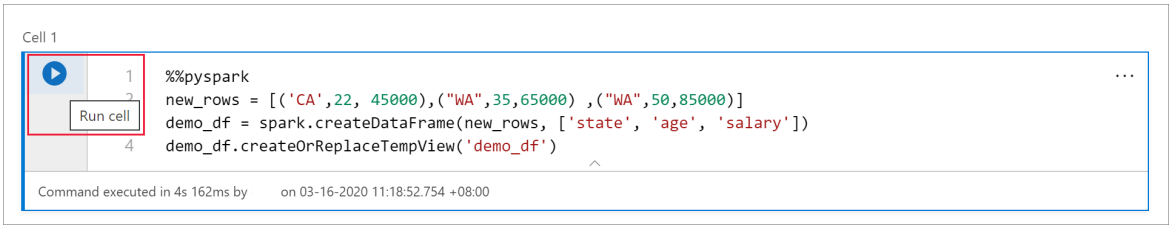
Run notebooks

You can run the code cells in your notebook individually or all at once. The status and progress of each cell is represented in the notebook.

Run a cell

There are several ways to run the code in a cell.

1. Hover on the cell you want to run and select the **Run Cell** button or press **Ctrl+Enter**.



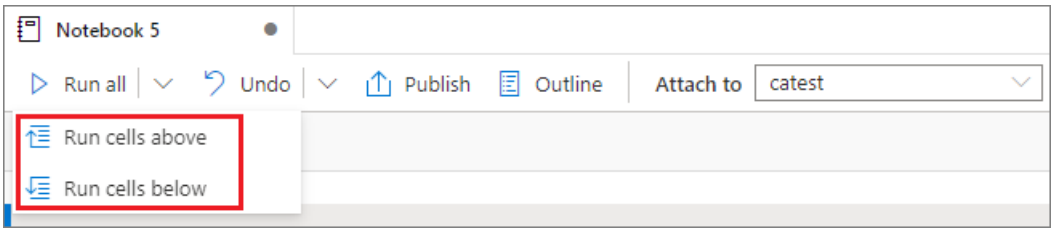
2. Use Shortcut keys under command mode. Press **Shift+Enter** to run the current cell and select the cell below. Press **Alt+Enter** to run the current cell and insert a new cell below.

Run all cells

Select the **Run All** button to run all the cells in current notebook in sequence.

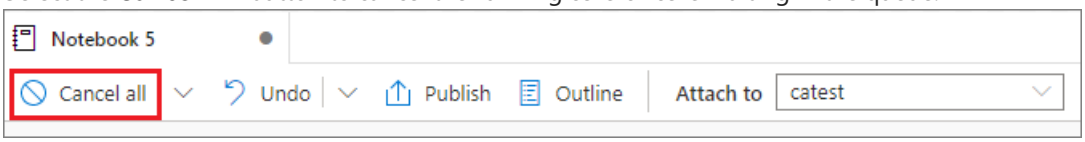
Run all cells above or below

Expand the dropdown list from **Run all** button, then select **Run cells above** to run all the cells above the current in sequence. Select **Run cells below** to run all the cells below the current in sequence.



Cancel all running cells

Select the **Cancel All** button to cancel the running cells or cells waiting in the queue.



Notebook reference

You can use `%run <notebook path>` magic command to reference another notebook within current notebook's context. All the variables defined in the reference notebook are available in the current notebook. `%run` magic command supports nested calls but not support recursive calls. You will receive an exception if the statement depth is larger than **five**.

Example:

```
%run /<path>/Notebook1 { "parameterInt": 1, "parameterFloat": 2.5, "parameterBool": true, "parameterString": "abc" }
```

.

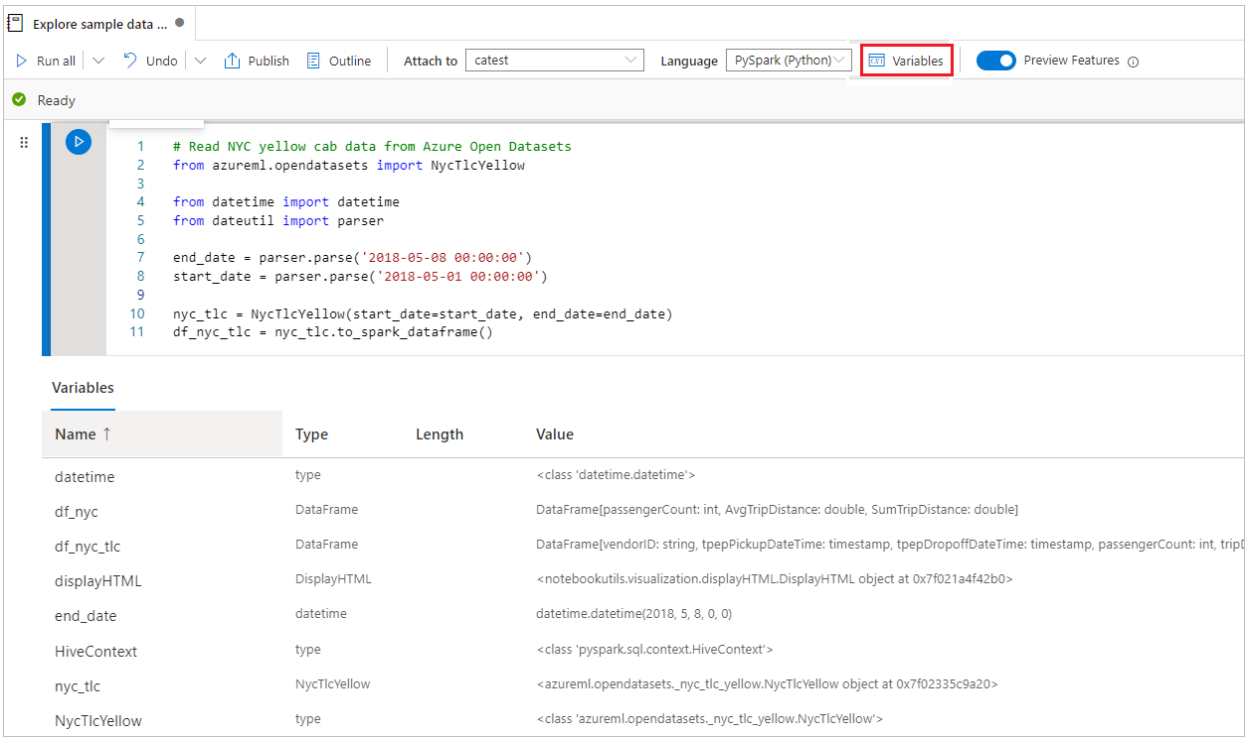
Notebook reference works in both interactive mode and Synapse pipeline.

> **NOTE**
>
> - `%run` command currently only supports to pass a absolute path or notebook name only as parameter, relative path is not supported.
> - `%run` command currently only supports to 4 parameter value types: `int`, `float`, `bool`, `string`, variable replacement operation is not supported.
> - The referenced notebooks are required to be published. You need to publish the notebooks to reference them unless Reference unpublished notebook is enabled. Synapse Studio does not recognize the unpublished notebooks from the Git repo.
> - Referenced notebooks do not support statement that depth is larger than **five**.

Variable explorer

Synapse notebook provides a built-in variables explorer for you to see the list of the variables name, type, length, and value in the current Spark session for PySpark (Python) cells. More variables will show up automatically as they are defined in the code cells. Clicking on each column header will sort the variables in the table.

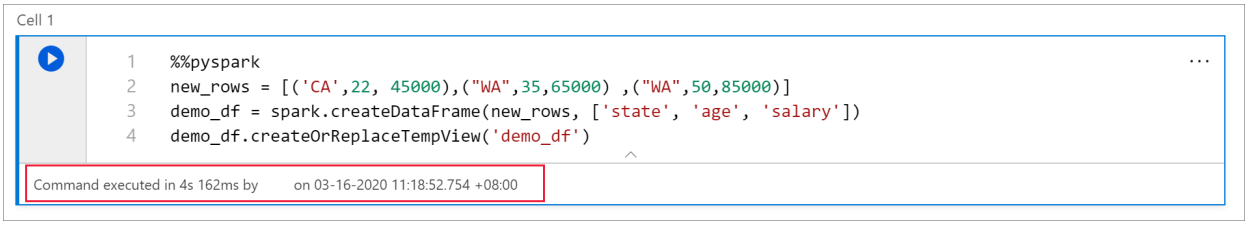
You can select the **Variables** button on the notebook command bar to open or hide the variable explorer.



> **NOTE**
>
> Variable explorer only supports python.

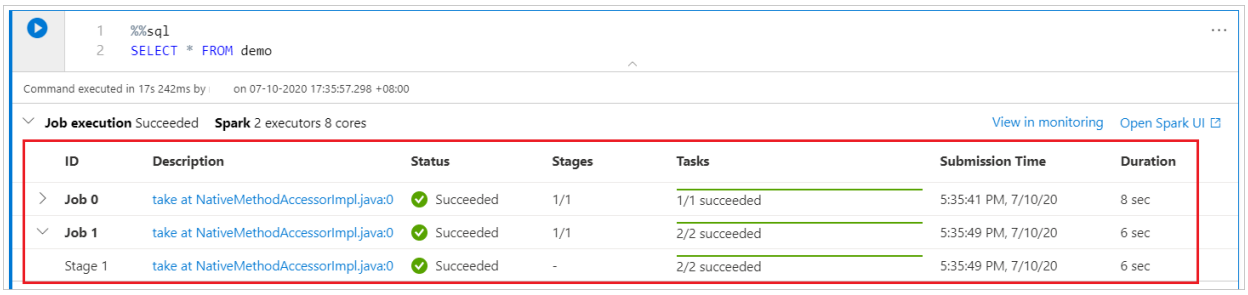
Cell status indicator

A step-by-step cell execution status is displayed beneath the cell to help you see its current progress. Once the cell run is complete, an execution summary with the total duration and end time are shown and kept there for future reference.



Spark progress indicator

Synapse notebook is purely Spark based. Code cells are executed on the serverless Apache Spark pool remotely. A Spark job progress indicator is provided with a real-time progress bar appears to help you understand the job execution status. The number of tasks per each job or stage help you to identify the parallel level of your spark job. You can also drill deeper to the Spark UI of a specific job (or stage) via selecting the link on the job (or stage) name.

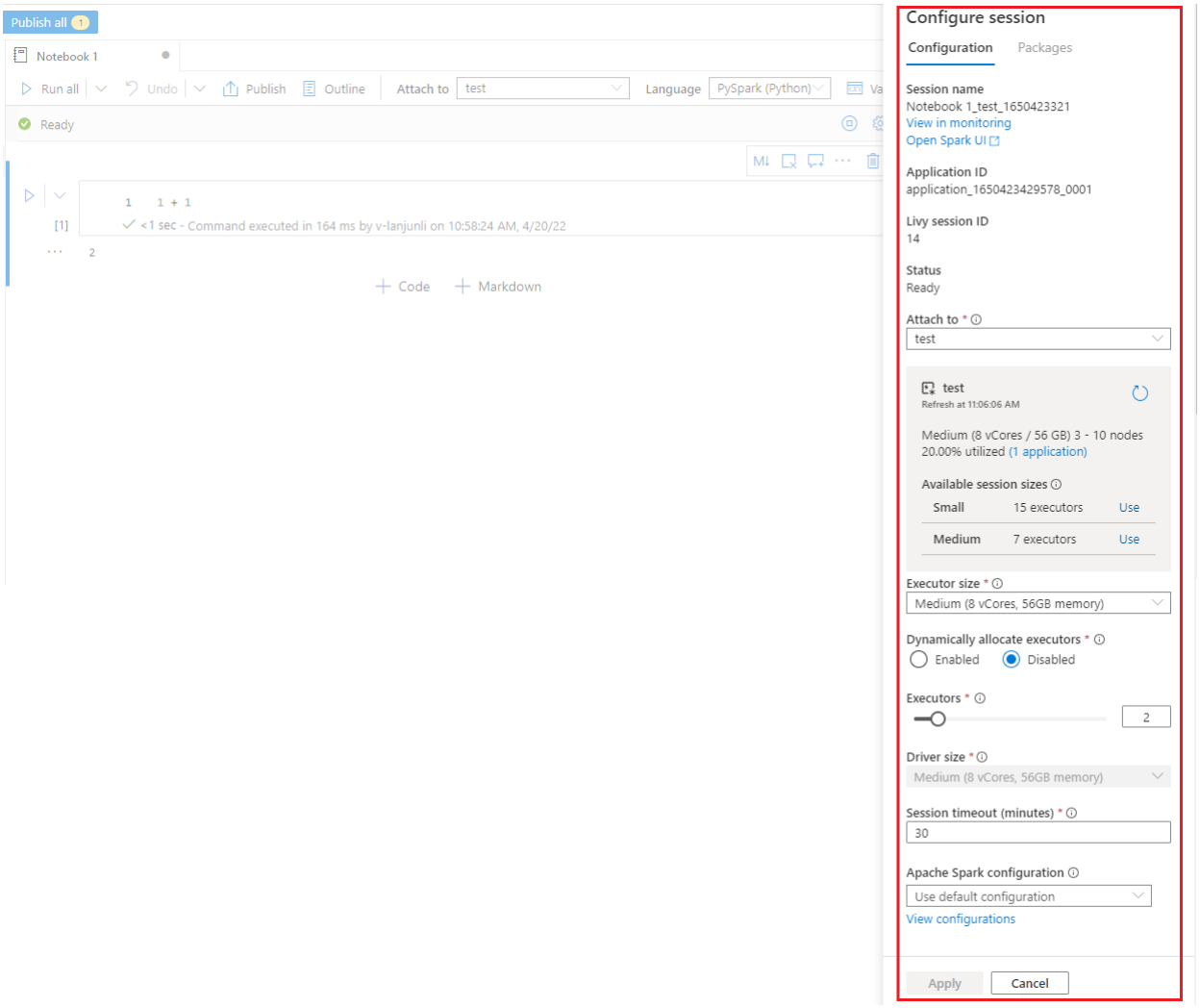


Spark session configuration

You can specify the timeout duration, the number, and the size of executors to give to the current Spark session

in **Configure session**. Restart the Spark session is for configuration changes to take effect. All cached notebook variables are cleared.

You can also create a configuration from the Apache Spark configuration or select an existing configuration. For details, please refer to Apache Spark Configuration Management.



Spark session configuration magic command

You can also specify spark session settings via a magic command **%%configure**. The spark session needs to restart to make the settings effect. We recommend you to run the **%%configure** at the beginning of your notebook. Here is a sample, refer to https://github.com/cloudera/livy#request-body for full list of valid parameters.

```
%%configure
{
    //You can get a list of valid parameters to config the session from
https://github.com/cloudera/livy#request-body.
    "driverMemory":"28g", // Recommended values: ["28g", "56g", "112g", "224g", "400g", "472g"]
    "driverCores":4, // Recommended values: [4, 8, 16, 32, 64, 80]
    "executorMemory":"28g",
    "executorCores":4,
    "jars":
["abfs[s]://<file_system>@<account_name>.dfs.core.windows.net/<path>/myjar.jar","wasb[s]://<containername>@<
accountname>.blob.core.windows.net/<path>/myjar1.jar"],
    "conf":{
    //Example of standard spark property, to find more available properties please
visit:https://spark.apache.org/docs/latest/configuration.html#application-properties.
        "spark.driver.maxResultSize":"10g",
    //Example of customized property, you can specify count of lines that Spark SQL returns by configuring
"livy.rsc.sql.num-rows".
        "livy.rsc.sql.num-rows":"3000"
    }
}
```

NOTE

- "DriverMemory" and "ExecutorMemory" are recommended to set as same value in %%configure, so do "driverCores" and "executorCores".
- You can use %%configure in Synapse pipelines, but if it's not set in the first code cell, the pipeline run will fail due to cannot restart session.
- The %%configure used in mssparkutils.notebook.run is going to be ignored but used in %run notebook will continue executing.
- The standard Spark configuration properties must be used in the "conf" body. We do not support first level reference for the Spark configuration properties.
- Some special spark properties including "spark.driver.cores", "spark.executor.cores", "spark.driver.memory", "spark.executor.memory", "spark.executor.instances" won't take effect in "conf" body.

Parameterized session configuration from pipeline

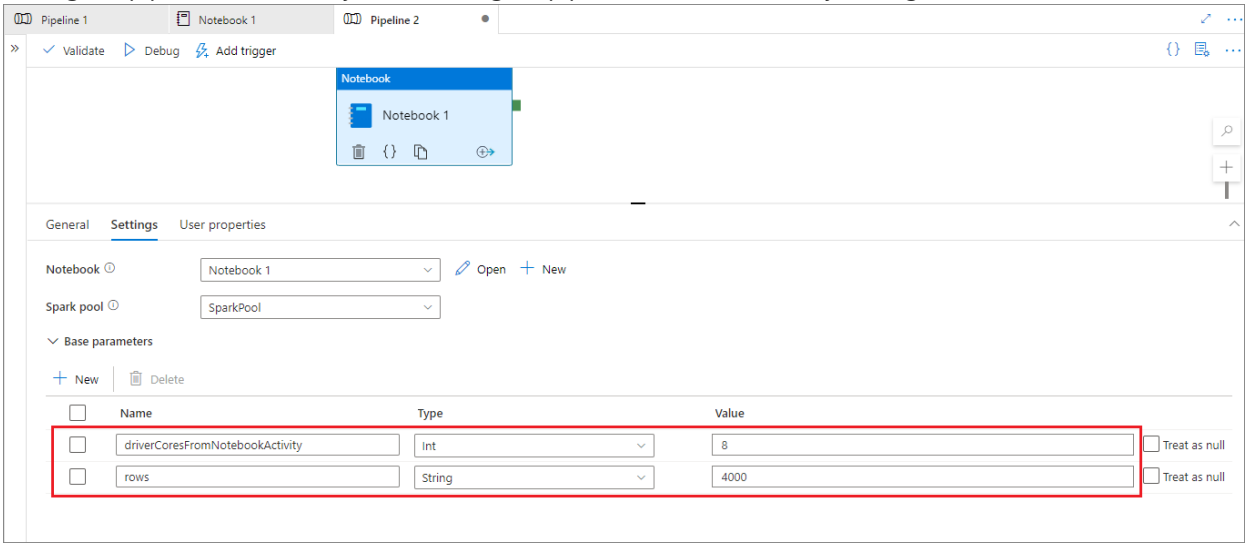
Parameterized session configuration allows you to replace the value in %%configure magic with Pipeline run (Notebook activity) parameters. When preparing %%configure code cell, you can override default values (also configurable, 4 and "2000" in the below example) with an object like this:

```
{
      "activityParameterName": "paramterNameInPipelineNotebookActivity",
      "defaultValue": "defaultValueIfNoParamterFromPipelineNotebookActivity"
}
```

```
%%configure

{
    "driverCores":
    {
        "activityParameterName": "driverCoresFromNotebookActivity",
        "defaultValue": 4
    },
    "conf":
    {
        "livy.rsc.sql.num-rows":
        {
            "activityParameterName": "rows",
            "defaultValue": "2000"
        }
    }
}
```

Notebook will use default value if run a notebook in interactive mode directly or no parameter that match "activityParameterName" is given from Pipeline Notebook activity.

During the pipeline run mode, you can configure pipeline Notebook activity settings as below:



If you want to change the session configuration, pipeline Notebook activity parameters name should be same as activityParameterName in the notebook. When running this pipeline, in this example driverCores in %%configure will be replaced by 8 and livy.rsc.sql.num-rows will be replaced by 4000.

> **NOTE**
> If run pipeline failed because of using this new %%configure magic, you can check more error information by running %%configure magic cell in the interactive mode of the notebook.

Bring data to a notebook

You can load data from Azure Blob Storage, Azure Data Lake Store Gen 2, and SQL pool as shown in the code samples below.

Read a CSV from Azure Data Lake Store Gen2 as a Spark DataFrame

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
account_name = "Your account name"
container_name = "Your container name"
relative_path = "Your path"
adls_path = 'abfss://%s@%s.dfs.core.windows.net/%s' % (container_name, account_name, relative_path)

df1 = spark.read.option('header', 'true') \
                .option('delimiter', ',') \
                .csv(adls_path + '/Testfile.csv')
```

Read a CSV from Azure Blob Storage as a Spark DataFrame

```
from pyspark.sql import SparkSession

# Azure storage access info
blob_account_name = 'Your account name' # replace with your blob name
blob_container_name = 'Your container name' # replace with your container name
blob_relative_path = 'Your path' # replace with your relative folder path
linked_service_name = 'Your linked service name' # replace with your linked service name

blob_sas_token = mssparkutils.credentials.getConnectionStringOrCreds(linked_service_name)

# Allow SPARK to access from Blob remotely

wasb_path = 'wasbs://%s@%s.blob.core.windows.net/%s' % (blob_container_name, blob_account_name,
blob_relative_path)

spark.conf.set('fs.azure.sas.%s.%s.blob.core.windows.net' % (blob_container_name, blob_account_name),
blob_sas_token)
print('Remote blob path: ' + wasb_path)

df = spark.read.option("header", "true") \
            .option("delimiter","|") \
            .schema(schema) \
            .csv(wasbs_path)
```

Read data from the primary storage account

You can access data in the primary storage account directly. There's no need to provide the secret keys. In Data Explorer, right-click on a file and select **New notebook** to see a new notebook with data extractor autogenerated.

IPython Widgets

Widgets are eventful python objects that have a representation in the browser, often as a control like a slider, textbox etc. IPython Widgets only works in Python environment, it's not supported in other languages (e.g. Scala, SQL, C#) yet.

To use IPython Widget

1. You need to import `ipywidgets` module first to use the Jupyter Widget framework.

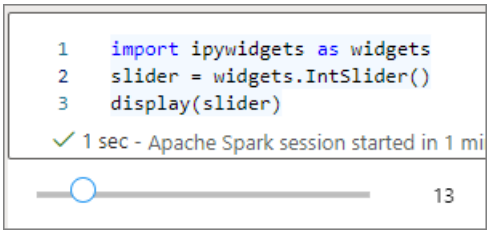
```
import ipywidgets as widgets
```

2. You can use top-level `display` function to render a widget, or leave an expression of **widget** type at the last line of code cell.

```
slider = widgets.IntSlider()
display(slider)
```

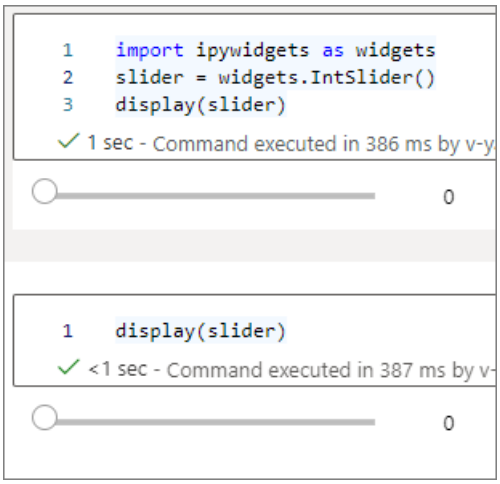
```
slider = widgets.IntSlider()
slider
```

3. Run the cell, the widget will display at the output area.



4. You can use multiple `display()` calls to render the same widget instance multiple times, but they will remain in sync with each other.

```
slider = widgets.IntSlider()
display(slider)
display(slider)
```



5. To render two widgets independent of each other, create two widget instances:

```
slider1 = widgets.IntSlider()
slider2 = widgets.IntSlider()
display(slider1)
display(slider2)
```

Supported widgets

WIDGETS TYPE	WIDGETS
Numeric widgets	IntSlider, FloatSlider, FloatLogSlider, IntRangeSlider, FloatRangeSlider, IntProgress, FloatProgress, BoundedIntText, BoundedFloatText, IntText, FloatText
Boolean widgets	ToggleButton, Checkbox, Valid
Selection widgets	Dropdown, RadioButtons, Select, SelectionSlider, SelectionRangeSlider, ToggleButtons, SelectMultiple
String Widgets	Text, Text area, Combobox, Password, Label, HTML, HTML Math, Image, Button
Play (Animation) widgets	Date picker, Color picker, Controller
Container/Layout widgets	Box, HBox, VBox, GridBox, Accordion, Tabs, Stacked

Known limitations

1. The following widgets are not supported yet, you could follow the corresponding workaround as below:

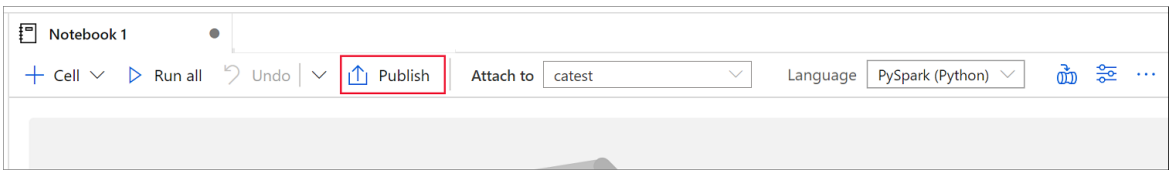
FUNCTIONALITY	WORKAROUND
`Output` widget	You can use `print()` function instead to write text into stdout.
`widgets.jslink()`	You can use `widgets.link()` function to link two similar widgets.
`FileUpload` widget	Not support yet.

2. Global `display` function provided by Synapse does not support displaying multiple widgets in 1 call (i.e. `display(a, b)`), which is different from IPython `display` function.
3. If you close a notebook that contains IPython Widget, you will not be able to see or interact with it until you execute the corresponding cell again.

Save notebooks

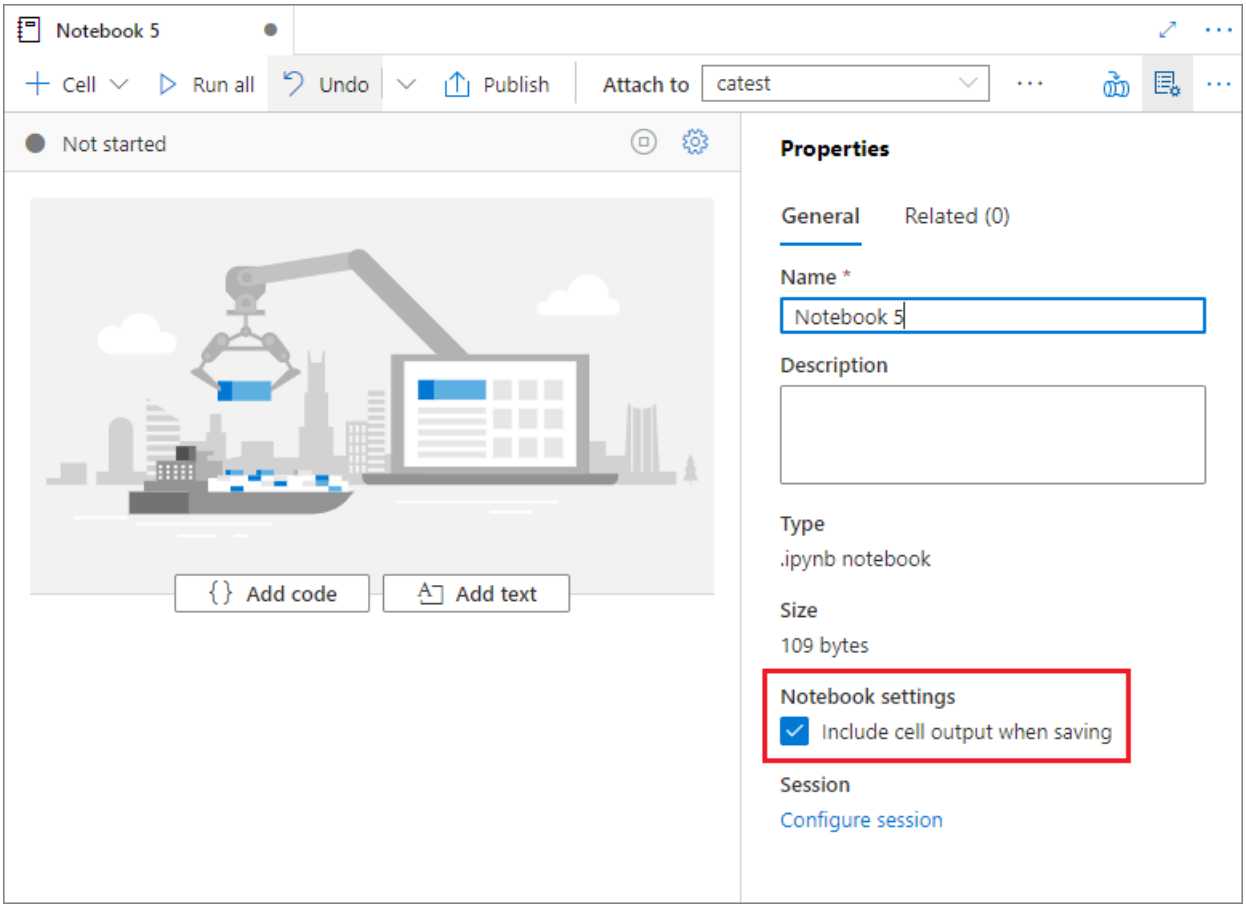
You can save a single notebook or all notebooks in your workspace.

1. To save changes you made to a single notebook, select the **Publish** button on the notebook command bar.



2. To save all notebooks in your workspace, select the **Publish all** button on the workspace command bar.

In the notebook properties, you can configure whether to include the cell output when saving.



Magic commands

You can use familiar Jupyter magic commands in Synapse notebooks. Review the following list as the current available magic commands. Tell us your use cases on GitHub so that we can continue to build out more magic commands to meet your needs.

> **NOTE**
> Only following magic commands are supported in Synapse pipeline : %%pyspark, %%spark, %%csharp, %%sql.

Available line magics: %lsmagic, %time, %timeit, %history, %run, %load

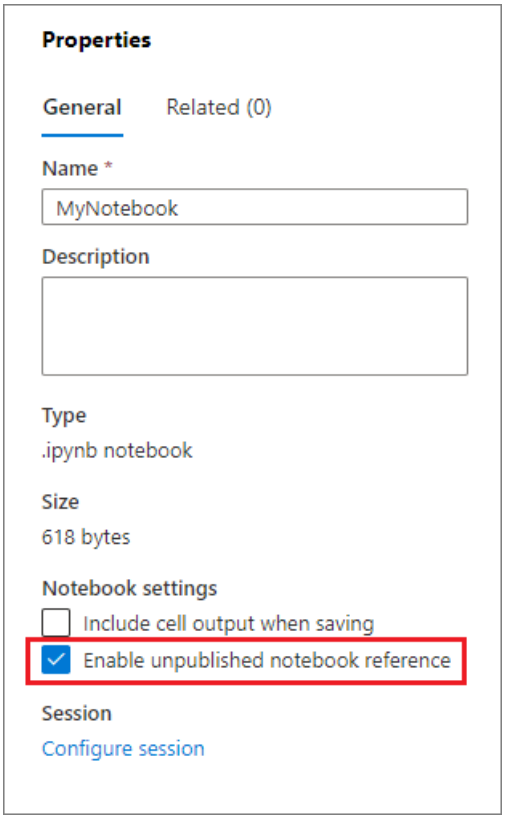
Available cell magics: %%time, %%timeit, %%capture, %%writefile, %%sql, %%pyspark, %%spark, %%csharp, %%html, %%configure

Reference unpublished notebook

Reference unpublished notebook is helpful when you want to debug "locally", when enabling this feature, notebook run will fetch the current content in web cache, if you run a cell including a reference notebooks statement, you will reference the presenting notebooks in the current notebook browser instead of a saved versions in cluster, that means the changes in your notebook editor can be referenced immediately by other

notebooks without having to be published(Live mode) or committed(Git mode), by leveraging this approach you can easily avoid common libraries getting polluted during developing or debugging process.

You can enable Reference unpublished notebook from Properties panel:



For different cases comparison please check the table below:

Notice that %run and mssparkutils.notebook.run has same behavior here. We use `%run` here as an example.

CASE	DISABLE	ENABLE
Live Mode		
- Nb1 (Published) `%run Nb1`	Run published version of Nb1	Run published version of Nb1
- Nb1 (New) `%run Nb1`	Error	Run new Nb1
- Nb1 (Previously published, edited) `%run Nb1`	Run **published** version of Nb1	Run **edited** version of Nb1
Git Mode		
- Nb1 (Published) `%run Nb1`	Run published version of Nb1	Run published version of Nb1
- Nb1 (New) `%run Nb1`	Error	Run new Nb1
- Nb1 (Not published, committed) `%run Nb1`	Error	Run committed Nb1

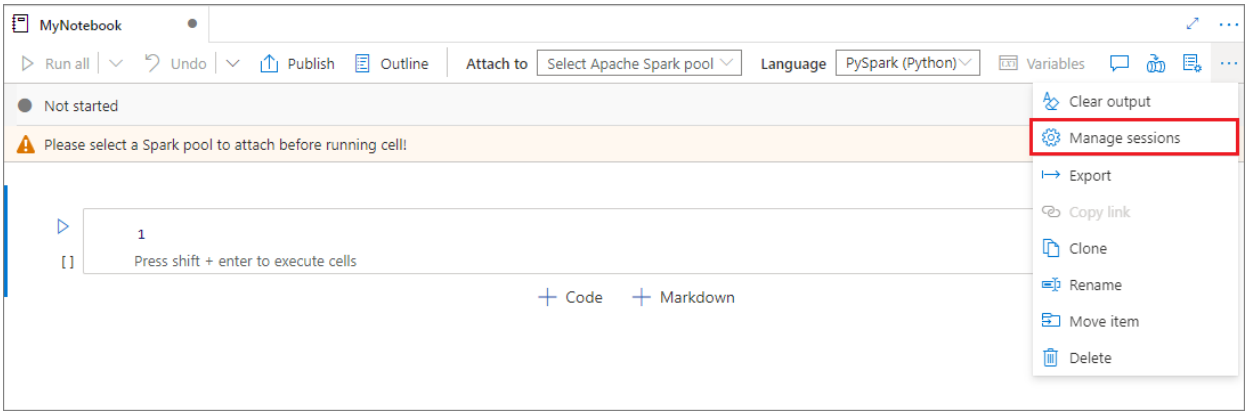
CASE	DISABLE	ENABLE
- Nb1 (Previously published, committed) `%run Nb1`	Run **published** version of Nb1	Run **committed** version of Nb1
- Nb1 (Previously published, new in current branch) `%run Nb1`	Run **published** version of Nb1	Run **new** Nb1
- Nb1 (Not published, previously committed, edited) `%run Nb1`	Error	Run **edited** version of Nb1
- Nb1 (Previously published and committed, edited) `%run Nb1`	Run **published** version of Nb1	Run **edited** version of Nb1

Conclusion

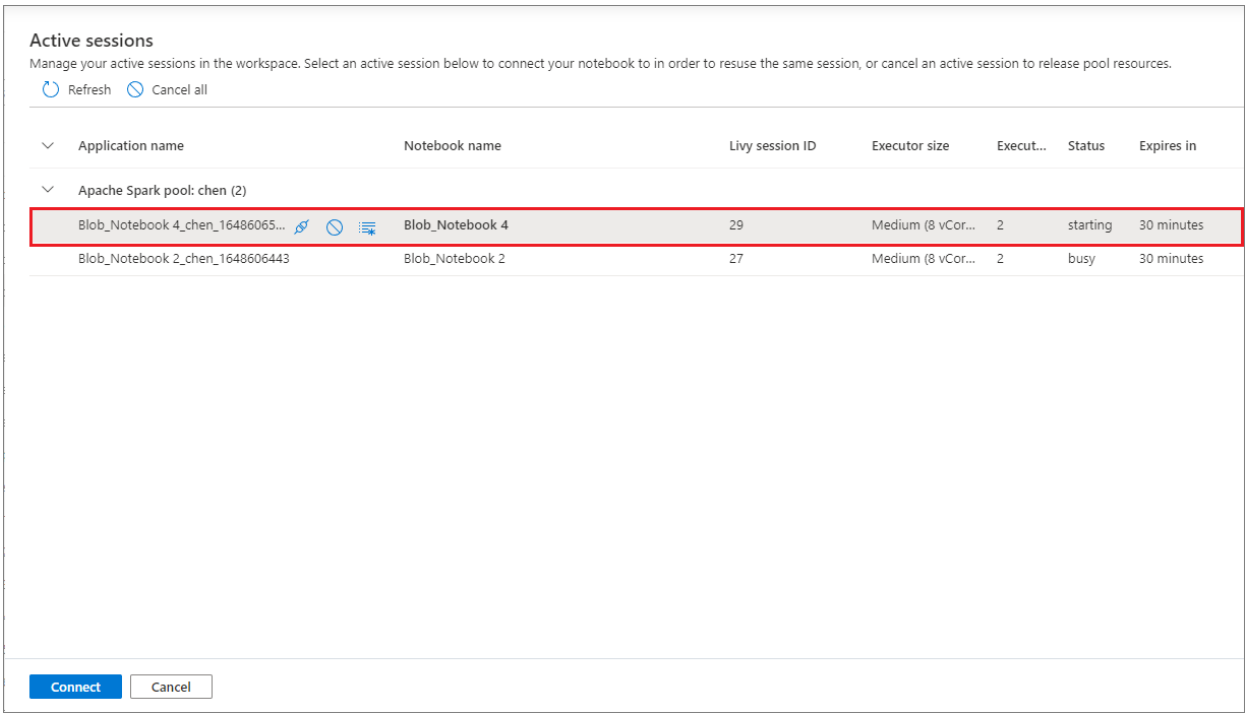
- If disabled, always run **published** version.
- If enabled, priority is: edited / new > committed > published.

Active session management

You can reuse your notebook sessions conveniently now without having to start new ones. Synapse notebook now supports managing your active sessions in the **Manage sessions** list, you can see all the sessions in the current workspace started by you from notebook.



In the **Active sessions** list you can see the session information and the corresponding notebook that is currently attached to the session. You can operate Detach with notebook, Stop the session, and View in monitoring from here. Moreover, you can easily connect your selected notebook to an active session in the list started from another notebook, the session will be detached from the previous notebook (if it's not idle) then attach to the current one.



Python logging in Notebook

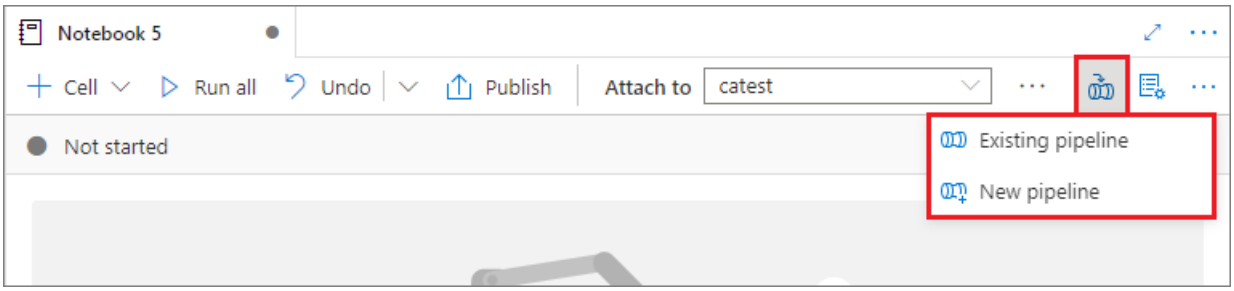
You can find Python logs and set different log levels and format following the sample code below:

```
import logging

# Customize the logging format for all loggers
FORMAT = "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
formatter = logging.Formatter(fmt=FORMAT)
for handler in logging.getLogger().handlers:
    handler.setFormatter(formatter)

# Customize log level for all loggers
logging.getLogger().setLevel(logging.INFO)

# Customize the log level for a specific logger
customizedLogger = logging.getLogger('customized')
customizedLogger.setLevel(logging.WARNING)

# logger that use the default global log level
defaultLogger = logging.getLogger('default')
defaultLogger.debug("default debug message")
defaultLogger.info("default info message")
defaultLogger.warning("default warning message")
defaultLogger.error("default error message")
defaultLogger.critical("default critical message")

# logger that use the customized log level
customizedLogger.debug("customized debug message")
customizedLogger.info("customized info message")
customizedLogger.warning("customized warning message")
customizedLogger.error("customized error message")
customizedLogger.critical("customized critical message")
```

Integrate a notebook

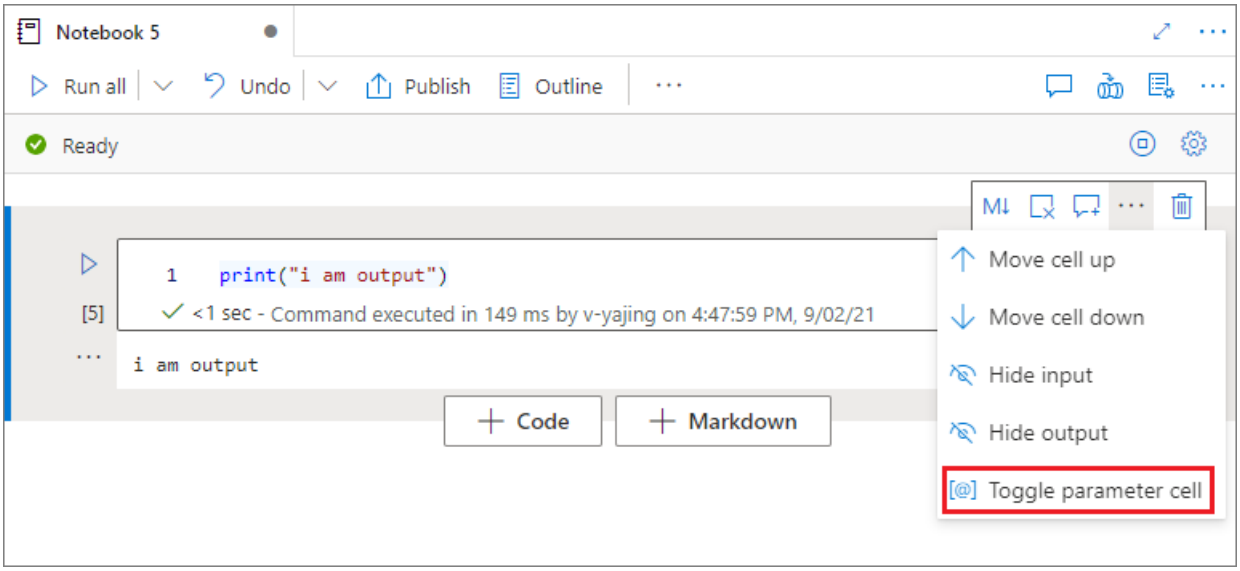
Add a notebook to a pipeline

Select the **Add to pipeline** button on the upper right corner to add a notebook to an existing pipeline or create a new pipeline.



Designate a parameters cell

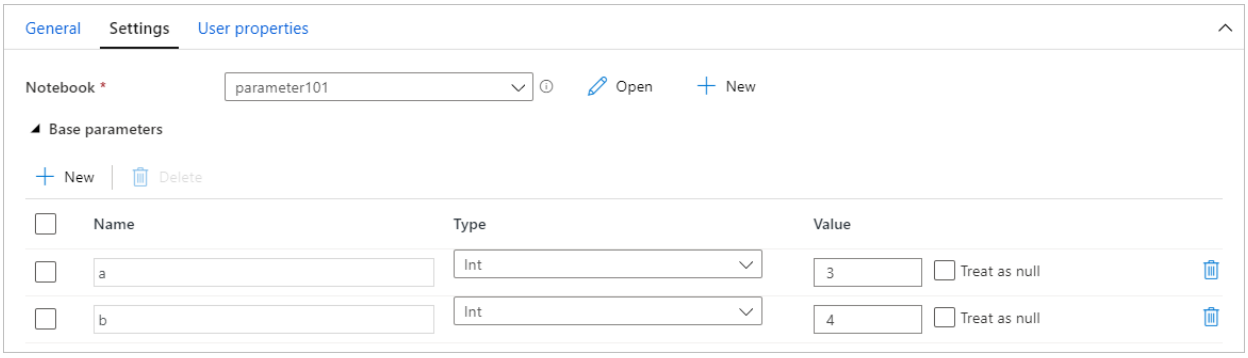
To parameterize your notebook, select the ellipses (...) to access the **more commands** at the cell toolbar. Then select **Toggle parameter cell** to designate the cell as the parameters cell.



Azure Data Factory looks for the parameters cell and treats this cell as defaults for the parameters passed in at execution time. The execution engine will add a new cell beneath the parameters cell with input parameters in order to overwrite the default values.

Assign parameters values from a pipeline

Once you've created a notebook with parameters, you can execute it from a pipeline with the Synapse Notebook activity. After you add the activity to your pipeline canvas, you will be able to set the parameters values under **Base parameters** section on the **Settings** tab.

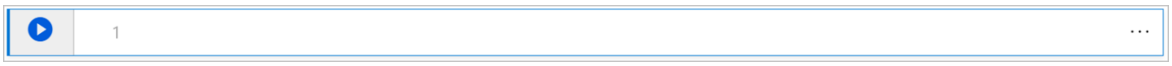


When assigning parameter values, you can use the pipeline expression language or system variables.

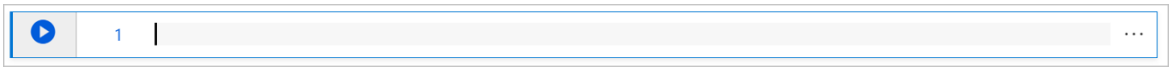
Shortcut keys

Similar to Jupyter Notebooks, Synapse notebooks have a modal user interface. The keyboard does different things depending on which mode the notebook cell is in. Synapse notebooks support the following two modes for a given code cell: command mode and edit mode.

1. A cell is in command mode when there is no text cursor prompting you to type. When a cell is in Command mode, you can edit the notebook as a whole but not type into individual cells. Enter command mode by pressing `ESC` or using the mouse to select outside of a cell's editor area.



2. Edit mode is indicated by a text cursor prompting you to type in the editor area. When a cell is in edit mode, you can type into the cell. Enter edit mode by pressing `Enter` or using the mouse to select on a cell's editor area.



Shortcut keys under command mode

ACTION	SYNAPSE NOTEBOOK SHORTCUTS
Run the current cell and select below	Shift+Enter
Run the current cell and insert below	Alt+Enter
Run current cell	Ctrl+Enter
Select cell above	Up
Select cell below	Down
Select previous cell	K
Select next cell	J
Insert cell above	A
Insert cell below	B
Delete selected cells	Shift+D
Switch to edit mode	Enter

Shortcut keys under edit mode

Using the following keystroke shortcuts, you can more easily navigate and run code in Synapse notebooks when in Edit mode.

ACTION	SYNAPSE NOTEBOOK SHORTCUTS
Move cursor up	Up
Move cursor down	Down
Undo	Ctrl + Z
Redo	Ctrl + Y

ACTION	SYNAPSE NOTEBOOK SHORTCUTS
Comment/Uncomment	Ctrl + /
Delete word before	Ctrl + Backspace
Delete word after	Ctrl + Delete
Go to cell start	Ctrl + Home
Go to cell end	Ctrl + End
Go one word left	Ctrl + Left
Go one word right	Ctrl + Right
Select all	Ctrl + A
Indent	Ctrl +]
Dedent	Ctrl + [
Switch to command mode	Esc

Next steps

- Check out Synapse sample notebooks
- Quickstart: Create an Apache Spark pool in Azure Synapse Analytics using web tools
- What is Apache Spark in Azure Synapse Analytics
- Use .NET for Apache Spark with Azure Synapse Analytics
- .NET for Apache Spark documentation
- Azure Synapse Analytics

Visualize data

8/19/2022 • 6 minutes to read • Edit Online

Azure Synapse is an integrated analytics service that accelerates time to insight, across data warehouses and big data analytics systems. Data visualization is a key component in being able to gain insight into your data. It helps make big and small data easier for humans to understand. It also makes it easier to detect patterns, trends, and outliers in groups of data.

When using Apache Spark in Azure Synapse Analytics, there are various built-in options to help you visualize your data, including Synapse notebook chart options, access to popular open-source libraries, and integration with Synapse SQL and Power BI.

Notebook chart options

When using an Azure Synapse notebook, you can turn your tabular results view into a customized chart using chart options. Here, you can visualize your data without having to write any code.

display(df) function

The `display` function allows you to turn SQL queries and Apache Spark dataframes and RDDs into rich data visualizations.The `display` function can be used on dataframes or RDDs created in PySpark, Scala, Java, and .NET.

To access the chart options:

1. The output of `%%sql` magic commands appear in the rendered table view by default. You can also call `display(df)` on Spark DataFrames or Resilient Distributed Datasets (RDD) function to produce the rendered table view.
2. Once you have a rendered table view, switch to the Chart View.
3. You can now customize your visualization by specifying the following values:

CONFIGURATION	DESCRIPTION
Chart Type	The `display` function supports a wide range of chart types, including bar charts, scatter plots, line graphs, and more
Key	Specify the range of values for the x-axis
Value	Specify the range of values for the y-axis values
Series Group	Used to determine the groups for the aggregation
Aggregation	Method to aggregate data in your visualization

> **NOTE**
>
> By default the `display(df)` function will only take the first 1000 rows of the data to render the charts. Check the **Aggregation over all results** and click the **Apply** button, you will apply the chart generation from the whole dataset. A Spark job will be triggered when the chart setting changes. Please note that it may take several minutes to complete the calculation and render the chart.

4. Once done, you can view and interact with your final visualization!

display(df) statistic details

You can use `display(df, summary = true)` to check the statistics summary of a given Apache Spark DataFrame that include the column name, column type, unique values, and missing values for each column. You can also select on specific column to see its minimum value, maximum value, mean value and standard deviation.

displayHTML() option

Azure Synapse Analytics notebooks support HTML graphics using the `displayHTML` function.

The following image is an example of creating visualizations using D3.js.

Run the following code to create the visualization above.

```
displayHTML("""<!DOCTYPE html>
<meta charset="utf-8">

<!-- Load d3.js -->
<script src="https://d3js.org/d3.v4.js"></script>

<!-- Create a div where the graph will take place -->
<div id="my_dataviz"></div>
<script>

// set the dimensions and margins of the graph
var margin = {top: 10, right: 30, bottom: 30, left: 40},
  width = 400 - margin.left - margin.right,
  height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select("#my_dataviz")
.append("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
.append("g")
  .attr("transform",
        "translate(" + margin.left + "," + margin.top + ")");

// Create Data
var data = [12,19,11,13,12,22,13,4,15,16,18,19,20,12,11,9]

// Compute summary statistics used for the box:
var data_sorted = data.sort(d3.ascending)
var q1 = d3.quantile(data_sorted, .25)
var median = d3.quantile(data_sorted, .5)
var q3 = d3.quantile(data_sorted, .75)
var interQuantileRange = q3 - q1
var min = q1 - 1.5 * interQuantileRange
var max = q1 + 1.5 * interQuantileRange

// Show the Y scale
var y = d3.scaleLinear()
```

```
  .domain([0,24])
  .range([height, 0]);
svg.call(d3.axisLeft(y))

// a few features for the box
var center = 200
var width = 100

// Show the main vertical line
svg
.append("line")
  .attr("x1", center)
  .attr("x2", center)
  .attr("y1", y(min) )
  .attr("y2", y(max) )
  .attr("stroke", "black")

// Show the box
svg
.append("rect")
  .attr("x", center - width/2)
  .attr("y", y(q3) )
  .attr("height", (y(q1)-y(q3)) )
  .attr("width", width )
  .attr("stroke", "black")
  .style("fill", "#69b3a2")

// show median, min and max horizontal lines
svg
.selectAll("toto")
.data([min, median, max])
.enter()
.append("line")
  .attr("x1", center-width/2)
  .attr("x2", center+width/2)
  .attr("y1", function(d){ return(y(d))} )
  .attr("y2", function(d){ return(y(d))} )
  .attr("stroke", "black")
</script>

"""
)
```

Popular Libraries

When it comes to data visualization, Python offers multiple graphing libraries that come packed with many different features. By default, every Apache Spark Pool in Azure Synapse Analytics contains a set of curated and popular open-source libraries. You can also add or manage additional libraries & versions by using the Azure Synapse Analytics library management capabilities.

Matplotlib

You can render standard plotting libraries, like Matplotlib, using the built-in rendering functions for each library.

The following image is an example of creating a bar chart using **Matplotlib**.

Run the following sample code to draw the image above.

```
# Bar chart

import matplotlib.pyplot as plt

x1 = [1, 3, 4, 5, 6, 7, 9]
y1 = [4, 7, 2, 4, 7, 8, 3]

x2 = [2, 4, 6, 8, 10]
y2 = [5, 6, 2, 6, 2]

plt.bar(x1, y1, label="Blue Bar", color='b')
plt.bar(x2, y2, label="Green Bar", color='g')
plt.plot()

plt.xlabel("bar number")
plt.ylabel("bar height")
plt.title("Bar Chart Example")
plt.legend()
plt.show()
```

Bokeh

You can render HTML or interactive libraries, like **bokeh**, using the `displayHTML(df)`.

The following image is an example of plotting glyphs over a map using **bokeh**.

Run the following sample code to draw the image above.

```
from bokeh.plotting import figure, output_file
from bokeh.tile_providers import get_provider, Vendors
from bokeh.embed import file_html
from bokeh.resources import CDN
from bokeh.models import ColumnDataSource

tile_provider = get_provider(Vendors.CARTODBPOSITRON)

# range bounds supplied in web mercator coordinates
p = figure(x_range=(-9000000,-8000000), y_range=(4000000,5000000),
           x_axis_type="mercator", y_axis_type="mercator")
p.add_tile(tile_provider)

# plot datapoints on the map
source = ColumnDataSource(
    data=dict(x=[ -8800000, -8500000 , -8800000],
              y=[4200000, 4500000, 4900000])
)

p.circle(x="x", y="y", size=15, fill_color="blue", fill_alpha=0.8, source=source)

# create an html document that embeds the Bokeh plot
html = file_html(p, CDN, "my plot1")

# display this html
displayHTML(html)
```

Plotly

You can render HTML or interactive libraries like **Plotly**, using the **displayHTML()**.

Run the following sample code to draw the image below.

```
from urllib.request import urlopen
import json
with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-counties-fips.json') as
response:
    counties = json.load(response)

import pandas as pd
df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/fips-unemp-16.csv",
                   dtype={"fips": str})

import plotly
import plotly.express as px

fig = px.choropleth(df, geojson=counties, locations='fips', color='unemp',
                           color_continuous_scale="Viridis",
                           range_color=(0, 12),
                           scope="usa",
                           labels={'unemp':'unemployment rate'}
                          )
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})

# create an html document that embeds the Plotly plot
h = plotly.offline.plot(fig, output_type='div')

# display this html
displayHTML(h)
```

Pandas

You can view html output of pandas dataframe as the default output, notebook will automatically show the styled html content.

```
import pandas as pd
import numpy as np

df = pd.DataFrame([[38.0, 2.0, 18.0, 22.0, 21, np.nan],[19, 439, 6, 452, 226,232]],

                  index=pd.Index(['Tumour (Positive)', 'Non-Tumour (Negative)'], name='Actual Label:'),

                  columns=pd.MultiIndex.from_product([['Decision Tree', 'Regression', 'Random'],['Tumour',
'Non-Tumour']], names=['Model:', 'Predicted:']))

df
```

Additional libraries

Beyond these libraries, the Azure Synapse Analytics Runtime also includes the following set of libraries that are often used for data visualization:

- Seaborn

You can visit the Azure Synapse Analytics Runtime documentation for the most up to date information about the available libraries and versions.

Connect to Power BI using Apache Spark & SQL On-Demand

Azure Synapse Analytics integrates deeply with Power BI allowing data engineers to build analytics solutions.

Azure Synapse Analytics allows the different workspace computational engines to share databases and tables between its Spark pools and serverless SQL pool. Using the shared metadata model,you can query your Apache

Spark tables using SQL on-demand. Once done, you can connect your SQL on-demand endpoint to Power BI to easily query your synced Spark tables.

Next steps

- For more information on how to set up the Spark SQL DW Connector: Synapse SQL connector
- View the default libraries: Azure Synapse Analytics runtime

Tutorial: Create an Apache Spark application with IntelliJ using a Synapse workspace

8/19/2022 • 7 minutes to read • Edit Online

This tutorial shows you how to use the Azure Toolkit for IntelliJ plug-in to develop Apache Spark applications, which are written in Scala, and then submit them to a serverless Apache Spark pool directly from the IntelliJ integrated development environment (IDE). You can use the plug-in in a few ways:

- Develop and submit a Scala Spark application on a Spark pool.
- Access your Spark pools resources.
- Develop and run a Scala Spark application locally.

In this tutorial, you learn how to:

- Use the Azure Toolkit for IntelliJ plug-in
- Develop Apache Spark applications
- Submit application to Spark pools

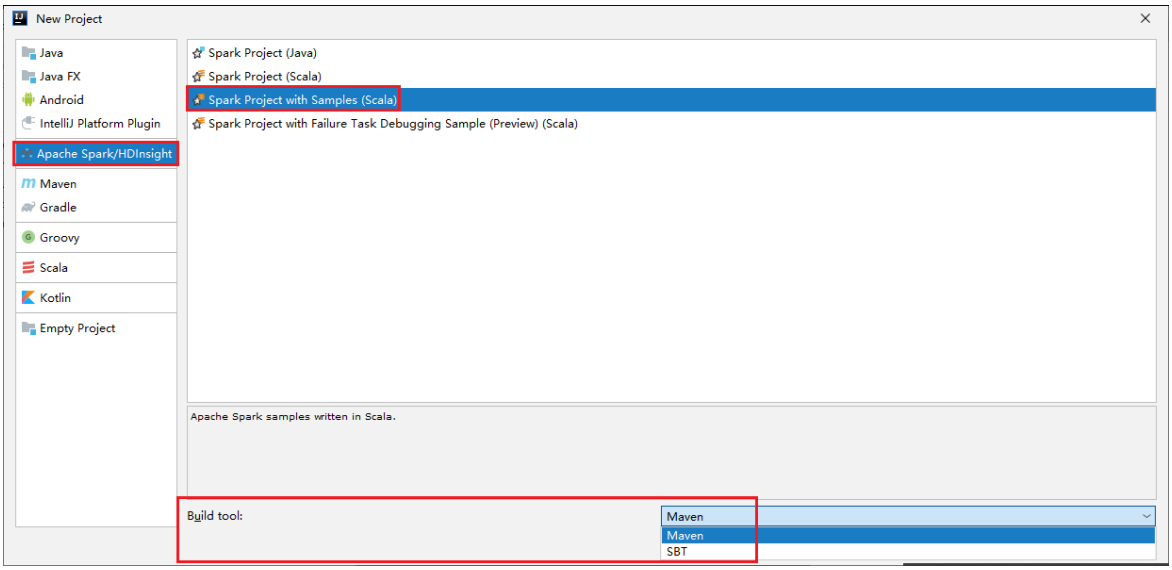
Prerequisites

- IntelliJ IDEA Community Version.
- Azure toolkit plugin 3.27.0-2019.2 – Install from IntelliJ Plugin repository
- JDK (Version 1.8).
- Scala Plugin – Install from IntelliJ Plugin repository.
- The following prerequisite is only for Windows users:

  While you're running the local Spark Scala application on a Windows computer, you might get an exception, as explained in SPARK-2356. The exception occurs because WinUtils.exe is missing on Windows. To resolve this error, download the WinUtils executable to a location such as **C:\WinUtils\bin**. Then, add the environment variable **HADOOP_HOME**, and set the value of the variable to **C:\WinUtils**.

Create a Spark Scala application for a Spark pool

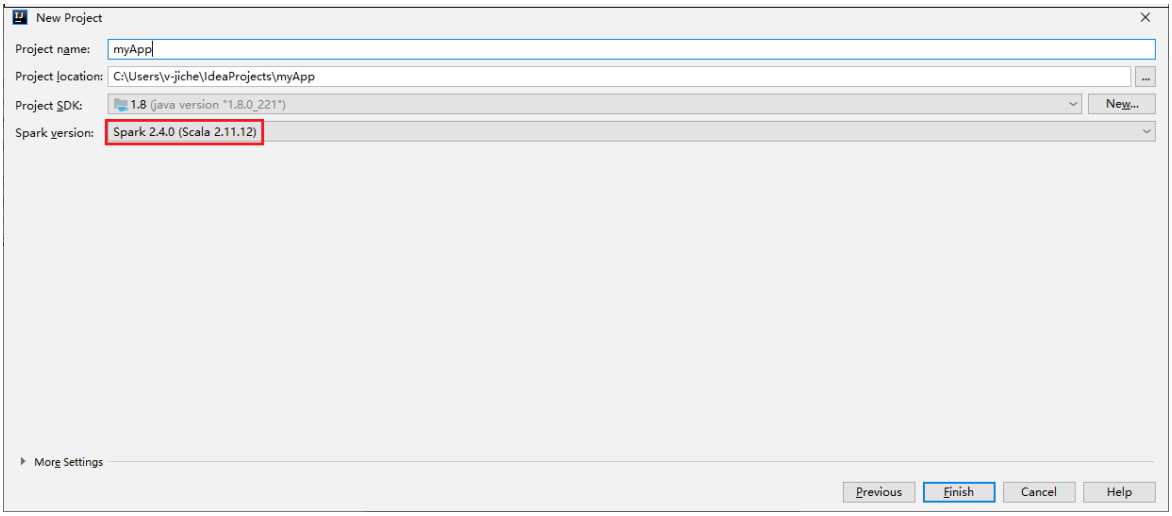
1. Start IntelliJ IDEA, and select **Create New Project** to open the **New Project** window.
2. Select **Apache Spark/HDInsight** from the left pane.
3. Select **Spark Project with Samples(Scala)** from the main window.
4. From the **Build tool** drop-down list, select one of the following types:
   - **Maven** for Scala project-creation wizard support.
   - **SBT** for managing the dependencies and building for the Scala project.



5. Select **Next**.

6. In the **New Project** window, provide the following information:

PROPERTY	DESCRIPTION
Project name	Enter a name. This tutorial uses `myApp`.
Project location	Enter the wanted location to save your project.
Project SDK	It might be blank on your first use of IDEA. Select **New...** and navigate to your JDK.
Spark Version	The creation wizard integrates the proper version for Spark SDK and Scala SDK. Synapse only supports **Spark 2.4.0**.



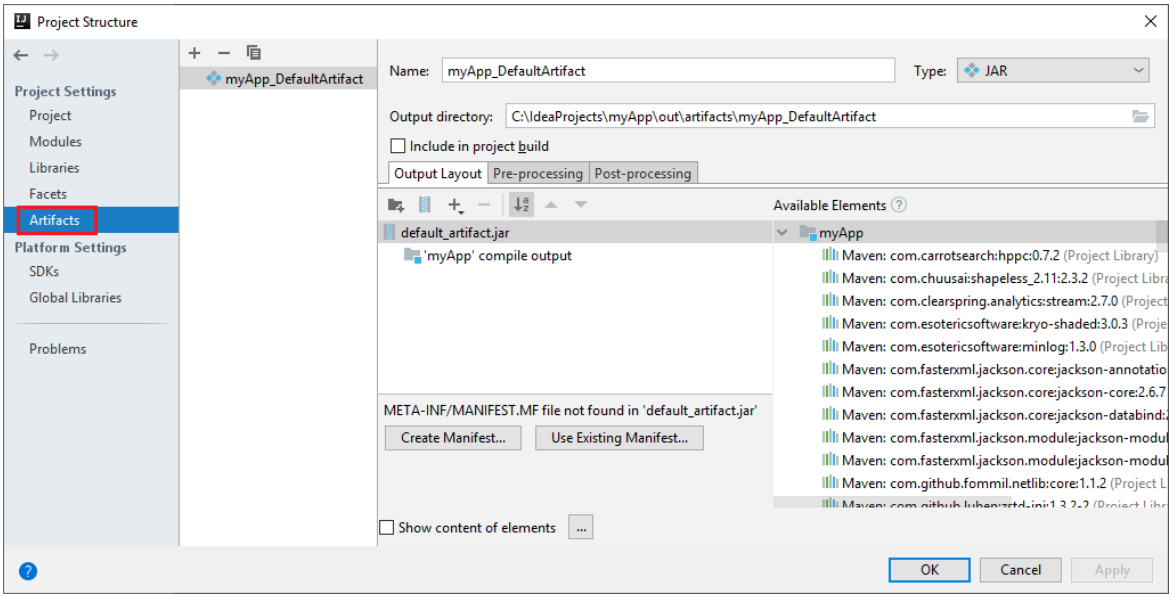
7. Select **Finish**. It may take a few minutes before the project becomes available.

8. The Spark project automatically creates an artifact for you. To view the artifact, do the following operating:

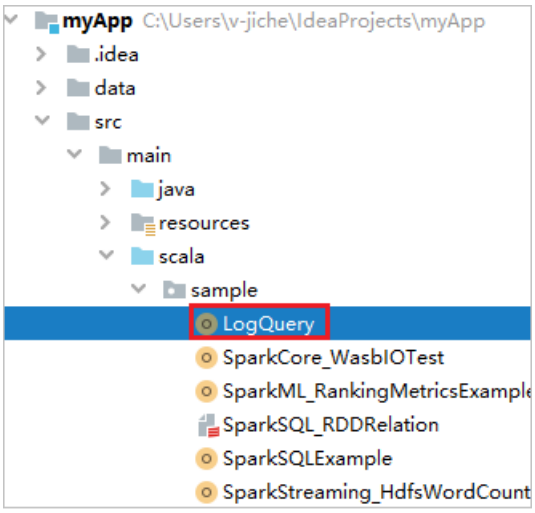
   a. From the menu bar, navigate to **File** > **Project Structure...**.

   b. From the **Project Structure** window, select **Artifacts**.

c. Select **Cancel** after viewing the artifact.



9. Find **LogQuery** from **myApp** > **src** > **main** > **scala**> **sample**> **LogQuery**. This tutorial uses **LogQuery** to run.

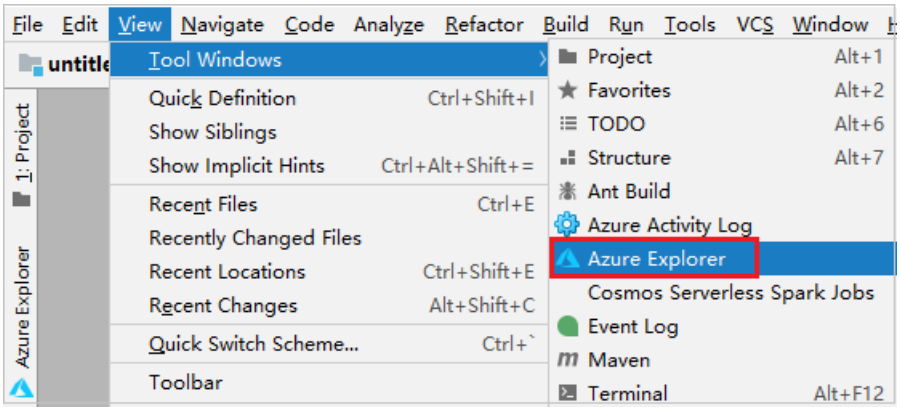


Connect to your Spark pools

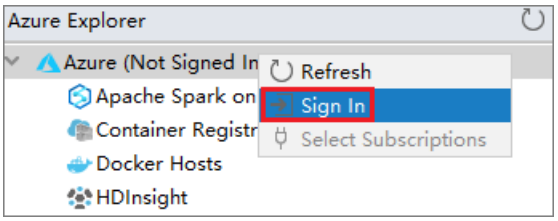
Sign in to Azure subscription to connect to your Spark pools.

Sign in to your Azure subscription

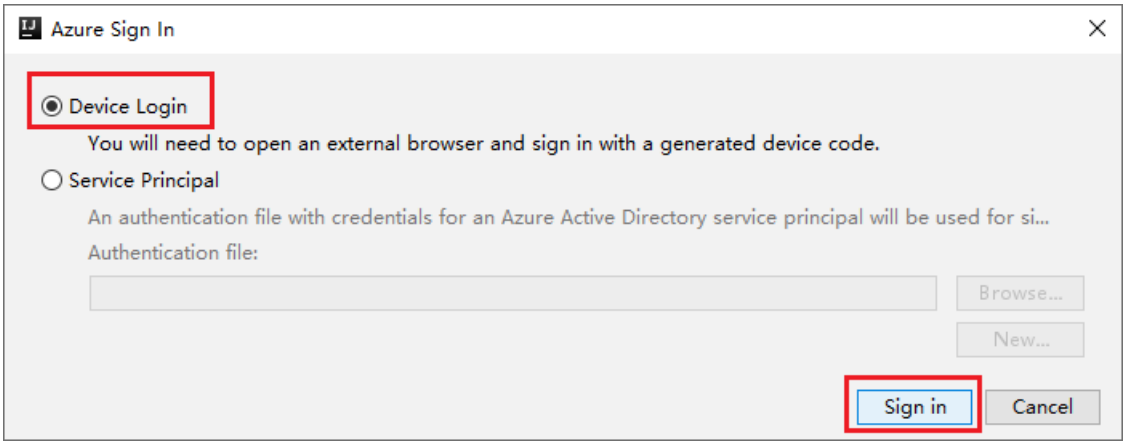
1. From the menu bar, navigate to **View** > **Tool Windows** > **Azure Explorer**.



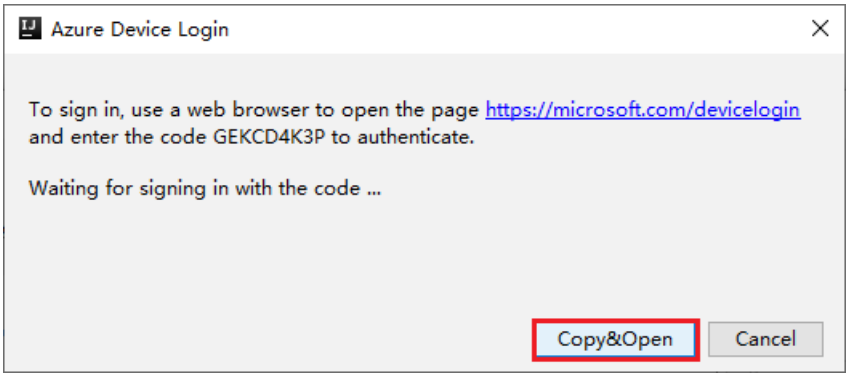
2. From Azure Explorer, right-click the **Azure** node, and then select **Sign In**.



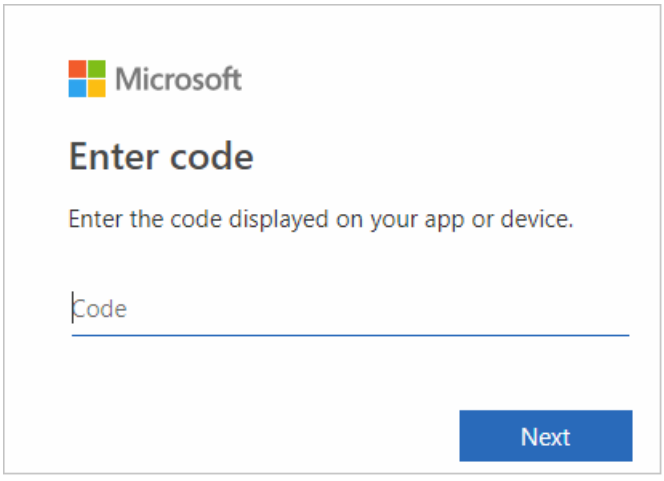
3. In the **Azure Sign In** dialog box, choose **Device Login**, and then select **Sign in**.



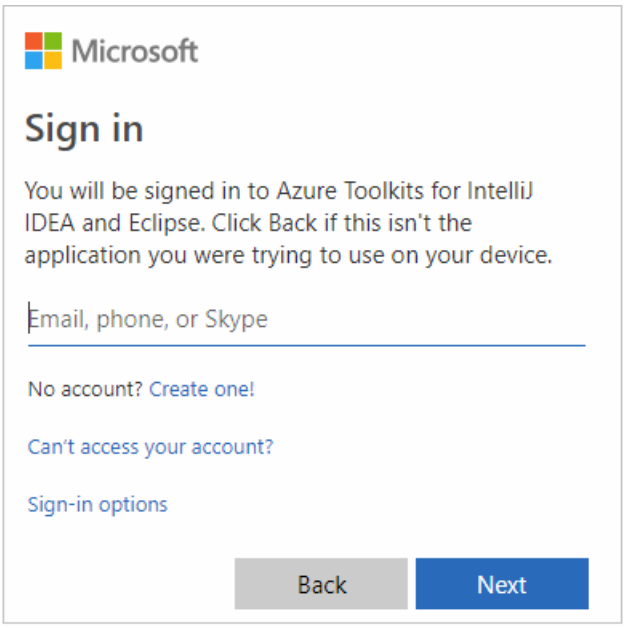
4. In the **Azure Device Login** dialog box, select **Copy&Open**.



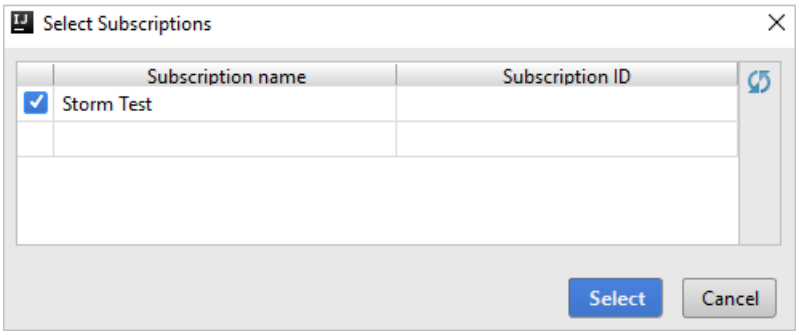
5. In the browser interface, paste the code, and then select **Next**.



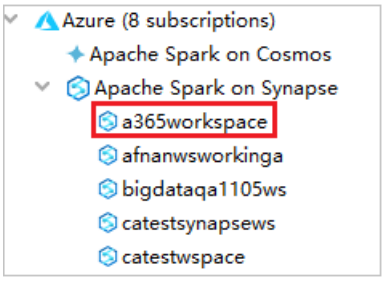
6. Enter your Azure credentials, and then close the browser.



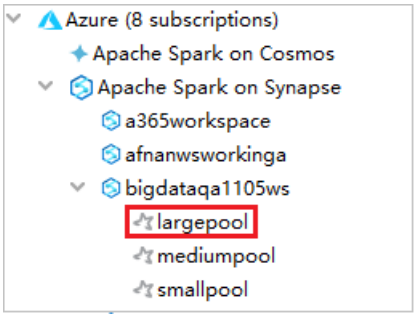
7. After you're signed in, the **Select Subscriptions** dialog box lists all the Azure subscriptions that are associated with the credentials. Select your subscription and then select **Select**.



8. From **Azure Explorer**, expand **Apache Spark on Synapse** to view the Workspaces that are in your subscriptions.



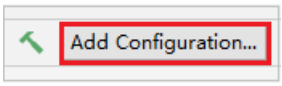
9. To view the Spark pools, you can further expand a workspace.



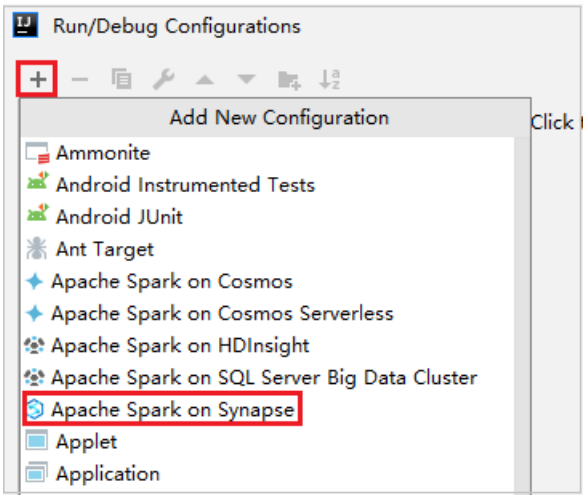
Remote Run a Spark Scala application on a Spark pool

After creating a Scala application, you can remotely run it.

1. Open **Run/Debug Configurations** window by selecting the icon.



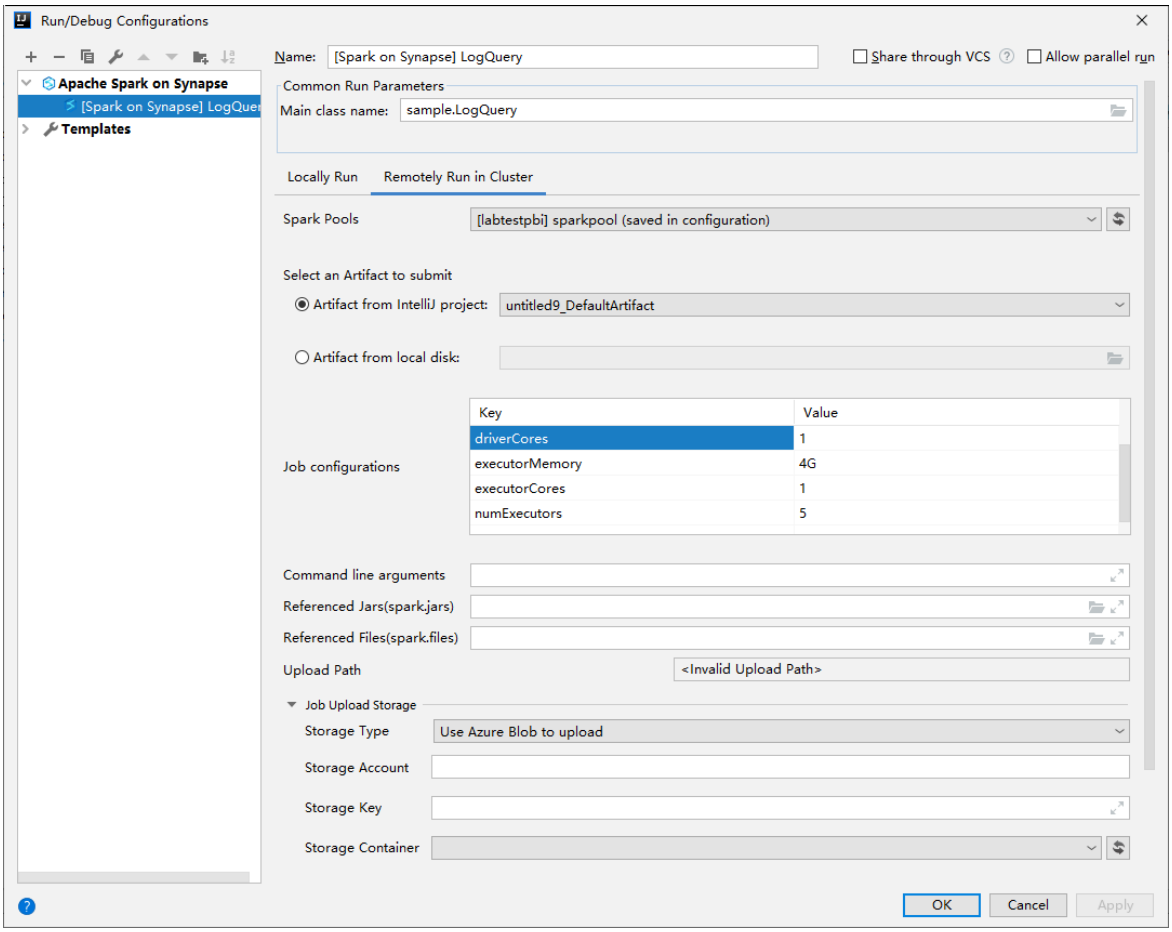
2. In the **Run/Debug Configurations** dialog window, select **+**, then select **Apache Spark on Synapse**.



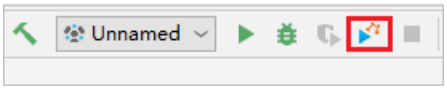
3. In the **Run/Debug Configurations** window, provide the following values, and then select **OK**:

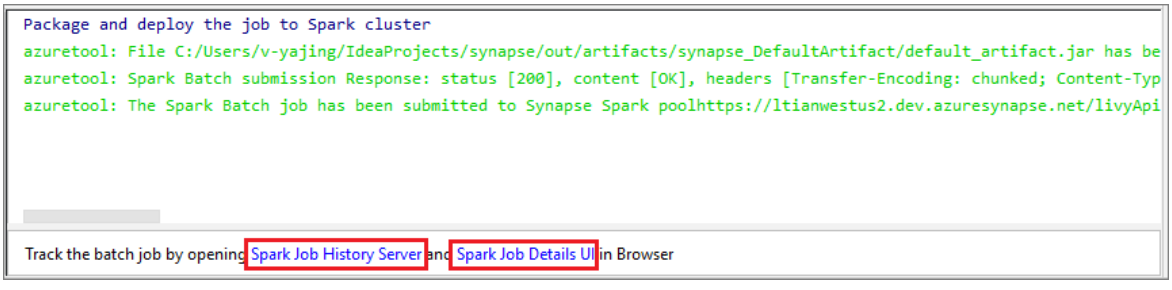
PROPERTY	VALUE
Spark pools	Select the Spark pools on which you want to run your application.
Select an Artifact to submit	Leave default setting.
Main class name	The default value is the main class from the selected file. You can change the class by selecting the ellipsis(...) and choosing another class.
Job configurations	You can change the default key and values. For more information, see Apache Livy REST API.
Command-line arguments	You can enter arguments separated by space for the main class if needed.
Referenced Jars and Referenced Files	You can enter the paths for the referenced Jars and files if any. You can also browse files in the Azure virtual file system, which currently only supports ADLS Gen2 cluster. For more information: Apache Spark Configuration and How to upload resources to cluster.
Job Upload Storage	Expand to reveal additional options.
Storage Type	Select **Use Azure Blob to upload** or **Use cluster default storage account to upload** from the drop-down list.
Storage Account	Enter your storage account.
Storage Key	Enter your storage key.

PROPERTY	VALUE
Storage Container	Select your storage container from the drop-down list once **Storage Account** and **Storage Key** has been entered.



4. Select **SparkJobRun** icon to submit your project to the selected Spark pool. The **Remote Spark Job in Cluster** tab displays the job execution progress at the bottom. You can stop the application by selecting the red button.



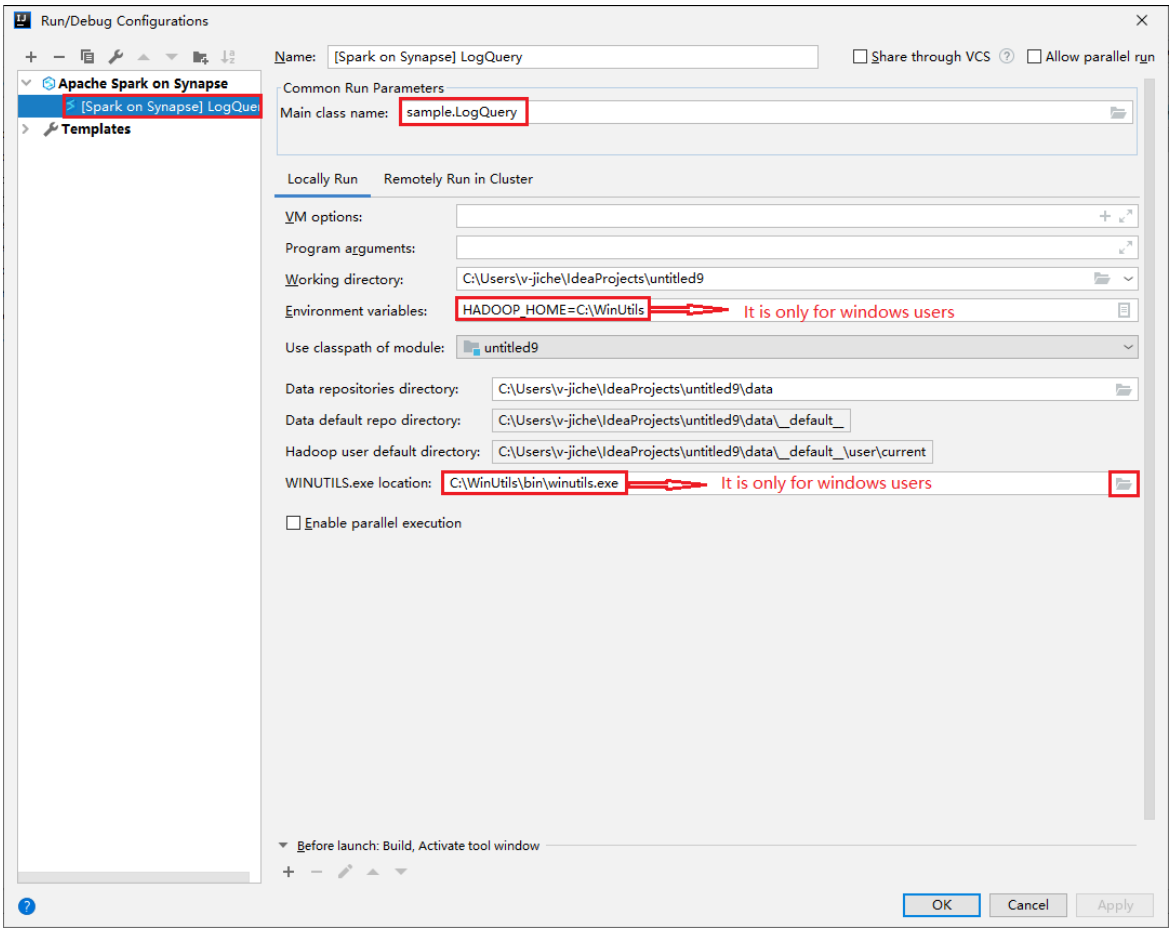


Local Run/Debug Apache Spark applications

You can follow the instructions below to set up your local run and local debug for your Apache Spark job.

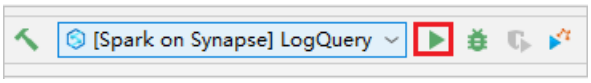
Scenario 1: Do local run

1. Open the **Run/Debug Configurations** dialog, select the plus sign (+). Then select the **Apache Spark on Synapse** option. Enter information for **Name**, **Main class name** to save.

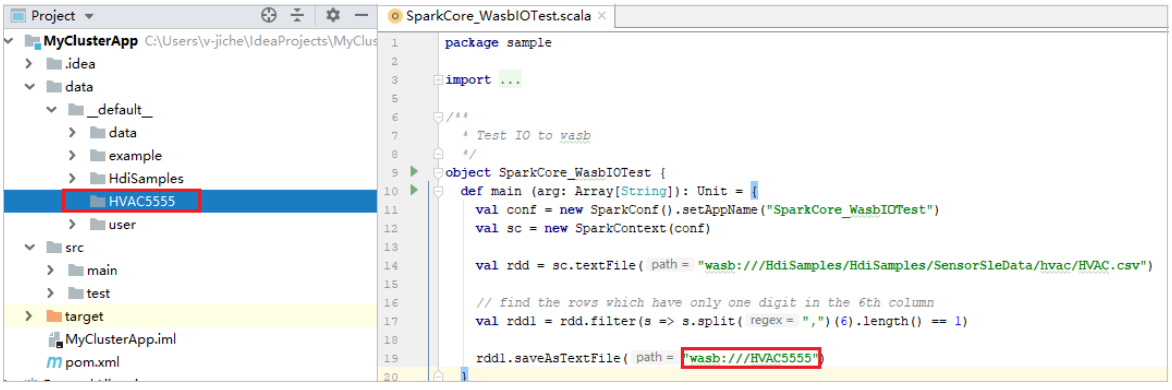


- Environment variables and WinUtils.exe Location are only for windows users.
- Environment variables: The system environment variable can be auto detected if you have set it before and no need to manually add.
- WinUtils.exe Location: You can specify the WinUtils location by selecting the folder icon on the right.

2. Then select the local play button.



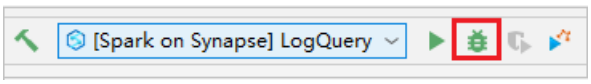
3. Once local run completed, if script includes output, you can check the output file from **data** > **default**.



Scenario 2: Do local debugging

1. Open the **LogQuery** script, set breakpoints.

2. Select **Local debug** icon to do local debugging.

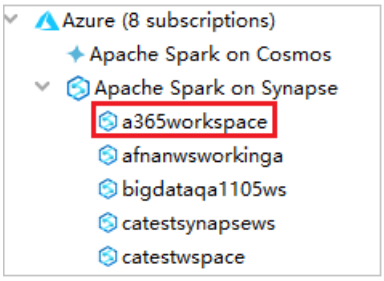


Access and manage Synapse Workspace

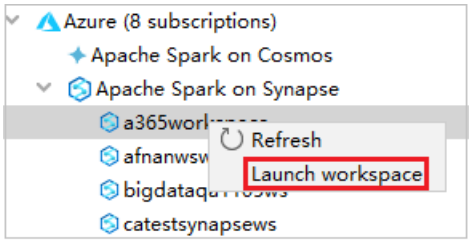
You can perform different operations in Azure Explorer within Azure Toolkit for IntelliJ. From the menu bar, navigate to **View** > **Tool Windows** > **Azure Explorer**.

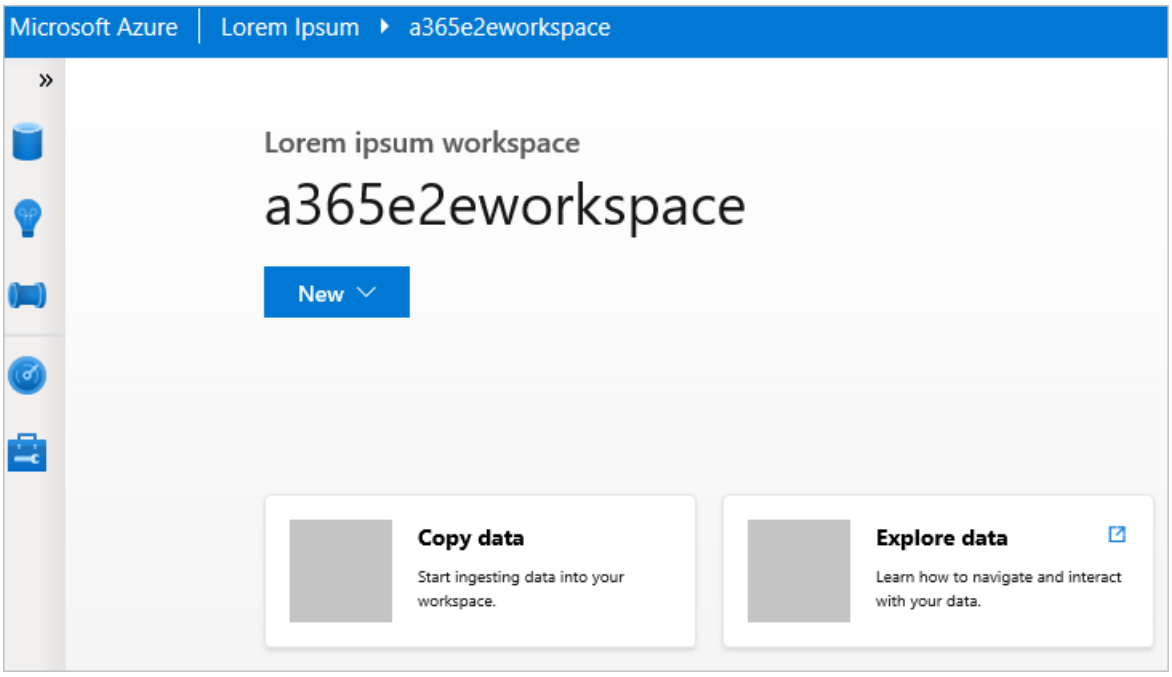
Launch workspace

1. From Azure Explorer, navigate to **Apache Spark on Synapse**, then expand it.



2. Right-click a workspace, then select **Launch workspace**, website will be opened.





Spark console

You can run Spark Local Console(Scala) or run Spark Livy Interactive Session Console(Scala).

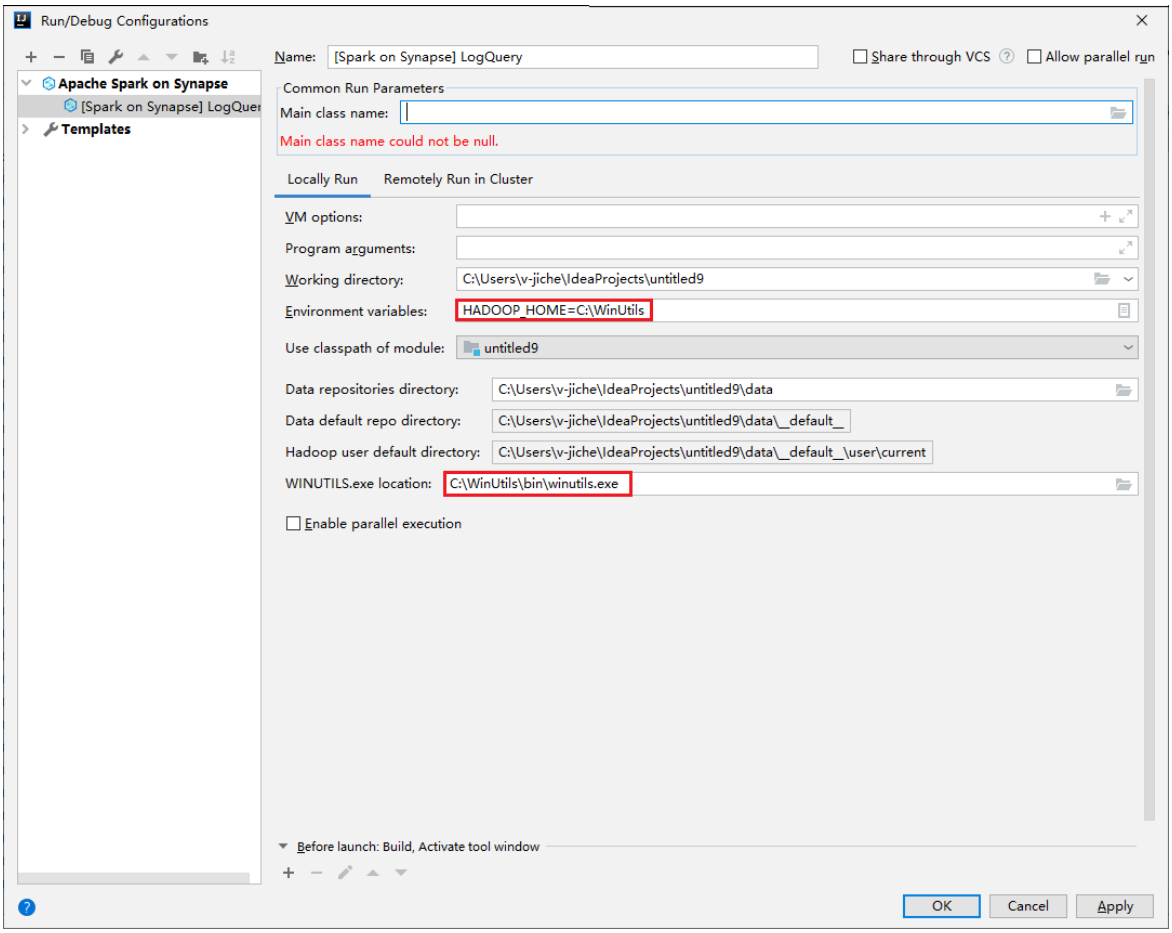
Spark local console (Scala)

Ensure you've satisfied the WINUTILS.EXE prerequisite.

1. From the menu bar, navigate to **Run** > **Edit Configurations...**.
2. From the **Run/Debug Configurations** window, in the left pane, navigate to **Apache Spark on Synapse** > **[Spark on Synapse] myApp**.
3. From the main window, select the **Locally Run** tab.

4. Provide the following values, and then select **OK**:

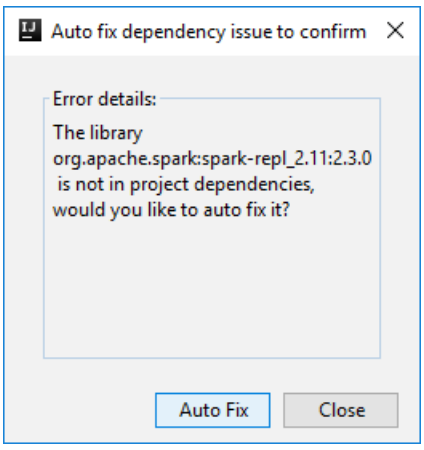
PROPERTY	VALUE
Environment variables	Ensure the value for HADOOP_HOME is correct.
WINUTILS.exe location	Ensure the path is correct.

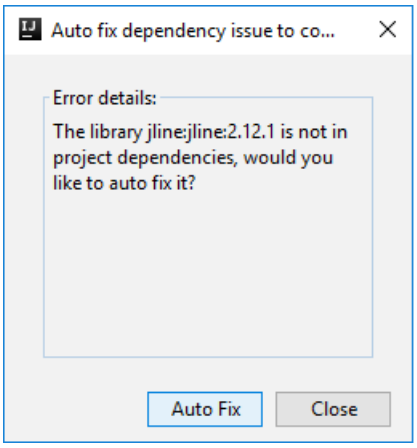


5. From Project, navigate to **myApp** > **src** > **main** > **scala** > **myApp**.

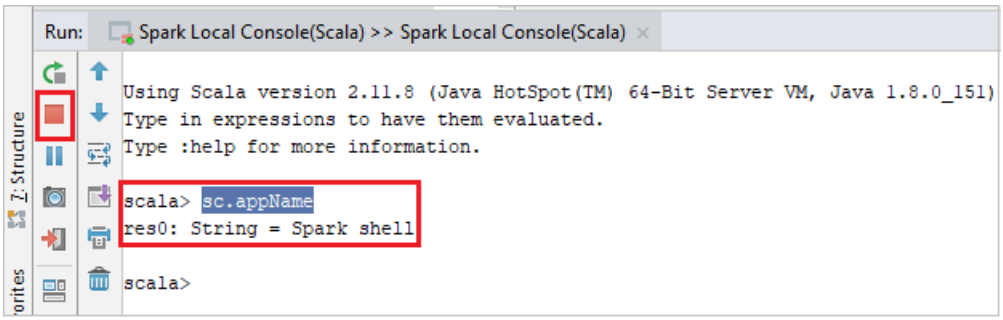
6. From the menu bar, navigate to **Tools** > **Spark console** > **Run Spark Local Console(Scala)**.

7. Then two dialogs may be displayed to ask you if you want to auto fix dependencies. If so, select **Auto Fix**.





8. The console should look similar to the picture below. In the console window type `sc.appName`, and then press ctrl+Enter. The result will be shown. You can stop the local console by selecting red button.

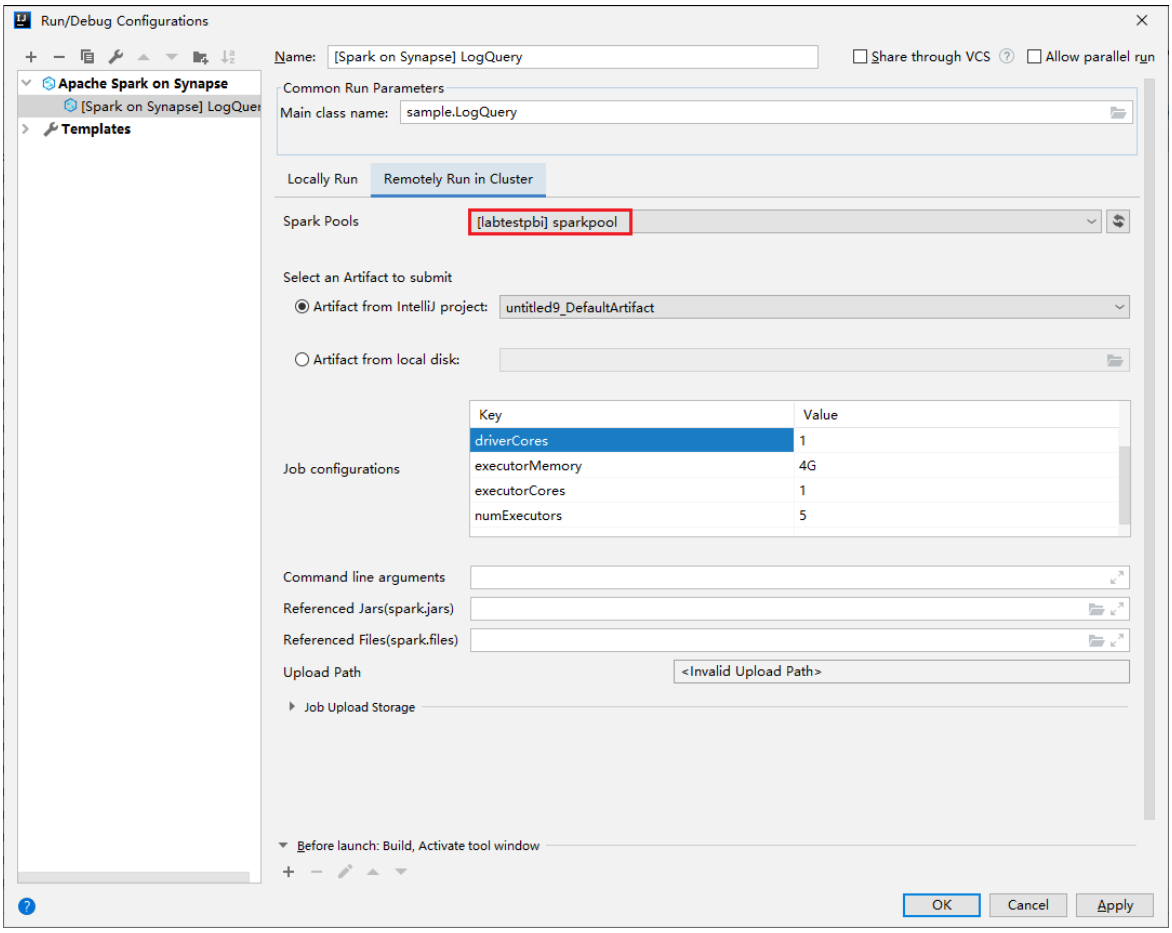


Spark Livy interactive session console (Scala)

It's only supported on IntelliJ 2018.2 and 2018.3.

1. From the menu bar, navigate to **Run** > **Edit Configurations...**.
2. From the **Run/Debug Configurations** window, in the left pane, navigate to **Apache Spark on synapse** > **[Spark on synapse] myApp**.
3. From the main window, select the **Remotely Run in Cluster** tab.
4. Provide the following values, and then select **OK**:

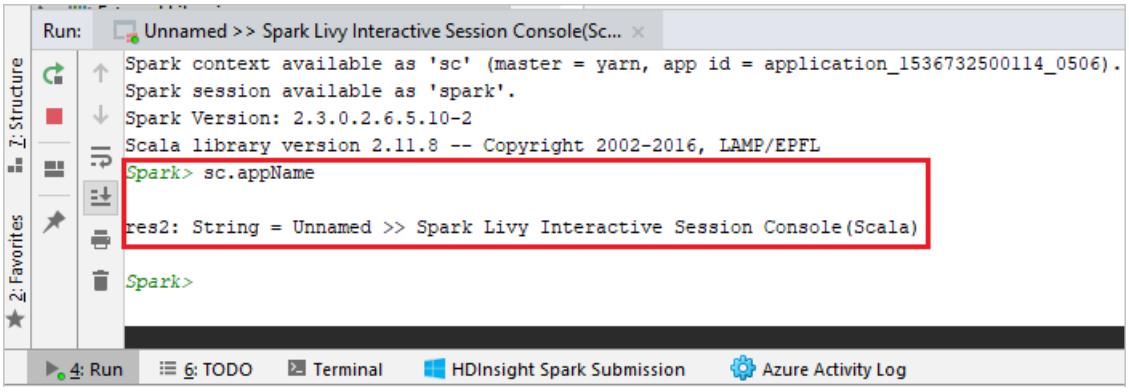
PROPERTY	VALUE
Main class name	Select the Main class name.
Spark pools	Select the Spark pools on which you want to run your application.



5. From Project, navigate to **myApp** > **src** > **main** > **scala** > **myApp**.

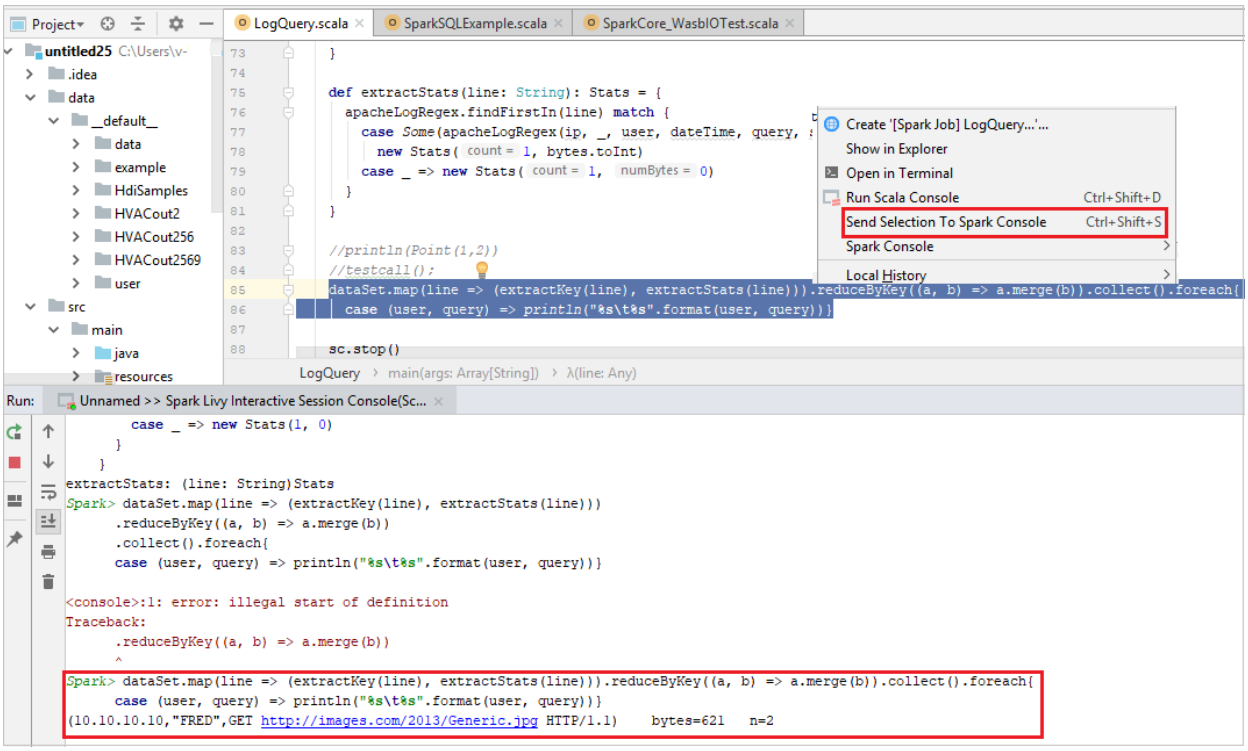
6. From the menu bar, navigate to **Tools** > **Spark console** > **Run Spark Livy Interactive Session Console(Scala)**.

7. The console should look similar to the picture below. In the console window type `sc.appName`, and then press ctrl+Enter. The result will be shown. You can stop the local console by selecting red button.



Send selection to Spark console

You may want to see the script result by sending some code to the local console or Livy Interactive Session Console(Scala). To do so, you can highlight some code in the Scala file, then right-click **Send Selection To Spark console**. The selected code will be sent to the console and be done. The result will be displayed after the code in the console. The console will check the existing errors.



Next steps

- Azure Synapse Analytics
- Create a new Apache Spark pool for an Azure Synapse Analytics workspace

Tutorial: Create an Apache Spark application with VSCode using a Synapse workspace

8/19/2022 • 7 minutes to read • Edit Online

Learn how to use Apache Spark & Hive Tools for Visual Studio Code. Use the tools to create and submit Apache Hive batch jobs, interactive Hive queries, and PySpark scripts for Apache Spark. First we'll describe how to install Spark & Hive Tools in Visual Studio Code. Then we'll walk through how to submit jobs to Spark & Hive Tools.

Spark & Hive Tools can be installed on platforms that are supported by Visual Studio Code. Note the following prerequisites for different platforms.

Prerequisites

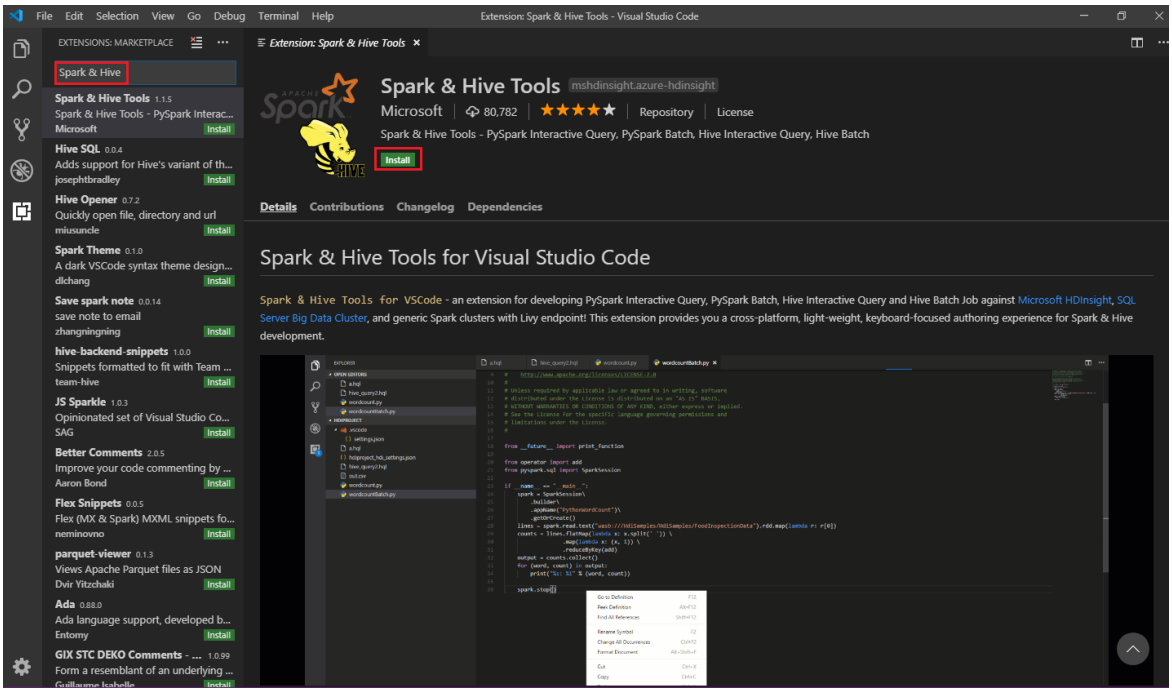
The following items are required for completing the steps in this article:

- A serverless Apache Spark pool. To create a serverless Apache Spark pool, see Create Apache Spark pool using Azure portal.
- Visual Studio Code.
- Mono. Mono is required only for Linux and macOS.
- A PySpark interactive environment for Visual Studio Code.
- A local directory. This article uses **C:\HD\Synaseexample**.

Install Spark & Hive Tools

After you meet the prerequisites, you can install Spark & Hive Tools for Visual Studio Code by following these steps:

1. Open Visual Studio Code.
2. From the menu bar, navigate to **View** > **Extensions**.
3. In the search box, enter **Spark & Hive**.
4. Select **Spark & Hive Tools** from the search results, and then select **Install**:



5. Select **Reload** when necessary.

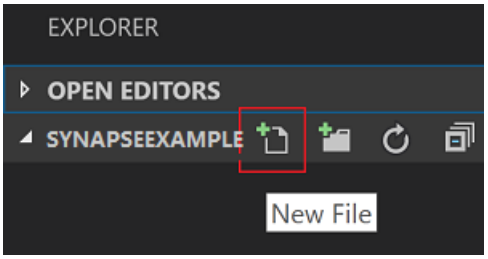
> **NOTE**
>
> **Synapse PySpark installation error** is an known issue.

Open a work folder

To open a work folder and to create a file in Visual Studio Code, follow these steps:

1. From the menu bar, navigate to **File** > **Open Folder...** > **C:\HD\Synaseexample**, and then select the **Select Folder** button. The folder appears in the **Explorer** view on the left.

2. In **Explorer** view, select the **Synaseexample** folder, and then select the **New File** icon next to the work folder:



3. Name the new file by using the `.py` (Spark script) file extension. This example uses **HelloWorld.py**.

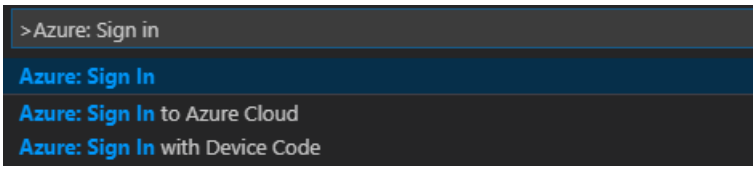
Connect to your Spark pools

Sign in to Azure subscription to connect to your Spark pools.

Sign in to your Azure subscription

Follow these steps to connect to Azure:

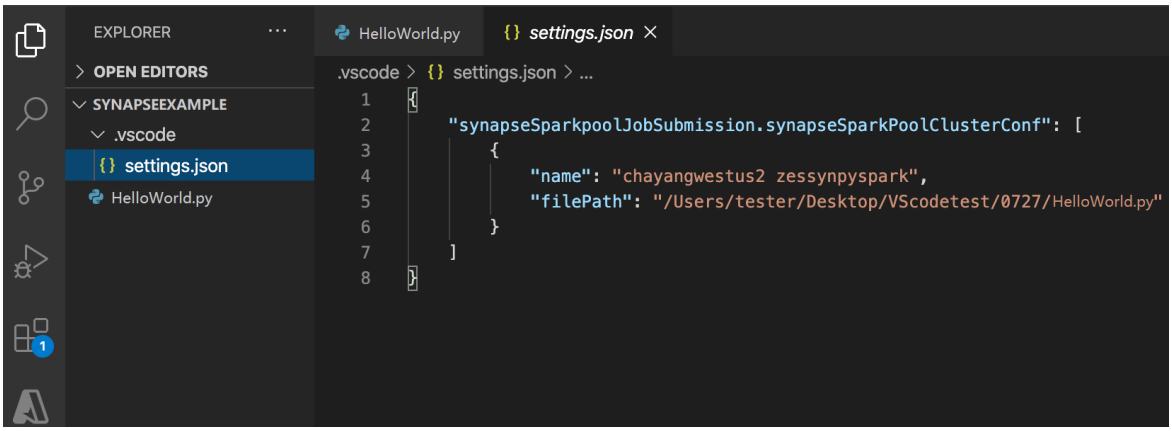
1. From the menu bar, navigate to **View** > **Command Palette...**, and enter **Azure: Sign In**:



2. Follow the sign-in instructions to sign in to Azure. After you're connected, your Azure account name shows on the status bar at the bottom of the Visual Studio Code window.

Set the default Spark pool

1. Reopen the **Synaseexample** folder that was discussed earlier, if closed.
2. Select the **HelloWorld.py** file that was created earlier. It opens in the script editor.
3. Right-click the script editor, and then select **Synapse: Set default Spark pool**.
4. Connect to your Azure account if you haven't yet done so.
5. Select a Spark pool as the default Spark pool for the current script file.
6. Use **Synapse: PySpark Interactive** to submit this file. And the tools automatically update the **.VSCode\settings.json** configuration file:



Submit interactive Synapse PySpark queries to Spark pool (Not supported anymore)

> **NOTE**
> For Synapse Pyspark interactive, since its dependency will not be maintained anymore by other team, this will not be maintained anymore as well. If you trying to use Synapse Pyspark interactive, please switch to use Azure Synapse Analytics instead. And it's a long term change.

Users can perform Synapse PySpark interactive on Spark pool in the following ways:

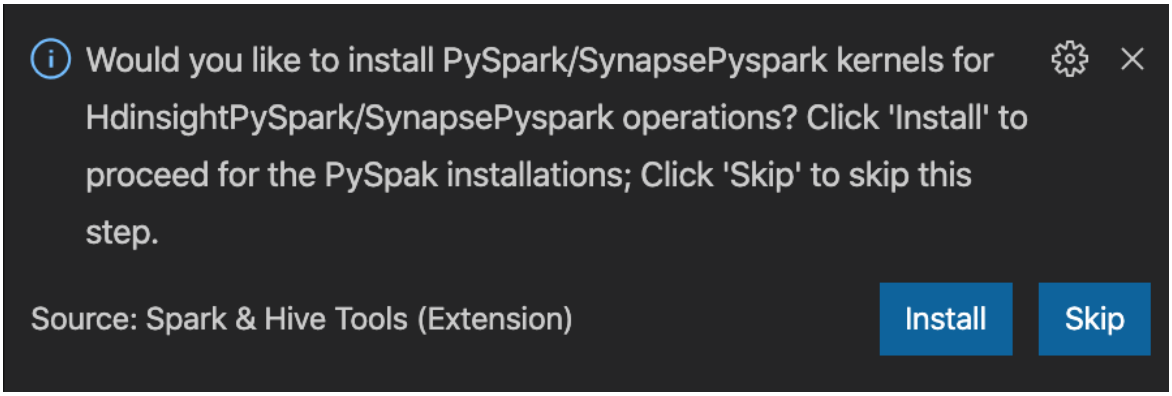
Using the Synapse PySpark interactive command in PY file

Using the PySpark interactive command to submit the queries, follow these steps:

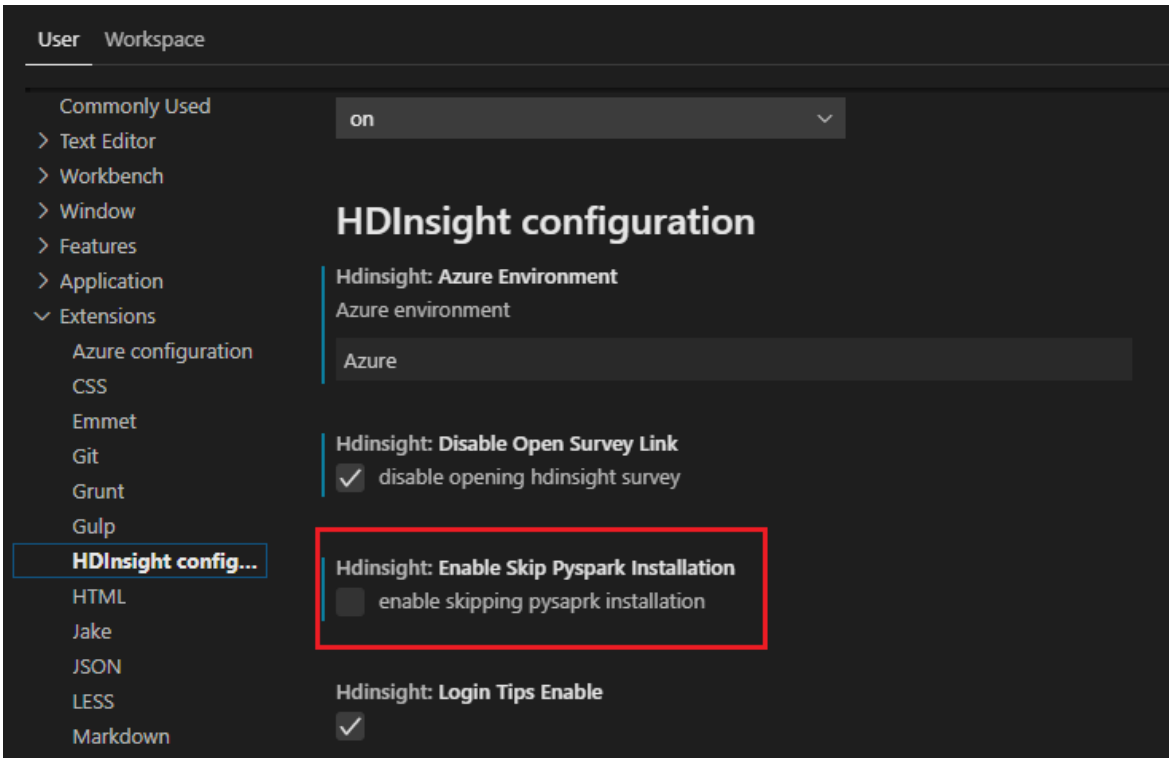
1. Reopen the **Synaseexample** folder that was discussed earlier, if closed.
2. Create a new **HelloWorld.py** file, following the earlier steps.
3. Copy and paste the following code into the script file:

```
import sys
from operator import add
from pyspark.sql import SparkSession, Row

spark = SparkSession\
 .builder\
 .appName("PythonWordCount")\
 .getOrCreate()

data = [Row(col1='pyspark and spark', col2=1), Row(col1='pyspark', col2=2), Row(col1='spark vs hadoop',
col2=2), Row(col1='spark', col2=2), Row(col1='hadoop', col2=2)]
df = spark.createDataFrame(data)
lines = df.rdd.map(lambda r: r[0])

counters = lines.flatMap(lambda x: x.split(' ')) \
 .map(lambda x: (x, 1)) \
 .reduceByKey(add)

output = counters.collect()
sortedCollection = sorted(output, key = lambda r: r[1], reverse = True)

for (word, count) in sortedCollection:
 print("%s: %i" % (word, count))
```

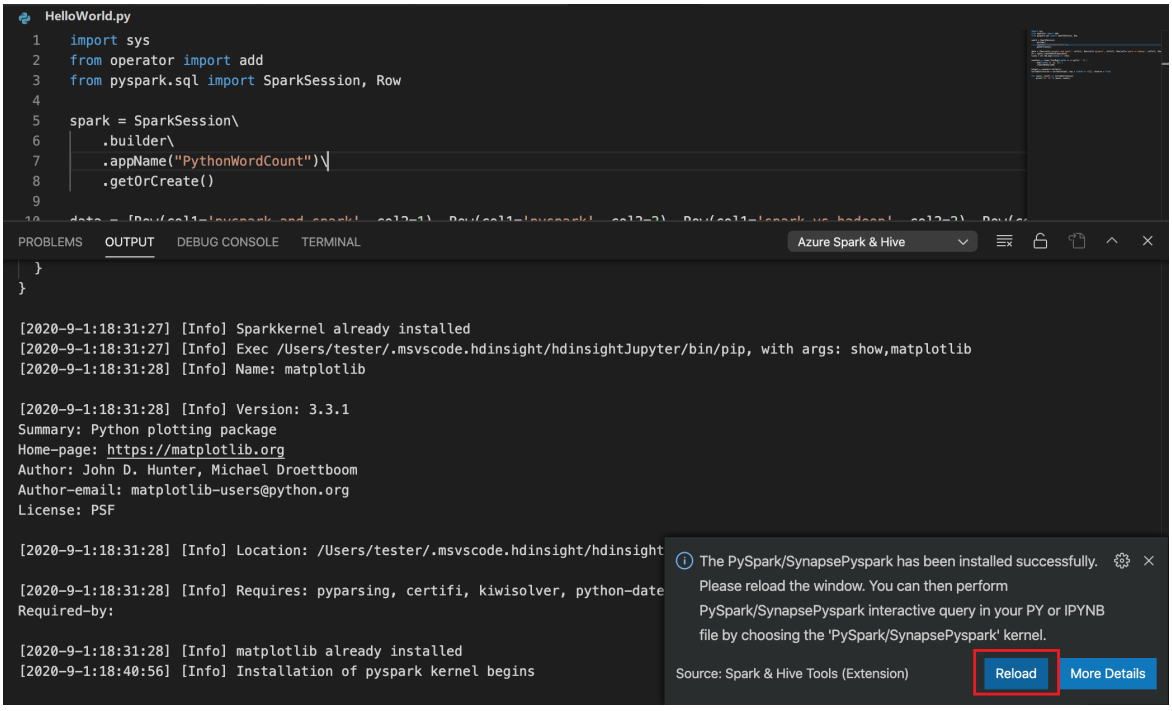
4. The prompt to install PySpark/Synapse Pyspark kernel is displayed in the lower right corner of the window. You can click on **Install** button to proceed for the PySpark/Synapse Pyspark installations; or click on **Skip** button to skip this step.



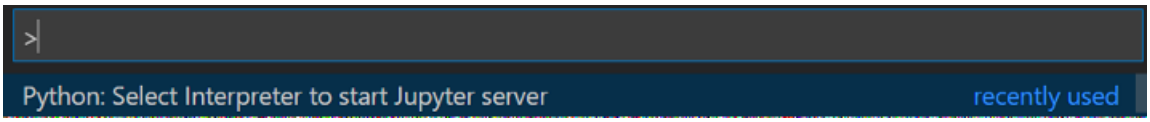
5. If you need to install it later, you can navigate to **File** > **Preference** > **Settings**, then uncheck **Hdinsight: Enable Skip Pyspark Installation** in the settings.



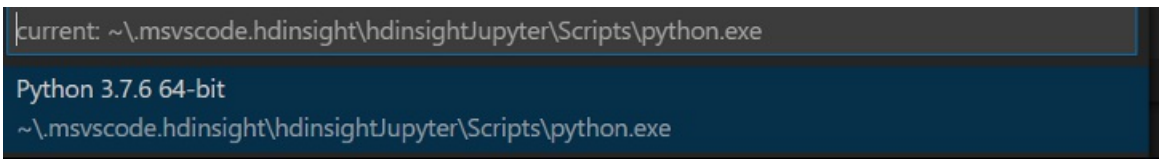
6. If the installation is successful in step 4, the "PySpark/Synapse Pyspark installed successfully" message box is displayed in the lower right corner of the window. Click on **Reload** button to reload the window.



7. From the menu bar, navigate to **View** > **Command Palette...** or use the **Shift + Ctrl + P** keyboard shortcut, and enter **Python: Select Interpreter to start Jupyter Server**.



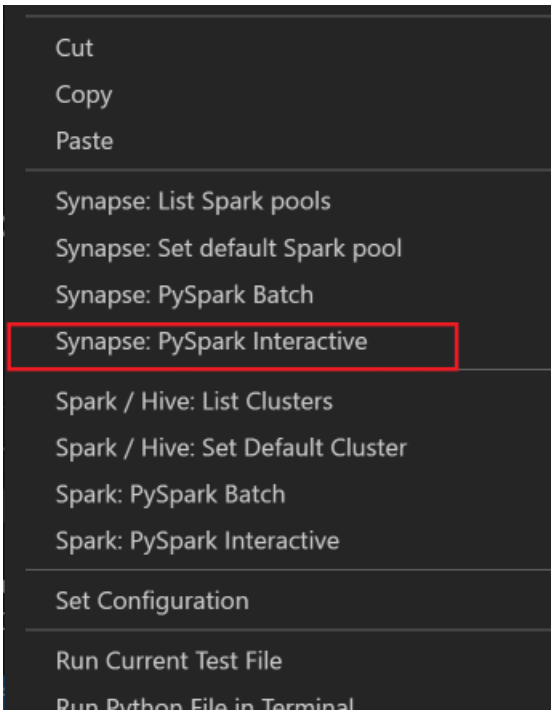
8. Select the Python option below.



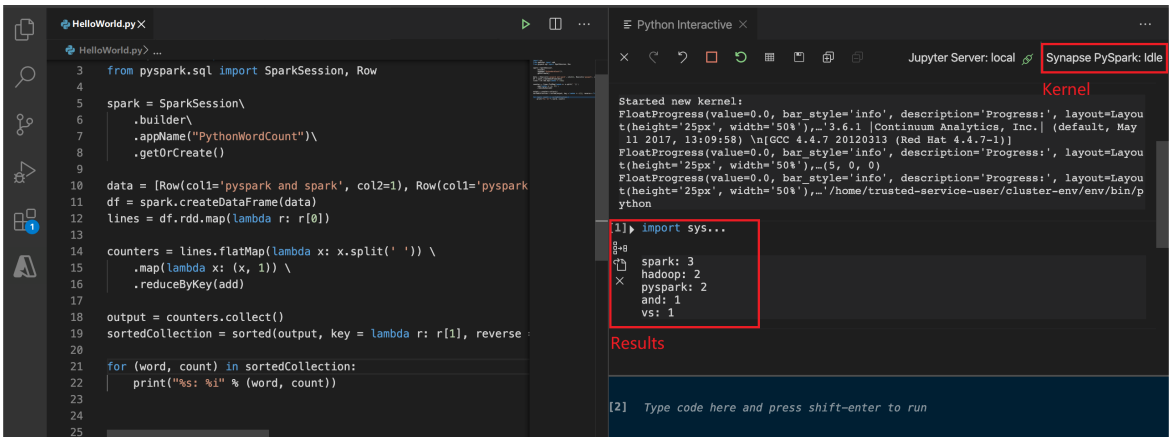
9. From the menu bar, navigate to **View** > **Command Palette...** or use the **Shift + Ctrl + P** keyboard shortcut, and enter **Developer: Reload Window**.

10. Connect to your Azure account if you haven't yet done so.

11. Select all the code, right-click the script editor, and select **Synapse: Pyspark Interactive** to submit the query.

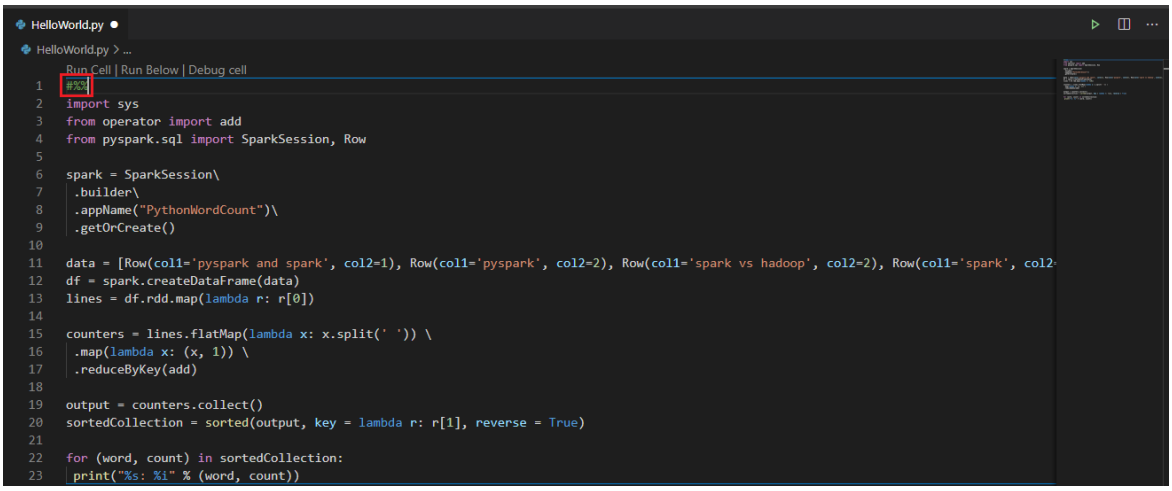


12. Select the Spark pool, if you haven't specified a default Spark pool. After a few moments, the **Python Interactive** results appear in a new tab. Click on PySpark to switch the kernel to **Synapse PySpark**, then, submit the selected code again, and the code will run successfully. The tools also let you submit a block of code instead of the whole script file by using the context menu:

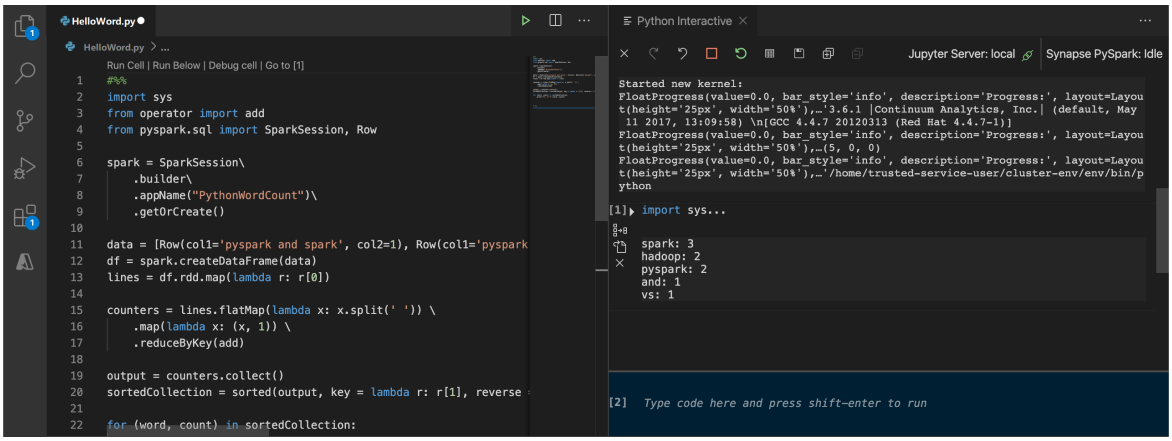


Perform interactive query in PY file using a #%% comment

1. Add **#%%** before the code to get notebook experience.



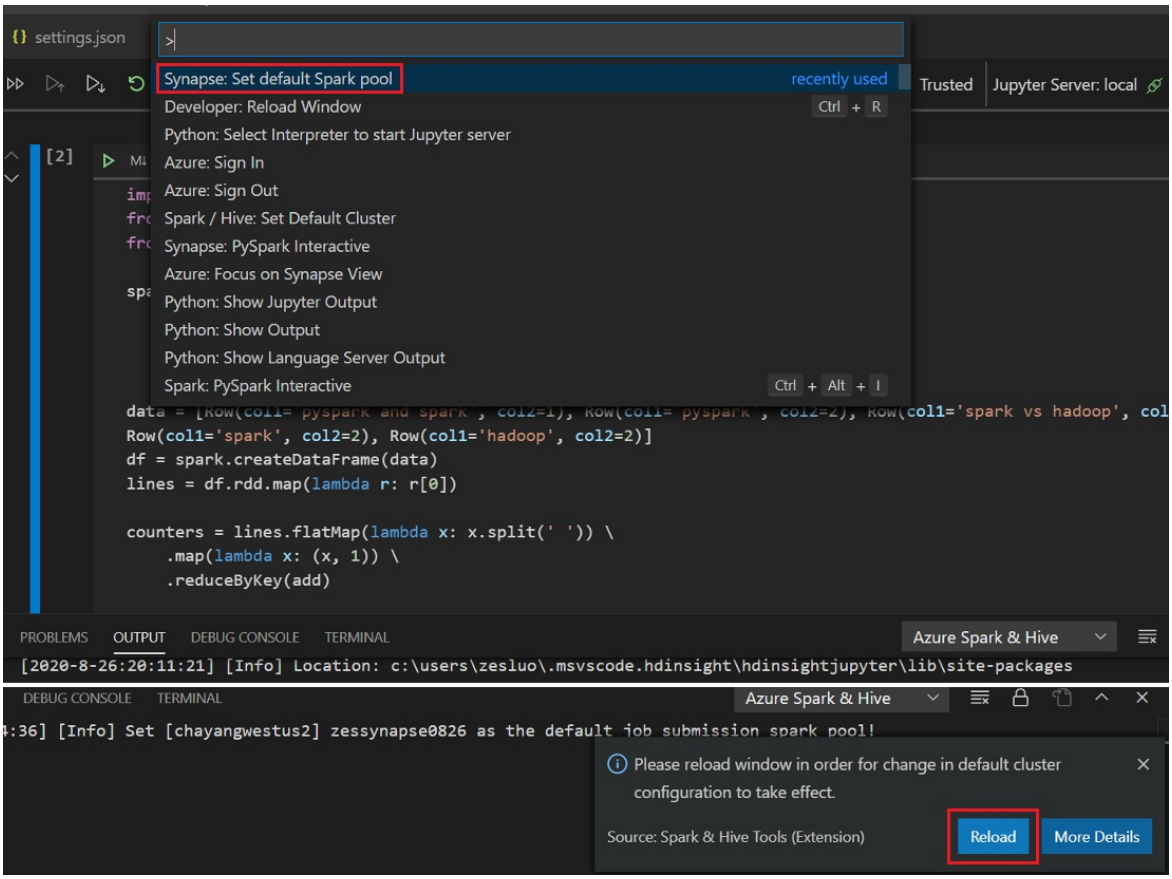
2. Click on **Run Cell**. After a few moments, the Python Interactive results appear in a new tab. Click on PySpark to switch the kernel to **Synapse PySpark**, then, click on **Run Cell** again, and the code will run successfully.



Leverage IPYNB support from Python extension

1. You can create a Jupyter Notebook by command from the Command Palette or by creating a new .ipynb file in your workspace. For more information, see Working with Jupyter Notebooks in Visual Studio Code

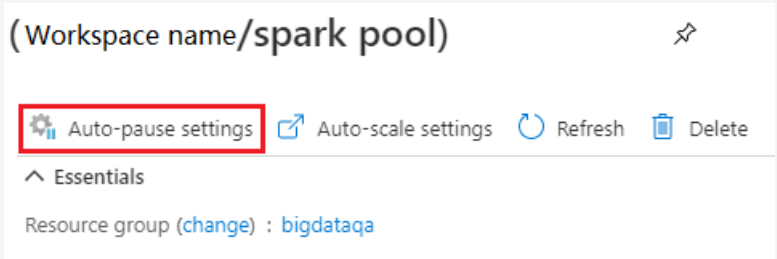
2. Click on **Run cell** button, follow the prompts to **Set the default spark pool** (strongly encourage to set default cluster/pool every time before opening a notebook) and then, **Reload** window.



3. Click on PySpark to switch kernel to **Synapse Pyspark**, and then click on **Run Cell**, after a while, the result will be displayed.

NOTE

1. Ms-python >=2020.5.78807 version is not supported on this extention is a known issue.
2. Switch to Synapse Pyspark kernel, disabling auto-settings in Azure Portal is encouraged. Otherwise it may take a long while to wake up the cluster and set synapse kernel for the first time use.



Spark session config

You can specify the timeout duration, the number, and the size of executors to give to the current Spark session in **Configure session**. Restart the Spark session is for configuration changes to take effect. All cached

notebook variables are cleared.

```
%%configure -f
{
    // refer to https://github.com/cloudera/livy#request-body for a list of valid parameters to config the
session.
    "driverMemory":"2g",
    "driverCores":3,
    "executorMemory":"2g",
    "executorCores":2,
    "jars":[],
    "conf":{
        "spark.driver.maxResultSize":"10g"
    }
}
```

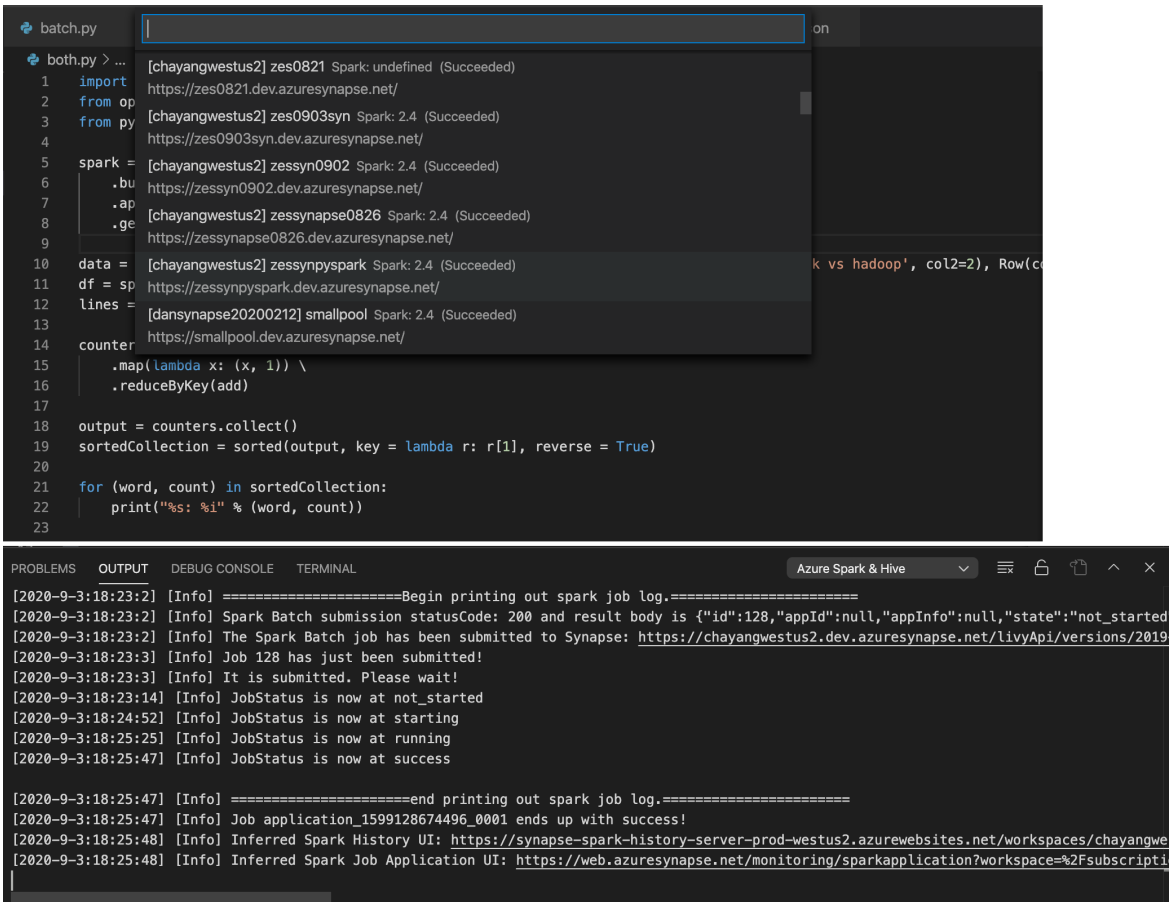
> **NOTE**
> Display function and Spark SQL may not be rendered properly in the output cell.

Submit PySpark batch job to Spark pool

1. Reopen the **Synaseexample** folder that you discussed earlier, if closed.
2. Create a new **BatchFile.py** file by following the earlier steps.
3. Copy and paste the following code into the script file:

```
from __future__ import print_function
import sys
from operator import add
from pyspark.sql import SparkSession
if __name__ == "__main__":
    spark = SparkSession\
        .builder\
        .appName("PythonWordCount")\
        .getOrCreate()

    lines = spark.read.text('/HdiSamples/HdiSamples/SensorSampleData/hvac/HVAC.csv').rdd.map(lambda
r: r[0])
    counts = lines.flatMap(lambda x: x.split(' '))\
                .map(lambda x: (x, 1))\
                .reduceByKey(add)
    output = counts.collect()
    for (word, count) in output:
        print("%s: %i" % (word, count))
    spark.stop()
```

4. Connect to your Azure account if you haven't yet done so.
5. Right-click the script editor, and then select **Synapse: PySpark Batch**.
6. Select a Spark pool to submit your PySpark job to:



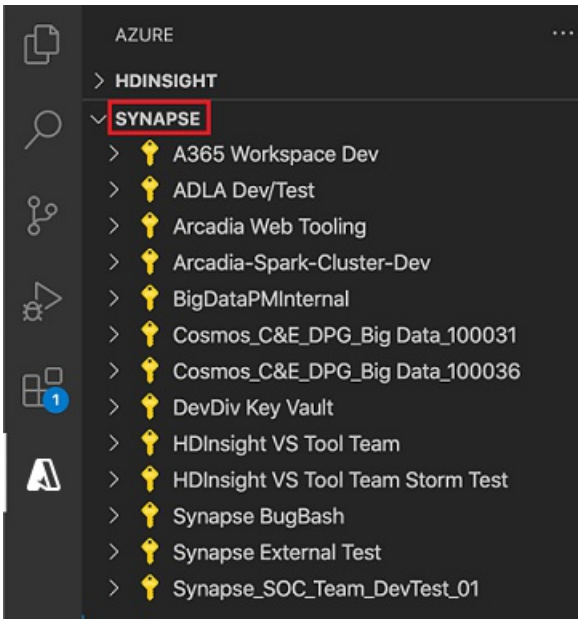
After you submit a batch job to spark pool, submission logs appear in the **OUTPUT** window in Visual Studio Code. The **Spark UI** URL and **Spark Job Application UI** URL are also shown. You can open the URL in a web browser to track the job status.

Access and manage Synapse Workspace

You can perform different operations in Azure Explorer within Spark & Hive tools for VSCode. From the Azure Explorer.

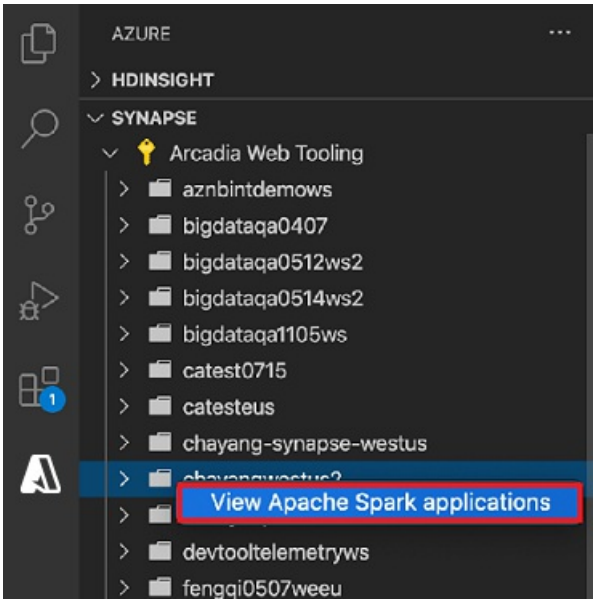
Launch workspace

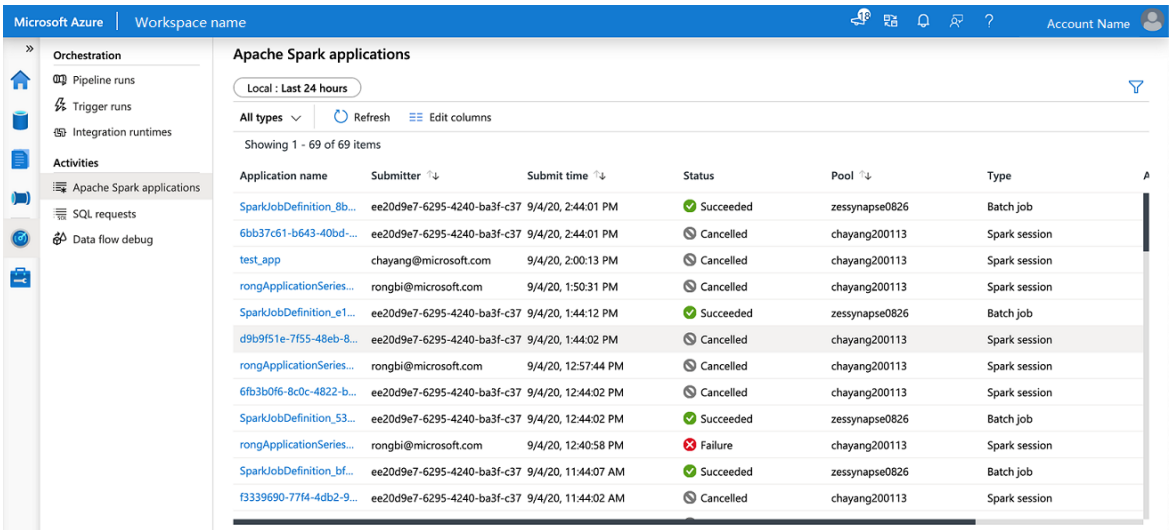
1. From Azure Explorer, navigate to **SYNAPSE**, expand it, and display the Synapse Subscription list.



2. Click on Subscription of Synapse workspace, expand it, and display the workspace list.

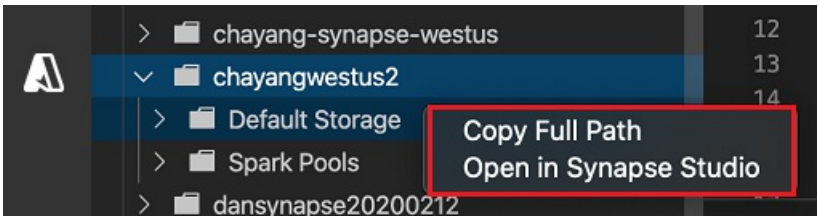
3. Right-click a workspace, then select **View Apache Spark applications**, the Apache Spark application page in the Synapse Studio website will be opened.



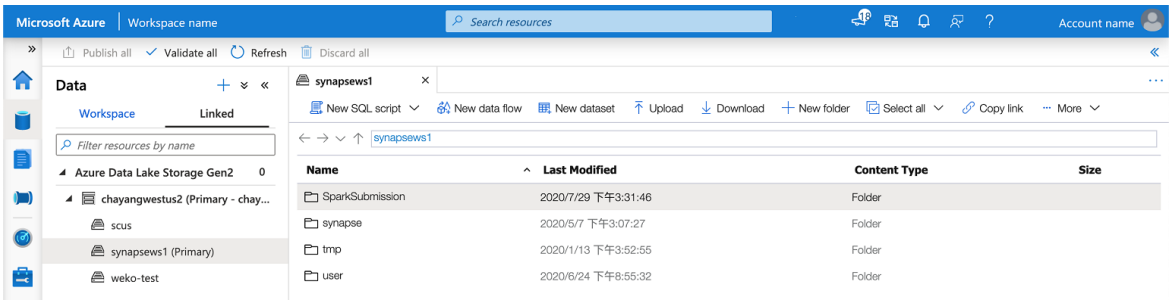


4. Expand a workspace, **Default Storage** and **Spark Pools** are displayed.

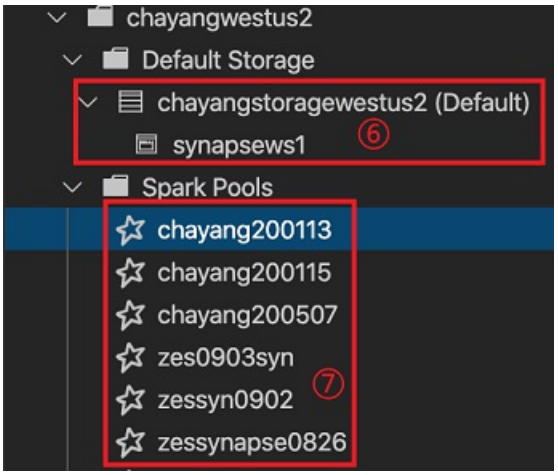
5. Right-click on **Default Storage**, the **Copy Full Path** and **Open in Synapse Studio** are displayed.



- Click on **Copy Full Path**, the Primary ADLS Gen2 account URL will be copied, you can paste it where you need。
- Click on **Open in Synapse Studio**, the Primary Storage Account will be opened in Synapse Studio.

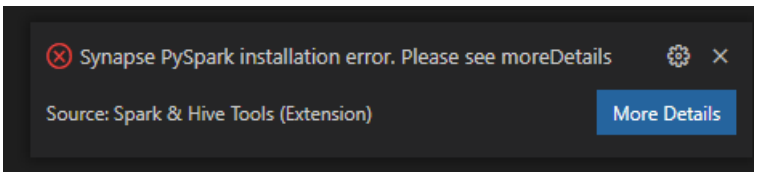


6. Expand the **Default Storage**, the Primary Storage Account is displayed.
7. Expand the **Spark Pools**, all spark pools in the workspace are displayed.



Known issues

Synapse PySpark installation error.



Not supported submit interactive Synapse PySpark queries to Spark pool anymore

For Synapse Pyspark interactive, since its dependency will not be maintained anymore by other team, this will not be maintained anymore as well. If you trying to use Synapse Pyspark interactive, please switch to use Azure Synapse Analytics instead. And it's a long term change.

Next steps

- Azure Synapse Analytics

- Create a new Apache Spark pool for an Azure Synapse Analytics workspace

Use .NET for Apache Spark with Azure Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

.NET for Apache Spark provides free, open-source, and cross-platform .NET support for Spark.

It provides .NET bindings for Spark, which allows you to access Spark APIs through C# and F#. With .NET for Apache Spark, you can also write and execute user-defined functions for Spark written in .NET. The .NET APIs for Spark enable you to access all aspects of Spark DataFrames that help you analyze your data, including Spark SQL, Delta Lake, and Structured Streaming.

You can analyze data with .NET for Apache Spark through Spark batch job definitions or with interactive Azure Synapse Analytics notebooks. In this article, you learn how to use .NET for Apache Spark with Azure Synapse using both techniques.

Submit batch jobs using the Spark job definition

Visit the tutorial to learn how to use Azure Synapse Analytics to create Apache Spark job definitions for Synapse Spark pools. If you haven't packaged your app to submit to Azure Synapse, complete the following steps.

1. Run the following commands to publish your app. Be sure to replace *mySparkApp* with the path to your app.

   ```
   cd mySparkApp
   dotnet publish -c Release -f netcoreapp3.1 -r ubuntu.18.04-x64
   ```

2. Zip the contents of the publish folder, `publish.zip` for example, that was created as a result of Step 1. All the assemblies should be in the first layer of the ZIP file and there should be no intermediate folder layer. This means when you unzip `publish.zip`, all assemblies are extracted into your current working directory.

   On Windows:

   Use an extraction program, like 7-Zip or WinZip, to extract the file into the bin directory with all the published binaries.

   On Linux:

   Open a bash shell and cd into the bin directory with all the published binaries and run the following command.

   ```
   zip -r publish.zip
   ```

.NET for Apache Spark in Azure Synapse Analytics notebooks

Notebooks are a great option for prototyping your .NET for Apache Spark pipelines and scenarios. You can start working with, understanding, filtering, displaying, and visualizing your data quickly and efficiently.

Data engineers, data scientists, business analysts, and machine learning engineers are all able to collaborate over a shared, interactive document. You see immediate results from data exploration, and can visualize your

data in the same notebook.

How to use .NET for Apache Spark notebooks

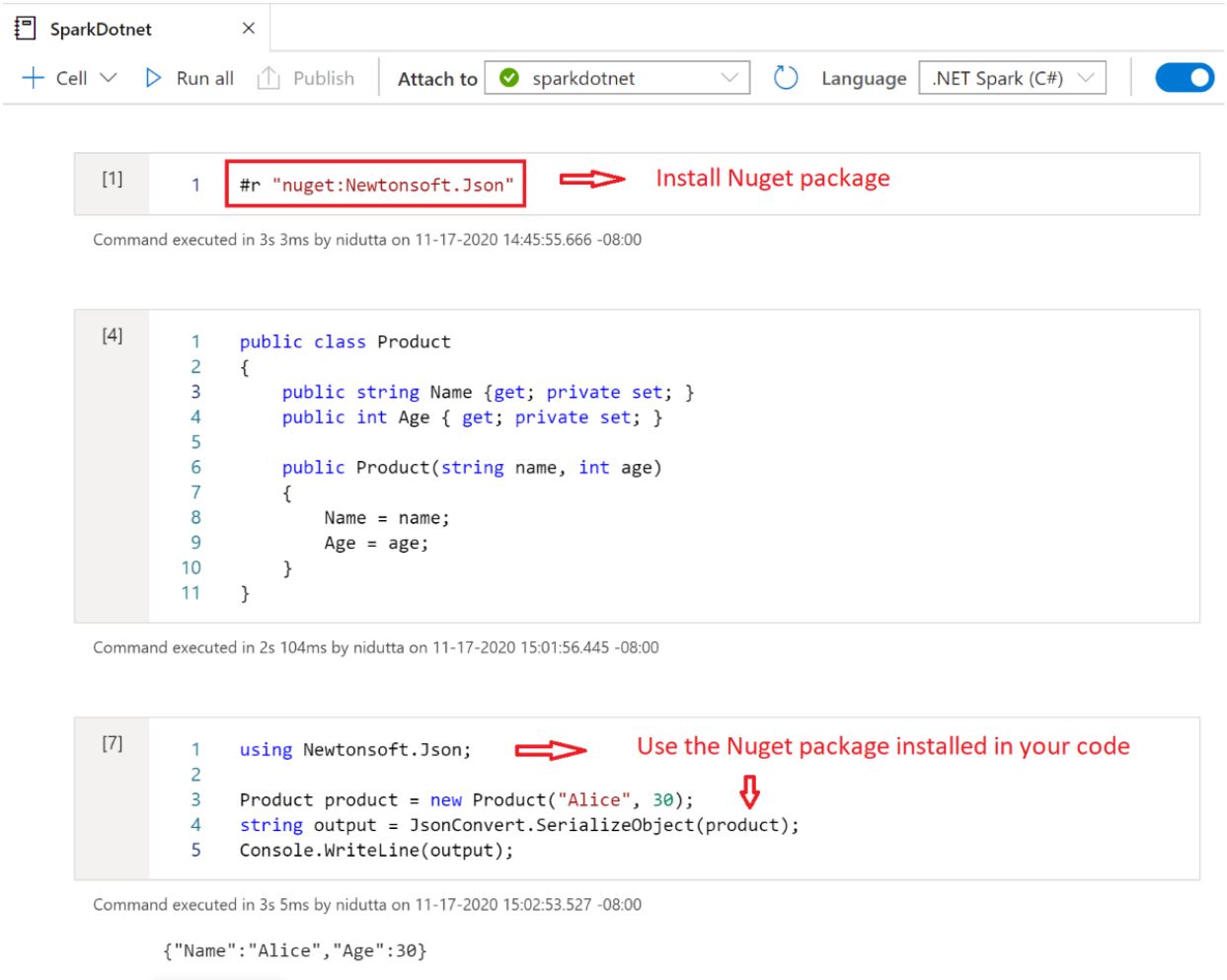
When you create a new notebook, you choose a language kernel that you wish to express your business logic. Kernel support is available for several languages, including C#.

To use .NET for Apache Spark in your Azure Synapse Analytics notebook, select **.NET Spark (C#)** as your kernel and attach the notebook to an existing serverless Apache Spark pool.

The .NET Spark notebook is based on the .NET interactive experiences and provides interactive C# experiences with the ability to use .NET for Spark out of the box with the Spark session variable `spark` already predefined.

Install NuGet packages in notebooks

You can install NuGet packages of your choice into your notebook by using the `#r nuget` magic command before the name of the NuGet package. The following diagram shows an example:



To learn more about how to work with NuGet packages in notebooks, see the .NET Interactive Documentation.

.NET for Apache Spark C# kernel features

The following features are available when you use .NET for Apache Spark in the Azure Synapse Analytics notebook:

- Declarative HTML: Generate output from your cells using HTML-syntax, such as headers, bulleted lists, and even displaying images.
- Simple C# statements (such as assignments, printing to console, throwing exceptions, and so on).
- Multi-line C# code blocks (such as if statements, foreach loops, class definitions, and so on).
- Access to the standard C# library (such as System, LINQ, Enumerables, and so on).
- Support for C# 8.0 language features.

- `spark` as a pre-defined variable to give you access to your Apache Spark session.
- Support for defining .NET user-defined functions that can run within Apache Spark. We recommend Write and call UDFs in .NET for Apache Spark Interactive environments for learning how to use UDFs in .NET for Apache Spark Interactive experiences.
- Support for visualizing output from your Spark jobs using different charts (such as line, bar, or histogram) and layouts (such as single, overlaid, and so on) using the `XPlot.Plotly` library.
- Ability to include NuGet packages into your C# notebook.

Next steps

- .NET for Apache Spark documentation
- .NET for Apache Spark Interactive guides
- Azure Synapse Analytics
- .NET Interactive

Secure credentials with linked services using the TokenLibrary

8/19/2022 • 5 minutes to read • Edit Online

Accessing data from external sources is a common pattern. Unless the external data source allows anonymous access, chances are you need to secure your connection with a credential, secret, or connection string.

Synapse uses Azure Active Directory (AAD) passthrough by default for authentication between resources. If you need to connect to a resource using other credentials, use the TokenLibrary directly. The TokenLibrary simplifies the process of retrieving SAS tokens, AAD tokens, connection strings, and secrets stored in a linked service or from an Azure Key Vault.

AAD passthrough uses permissions assigned to you as a user in AAD, rather than permissions assigned to Synapse or a separate service principal. For example, if you want to use AAD passthrough to access a blob in a storage account, then you should go to that storage account and assign blob contributor role to yourself.

When retrieving secrets from Azure Key Vault, we recommend creating a linked service to your Azure Key Vault. Ensure that the Synapse workspace managed service identity (MSI) has Secret Get privileges on your Azure Key Vault. Synapse will authenticate to Azure Key Vault using the Synapse workspace managed service identity. If you connect directly to Azure Key Vault without a linked service, you will authenticate using your user Azure Active Directory credential.

For more information, see linked services.

Usage

TokenLibrary.help()

This function displays the help documentation for the TokenLibrary.

```
TokenLibrary.help()
```

```
TokenLibrary.help()
```

```
Console.WriteLine(TokenLibrary.help());
```

TokenLibrary for Azure Data Lake Storage Gen2

ADLS Gen2 Primary Storage

Accessing files from the primary Azure Data Lake Storage uses Azure Active Directory passthrough for authentication by default and doesn't require the explicit use of the TokenLibrary.

```
val df = spark.read.csv("abfss://<CONTAINER>@<ACCOUNT>.dfs.core.windows.net/<FILE PATH>")
display(df.limit(10))
```

```
df = spark.read.csv('abfss://<CONTAINER>@<ACCOUNT>.dfs.core.windows.net/<FILE PATH>')
display(df.limit(10))
```

ADLS Gen2 storage with linked services

Synapse provides an integrated linked services experience when connecting to Azure Data Lake Storage Gen2. Linked Services can be configured to authenticate using an **Account Key**, **Service Principal**, **Managed Identity**, or **Credential**.

When the linked service authentication method is set to **Account Key**, the linked service will authenticate using the provided storage account key, request a SAS key, and automatically apply it to the storage request using the **LinkedServiceBasedSASProvider**.

Synapse allows users to set the linked service for a particular storage account. This makes it possible to read/write data from **multiple storage accounts** in a single spark application/query. Once we set **spark.storage.synapse.{source_full_storage_account_name}.linkedServiceName** for each storage account that will be used, Synapse figures out which linked service to use for a particular read/write operation. However if our spark job only deals with a single storage account, we can simply omit the storage account name and use **spark.storage.synapse.linkedServiceName**

```
val sc = spark.sparkContext
val source_full_storage_account_name = "teststorage.dfs.core.windows.net"
spark.conf.set(f"spark.storage.synapse.{source_full_storage_account_name}.linkedServiceName", "<LINKED
SERVICE NAME>")
spark.conf.set(f"fs.azure.account.auth.type.{source_full_storage_account_name}", "SAS")
spark.conf.set(f"fs.azure.sas.token.provider.type.{source_full_storage_account_name}",
"com.microsoft.azure.synapse.tokenlibrary.LinkedServiceBasedSASProvider")

val df = spark.read.csv("abfss://<CONTAINER>@<ACCOUNT>.dfs.core.windows.net/<FILE PATH>")

display(df.limit(10))
```

```
%%pyspark
# Python code
val source_full_storage_account_name = "teststorage.dfs.core.windows.net"
spark.conf.set(f"spark.storage.synapse.{source_full_storage_account_name}.linkedServiceName", "<lINKED
SERVICE NAME>")
spark.conf.set(f"fs.azure.account.auth.type.{source_full_storage_account_name}", "SAS")
spark.conf.set(f"fs.azure.sas.token.provider.type.{source_full_storage_account_name}",
"com.microsoft.azure.synapse.tokenlibrary.LinkedServiceBasedSASProvider")

df = spark.read.csv('abfss://<CONTAINER>@<ACCOUNT>.dfs.core.windows.net/<DIRECTORY PATH>')

df.show()
```

When the linked service authentication method is set to **Managed Identity** or **Service Principal**, the linked service will use the Managed Identity or Service Principal token with the **LinkedServiceBasedTokenProvider** provider.

```
val sc = spark.sparkContext
val source_full_storage_account_name = "teststorage.dfs.core.windows.net"
spark.conf.set(f"spark.storage.synapse.{source_full_storage_account_name}.linkedServiceName", "<LINKED
SERVICE NAME>")
spark.conf.set(f"fs.azure.account.oauth.provider.type.{source_full_storage_account_name}",
"com.microsoft.azure.synapse.tokenlibrary.LinkedServiceBasedTokenProvider")
val df = spark.read.csv("abfss://<CONTAINER>@<ACCOUNT>.dfs.core.windows.net/<FILE PATH>")

display(df.limit(10))
```

```
%%pyspark
# Python code
val source_full_storage_account_name = "teststorage.dfs.core.windows.net"
spark.conf.set(f"spark.storage.synapse.{source_full_storage_account_name}.linkedServiceName", "<LINKED
SERVICE NAME>")
spark.conf.set(f"fs.azure.account.oauth.provider.type.{source_full_storage_account_name}",
"com.microsoft.azure.synapse.tokenlibrary.LinkedServiceBasedTokenProvider")

df = spark.read.csv('abfss://<CONTAINER>@<ACCOUNT>.dfs.core.windows.net/<DIRECTORY PATH>')

df.show()
```

ADLS Gen2 storage (without linked services)

Connect to ADLS Gen2 storage directly by using a SAS key use the **ConfBasedSASProvider** and provide the SAS key to the **spark.storage.synapse.sas** configuration setting.

```
%%spark
spark.conf.set("fs.azure.account.auth.type", "SAS")
spark.conf.set("fs.azure.sas.token.provider.type",
"com.microsoft.azure.synapse.tokenlibrary.ConfBasedSASProvider")
spark.conf.set("spark.storage.synapse.sas", "<SAS KEY>")

val df = spark.read.csv("abfss://<CONTAINER>@<ACCOUNT>.dfs.core.windows.net/<FILE PATH>")

display(df.limit(10))
```

```
%%pyspark

spark.conf.set("fs.azure.account.auth.type", "SAS")
spark.conf.set("fs.azure.sas.token.provider.type",
"com.microsoft.azure.synapse.tokenlibrary.ConfBasedSASProvider")
spark.conf.set("spark.storage.synapse.sas", "<SAS KEY>")

df = spark.read.csv('abfss://<CONTAINER>@<ACCOUNT>.dfs.core.windows.net/<FILE PATH>')

display(df.limit(10))
```

ADLS Gen2 storage with Azure Key Vault

Connect to ADLS Gen2 storage using a SAS token stored in Azure Key Vault secret.

```
%%spark
spark.conf.set("fs.azure.account.auth.type", "SAS")
spark.conf.set("fs.azure.sas.token.provider.type",
"com.microsoft.azure.synapse.tokenlibrary.AkvBasedSASProvider")
spark.conf.set("spark.storage.synapse.akv", "<AZURE KEY VAULT NAME>")
spark.conf.set("spark.storage.akv.secret", "<SECRET KEY>")

val df = spark.read.csv("abfss://<CONTAINER>@<ACCOUNT>.dfs.core.windows.net/<FILE PATH>")

display(df.limit(10))
```

```
%%pyspark
spark.conf.set("fs.azure.account.auth.type", "SAS")
spark.conf.set("fs.azure.sas.token.provider.type",
"com.microsoft.azure.synapse.tokenlibrary.AkvBasedSASProvider")
spark.conf.set("spark.storage.synapse.akv", "<AZURE KEY VAULT NAME>")
spark.conf.set("spark.storage.akv.secret", "<SECRET KEY>")

df = spark.read.csv('abfss://<CONTAINER>@<ACCOUNT>.dfs.core.windows.net/<FILE PATH>')

display(df.limit(10))
```

TokenLibrary for other linked services

To connect to other linked services, you can make a direct call to the TokenLibrary.

getConnectionString()

To retrieve the connection string, use the **getConnectionString** function and pass in the **linked service name**.

```
%%spark
// retrieve connectionstring from TokenLibrary

import com.microsoft.azure.synapse.tokenlibrary.TokenLibrary

val connectionString: String = TokenLibrary.getConnectionString("<LINKED SERVICE NAME>")
println(connectionString)
```

```
%%pyspark
# retrieve connectionstring from TokenLibrary

from pyspark.sql import SparkSession

sc = SparkSession.builder.getOrCreate()
token_library = sc._jvm.com.microsoft.azure.synapse.tokenlibrary.TokenLibrary
connection_string = token_library.getConnectionString("<LINKED SERVICE NAME>")
print(connection_string)
```

```
%%csharp
// retrieve connectionstring from TokenLibrary

using Microsoft.Spark.Extensions.Azure.Synapse.Analytics.Utils;

string connectionString = TokenLibrary.GetConnectionString(<LINKED SERVICE NAME>);
Console.WriteLine(connectionString);
```

getConnectionStringAsMap()

The getConnectionStringAsMap is a helper function available in Scala and Python to parse specific values from a *key=value* pair in the connection string such as

```
DefaultEndpointsProtocol=https;AccountName=<ACCOUNT NAME>;AccountKey=<ACCOUNT KEY>
```

use the **getConnectionStringAsMap** function and pass the key to return the value. In the above connection string example,

TokenLibrary.getConnectionStringAsMap("DefaultEndpointsProtocol")

would return

"https"

```
// Linked services can be used for storing and retrieving credentials (e.g, account key)
// Example connection string (for storage): "DefaultEndpointsProtocol=https;AccountName=
<accountname>;AccountKey=<accountkey>"
import com.microsoft.azure.synapse.tokenlibrary.TokenLibrary

val accountKey: String = TokenLibrary.getConnectionStringAsMap("<LINKED SERVICE NAME">).get("<KEY NAME>")
println(accountKey)
```

```
# Linked services can be used for storing and retrieving credentials (e.g, account key)
# Example connection string (for storage): "DefaultEndpointsProtocol=https;AccountName=
<accountname>;AccountKey=<accountkey>"
from pyspark.sql import SparkSession

sc = SparkSession.builder.getOrCreate()
token_library = sc._jvm.com.microsoft.azure.synapse.tokenlibrary.TokenLibrary
accountKey = token_library.getConnectionStringAsMap("<LINKED SERVICE NAME>").get("<KEY NAME>")
print(accountKey)
```

getSecret()

To retrieve a secret stored from Azure Key Vault, we recommend that you create a linked service to Azure Key Vault within the Synapse workspace. The Synapse workspace managed service identity will need to be granted **GET** Secrets permission to the Azure Key Vault. The linked service will use the managed service identity to connect to Azure Key Vault service to retrieve the secret. Otherwise, connecting directly to Azure Key Vault will use the user's Azure Active Directory (AAD) credential. In this case, the user will need to be granted the Get Secret permissions in Azure Key Vault.

```
TokenLibrary.getSecret("<AZURE KEY VAULT NAME>", "<SECRET KEY>" [, <LINKED SERVICE NAME>])
```

To retrieve a secret from Azure Key Vault, use the **TokenLibrary.getSecret()** function.

```
import com.microsoft.azure.synapse.tokenlibrary.TokenLibrary

val connectionString: String = TokenLibrary.getSecret("<AZURE KEY VAULT NAME>", "<SECRET KEY>", "<LINKED
SERVICE NAME>")
println(connectionString)
```

```
import sys
from pyspark.sql import SparkSession

sc = SparkSession.builder.getOrCreate()
token_library = sc._jvm.com.microsoft.azure.synapse.tokenlibrary.TokenLibrary

connection_string = token_library.getSecret("<AZURE KEY VAULT NAME>", "<SECRET KEY>", "<LINKED SERVICE
NAME>")
print(connection_string)
```

```
using Microsoft.Spark.Extensions.Azure.Synapse.Analytics.Utils;

string connectionString = TokenLibrary.GetSecret("<AZURE KEY VAULT NAME>", "<SECRET KEY>", "<LINKED SERVICE
NAME>");
Console.WriteLine(connectionString);
```

Next steps

- Write to dedicated SQL pool

Optimize Apache Spark jobs in Azure Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

Learn how to optimize an Apache Spark cluster configuration for your particular workload. The most common challenge is memory pressure, because of improper configurations (particularly wrong-sized executors), long-running operations, and tasks that result in Cartesian operations. You can speed up jobs with appropriate caching, and by allowing for data skew. For the best performance, monitor and review long-running and resource-consuming Spark job executions.

The following sections describe common Spark job optimizations and recommendations.

Choose the data abstraction

Earlier Spark versions use RDDs to abstract data, Spark 1.3, and 1.6 introduced DataFrames and DataSets, respectively. Consider the following relative merits:

- **DataFrames**
  - Best choice in most situations.
  - Provides query optimization through Catalyst.
  - Whole-stage code generation.
  - Direct memory access.
  - Low garbage collection (GC) overhead.
  - Not as developer-friendly as DataSets, as there are no compile-time checks or domain object programming.
- **DataSets**
  - Good in complex ETL pipelines where the performance impact is acceptable.
  - Not good in aggregations where the performance impact can be considerable.
  - Provides query optimization through Catalyst.
  - Developer-friendly by providing domain object programming and compile-time checks.
  - Adds serialization/deserialization overhead.
  - High GC overhead.
  - Breaks whole-stage code generation.
- **RDDs**
  - You don't need to use RDDs, unless you need to build a new custom RDD.
  - No query optimization through Catalyst.
  - No whole-stage code generation.
  - High GC overhead.
  - Must use Spark 1.x legacy APIs.

Use optimal data format

Spark supports many formats, such as csv, json, xml, parquet, orc, and avro. Spark can be extended to support many more formats with external data sources - for more information, see Apache Spark packages.

The best format for performance is parquet with *snappy compression*, which is the default in Spark 2.x. Parquet stores data in columnar format, and is highly optimized in Spark. In addition, while *snappy compression* may

result in larger files than say gzip compression. Due to the splittable nature of those files, they will decompress faster.

Use the cache

Spark provides its own native caching mechanisms, which can be used through different methods such as `.persist()`, `.cache()`, and `CACHE TABLE`. This native caching is effective with small data sets as well as in ETL pipelines where you need to cache intermediate results. However, Spark native caching currently doesn't work well with partitioning, since a cached table doesn't keep the partitioning data.

Use memory efficiently

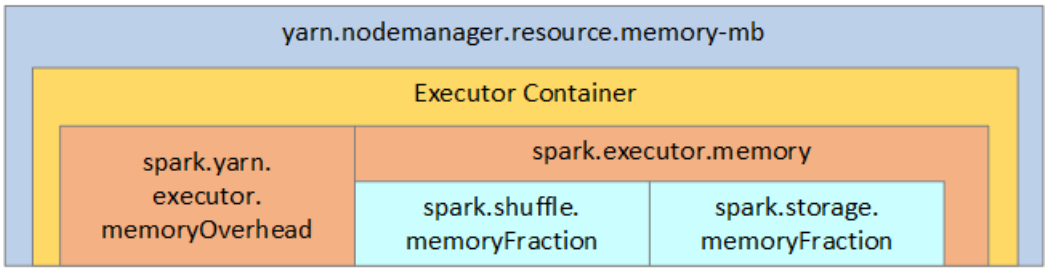
Spark operates by placing data in memory, so managing memory resources is a key aspect of optimizing the execution of Spark jobs. There are several techniques you can apply to use your cluster's memory efficiently.

- Prefer smaller data partitions and account for data size, types, and distribution in your partitioning strategy.
- Consider the newer, more efficient Kryo data serialization, rather than the default Java serialization.
- Monitor and tune Spark configuration settings.

For your reference, the Spark memory structure and some key executor memory parameters are shown in the next image.

Spark memory considerations

Apache Spark in Azure Synapse uses YARN Apache Hadoop YARN, YARN controls the maximum sum of memory used by all containers on each Spark node. The following diagram shows the key objects and their relationships.



To address 'out of memory' messages, try:

- Review DAG Management Shuffles. Reduce by map-side reducing, pre-partition (or bucketize) source data, maximize single shuffles, and reduce the amount of data sent.
- Prefer `ReduceByKey` with its fixed memory limit to `GroupByKey`, which provides aggregations, windowing, and other functions but it has an unbounded memory limit.
- Prefer `TreeReduce`, which does more work on the executors or partitions, to `Reduce`, which does all work on the driver.
- Leverage DataFrames rather than the lower-level RDD objects.
- Create ComplexTypes that encapsulate actions, such as "Top N", various aggregations, or windowing operations.

Optimize data serialization

Spark jobs are distributed, so appropriate data serialization is important for the best performance. There are two serialization options for Spark:

- Java serialization is the default.
- Kryo serialization is a newer format and can result in faster and more compact serialization than Java. Kryo

requires that you register the classes in your program, and it doesn't yet support all Serializable types.

Use bucketing

Bucketing is similar to data partitioning, but each bucket can hold a set of column values rather than just one. Bucketing works well for partitioning on large (in the millions or more) numbers of values, such as product identifiers. A bucket is determined by hashing the bucket key of the row. Bucketed tables offer unique optimizations because they store metadata about how they were bucketed and sorted.

Some advanced bucketing features are:

- Query optimization based on bucketing meta-information.
- Optimized aggregations.
- Optimized joins.

You can use partitioning and bucketing at the same time.

Optimize joins and shuffles

If you have slow jobs on a Join or Shuffle, the cause is probably *data skew*, which is asymmetry in your job data. For example, a map job may take 20 seconds, but running a job where the data is joined or shuffled takes hours. To fix data skew, you should salt the entire key, or use an *isolated salt* for only some subset of keys. If you're using an isolated salt, you should further filter to isolate your subset of salted keys in map joins. Another option is to introduce a bucket column and pre-aggregate in buckets first.

Another factor causing slow joins could be the join type. By default, Spark uses the `SortMerge` join type. This type of join is best suited for large data sets, but is otherwise computationally expensive because it must first sort the left and right sides of data before merging them.

A `Broadcast` join is best suited for smaller data sets, or where one side of the join is much smaller than the other side. This type of join broadcasts one side to all executors, and so requires more memory for broadcasts in general.

You can change the join type in your configuration by setting `spark.sql.autoBroadcastJoinThreshold`, or you can set a join hint using the DataFrame APIs (`dataframe.join(broadcast(df2))`).

```
// Option 1
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", 1*1024*1024*1024)

// Option 2
val df1 = spark.table("FactTableA")
val df2 = spark.table("dimMP")
df1.join(broadcast(df2), Seq("PK")).
    createOrReplaceTempView("V_JOIN")

sql("SELECT col1, col2 FROM V_JOIN")
```

If you're using bucketed tables, then you have a third join type, the `Merge` join. A correctly pre-partitioned and pre-sorted dataset will skip the expensive sort phase from a `SortMerge` join.

The order of joins matters, particularly in more complex queries. Start with the most selective joins. Also, move joins that increase the number of rows after aggregations when possible.

To manage parallelism for Cartesian joins, you can add nested structures, windowing, and perhaps skip one or more steps in your Spark Job.

Select the correct executor size

When deciding your executor configuration, consider the Java garbage collection (GC) overhead.

- Factors to reduce executor size:
  - Reduce heap size below 32 GB to keep GC overhead < 10%.
  - Reduce the number of cores to keep GC overhead < 10%.
- Factors to increase executor size:
  - Reduce communication overhead between executors.
  - Reduce the number of open connections between executors (N2) on larger clusters (>100 executors).
  - Increase heap size to accommodate for memory-intensive tasks.
  - Optional: Reduce per-executor memory overhead.
  - Optional: Increase utilization and concurrency by oversubscribing CPU.

As a general rule of thumb when selecting the executor size:

- Start with 30 GB per executor and distribute available machine cores.
- Increase the number of executor cores for larger clusters (> 100 executors).
- Modify size based both on trial runs and on the preceding factors such as GC overhead.

When running concurrent queries, consider the following:

- Start with 30 GB per executor and all machine cores.
- Create multiple parallel Spark applications by oversubscribing CPU (around 30% latency improvement).
- Distribute queries across parallel applications.
- Modify size based both on trial runs and on the preceding factors such as GC overhead.

Monitor your query performance for outliers or other performance issues, by looking at the timeline view, SQL graph, job statistics, and so forth. Sometimes one or a few of the executors are slower than the others, and tasks take much longer to execute. This frequently happens on larger clusters (> 30 nodes). In this case, divide the work into a larger number of tasks so the scheduler can compensate for slow tasks.

For example, have at least twice as many tasks as the number of executor cores in the application. You can also enable speculative execution of tasks with `conf: spark.speculation = true`.

Optimize job execution

- Cache as necessary, for example if you use the data twice, then cache it.
- Broadcast variables to all executors. The variables are only serialized once, resulting in faster lookups.
- Use the thread pool on the driver, which results in faster operation for many tasks.

Key to Spark 2.x query performance is the Tungsten engine, which depends on whole-stage code generation. In some cases, whole-stage code generation may be disabled.

For example, if you use a non-mutable type (`string`) in the aggregation expression, `SortAggregate` appears instead of `HashAggregate`. For example, for better performance, try the following and then re-enable code generation:

```
MAX(AMOUNT) -> MAX(cast(AMOUNT as DOUBLE))
```

Next steps

- Tuning Apache Spark
- How to Actually Tune Your Apache Spark Jobs So They Work

- Kryo Serialization

Hyperspace: An indexing subsystem for Apache Spark

8/19/2022 • 38 minutes to read • Edit Online

Hyperspace introduces the ability for Apache Spark users to create indexes on their datasets, such as CSV, JSON, and Parquet, and use them for potential query and workload acceleration.

In this article, we highlight the basics of Hyperspace, emphasize its simplicity, and show how it can be used by just about anyone.

Disclaimer: Hyperspace helps accelerate your workloads or queries under two circumstances:

- Queries contain filters on predicates with high selectivity. For example, you might want to select 100 matching rows from a million candidate rows.
- Queries contain a join that requires heavy shuffles. For example, you might want to join a 100-GB dataset with a 10-GB dataset.

You might want to carefully monitor your workloads and determine whether indexing is helping you on a case-by-case basis.

This document is also available in notebook form, for Python, C#, and Scala.

Setup

To begin with, start a new Spark session. Since this document is a tutorial merely to illustrate what Hyperspace can offer, you will make a configuration change that allows us to highlight what Hyperspace is doing on small datasets.

By default, Spark uses broadcast join to optimize join queries when the data size for one side of join is small (which is the case for the sample data we use in this tutorial). Therefore, we disable broadcast joins so that later when we run join queries, Spark uses sort-merge join. This is mainly to show how Hyperspace indexes would be used at scale for accelerating join queries.

The output of running the following cell shows a reference to the successfully created Spark session and prints out '-1' as the value for the modified join config, which indicates that broadcast join is successfully disabled.

```
// Start your Spark session
spark

// Disable BroadcastHashJoin, so Spark will use standard SortMergeJoin. Currently, Hyperspace indexes
utilize SortMergeJoin to speed up query.
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)

// Verify that BroadcastHashJoin is set correctly
println(spark.conf.get("spark.sql.autoBroadcastJoinThreshold"))
```

```
# Start your Spark session.
spark

# Disable BroadcastHashJoin, so Spark will use standard SortMergeJoin. Currently, Hyperspace indexes utilize
SortMergeJoin to speed up query.
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)

# Verify that BroadcastHashJoin is set correctly
print(spark.conf.get("spark.sql.autoBroadcastJoinThreshold"))
```

```
// Disable BroadcastHashJoin, so Spark will use standard SortMergeJoin. Currently, Hyperspace indexes
utilize SortMergeJoin to speed up query.
spark.Conf().Set("spark.sql.autoBroadcastJoinThreshold", -1);

// Verify that BroadcastHashJoin is set correctly.
Console.WriteLine(spark.Conf().Get("spark.sql.autoBroadcastJoinThreshold"));
```

Results in:

```
res3: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@297e957d
-1
```

Data preparation

To prepare your environment, you'll create sample data records and save them as Parquet data files. Parquet is used for illustration, but you can also use other formats such as CSV. In the subsequent cells, you'll see how you can create several Hyperspace indexes on this sample dataset and make Spark use them when running queries.

The example records correspond to two datasets: department and employee. You should configure the "empLocation" and "deptLocation" paths so that on the storage account they point to your desired location to save generated data files.

The output of running the following cell shows contents of our datasets as lists of triplets followed by references to dataFrames created to save the content of each dataset in our preferred location.

```
import org.apache.spark.sql.DataFrame

// Sample department records
val departments = Seq(
      (10, "Accounting", "New York"),
      (20, "Research", "Dallas"),
      (30, "Sales", "Chicago"),
      (40, "Operations", "Boston"))

// Sample employee records
val employees = Seq(
      (7369, "SMITH", 20),
      (7499, "ALLEN", 30),
      (7521, "WARD", 30),
      (7566, "JONES", 20),
      (7698, "BLAKE", 30),
      (7782, "CLARK", 10),
      (7788, "SCOTT", 20),
      (7839, "KING", 10),
      (7844, "TURNER", 30),
      (7876, "ADAMS", 20),
      (7900, "JAMES", 30),
      (7934, "MILLER", 10),
      (7902, "FORD", 20),
      (7654, "MARTIN", 30))

// Save sample data in the Parquet format
import spark.implicits._
val empData: DataFrame = employees.toDF("empId", "empName", "deptId")
val deptData: DataFrame = departments.toDF("deptId", "deptName", "location")

val empLocation: String = "/<yourpath>/employees.parquet"       //TODO ** customize this location path **
val deptLocation: String = "/<yourpath>/departments.parquet"     //TODO ** customize this location path **
empData.write.mode("overwrite").parquet(empLocation)
deptData.write.mode("overwrite").parquet(deptLocation)
```

```
from pyspark.sql.types import StructField, StructType, StringType, IntegerType

# Sample department records
departments = [(10, "Accounting", "New York"), (20, "Research", "Dallas"), (30, "Sales", "Chicago"), (40,
"Operations", "Boston")]

# Sample employee records
employees = [(7369, "SMITH", 20), (7499, "ALLEN", 30), (7521, "WARD", 30), (7566, "JONES", 20), (7698,
"BLAKE", 30)]

# Create a schema for the dataframe
dept_schema = StructType([StructField('deptId', IntegerType(), True), StructField('deptName', StringType(),
True), StructField('location', StringType(), True)])
emp_schema = StructType([StructField('empId', IntegerType(), True), StructField('empName', StringType(),
True), StructField('deptId', IntegerType(), True)])

departments_df = spark.createDataFrame(departments, dept_schema)
employees_df = spark.createDataFrame(employees, emp_schema)

#TODO ** customize this location path **
emp_Location = "/<yourpath>/employees.parquet"
dept_Location = "/<yourpath>/departments.parquet"

employees_df.write.mode("overwrite").parquet(emp_Location)
departments_df.write.mode("overwrite").parquet(dept_Location)
```

```
using Microsoft.Spark.Sql.Types;

// Sample department records
var departments = new List<GenericRow>()
{
    new GenericRow(new object[] {10, "Accounting", "New York"}),
    new GenericRow(new object[] {20, "Research", "Dallas"}),
    new GenericRow(new object[] {30, "Sales", "Chicago"}),
    new GenericRow(new object[] {40, "Operations", "Boston"})
};

// Sample employee records
var employees = new List<GenericRow>() {
      new GenericRow(new object[] {7369, "SMITH", 20}),
      new GenericRow(new object[] {7499, "ALLEN", 30}),
      new GenericRow(new object[] {7521, "WARD", 30}),
      new GenericRow(new object[] {7566, "JONES", 20}),
      new GenericRow(new object[] {7698, "BLAKE", 30}),
      new GenericRow(new object[] {7782, "CLARK", 10}),
      new GenericRow(new object[] {7788, "SCOTT", 20}),
      new GenericRow(new object[] {7839, "KING", 10}),
      new GenericRow(new object[] {7844, "TURNER", 30}),
      new GenericRow(new object[] {7876, "ADAMS", 20}),
      new GenericRow(new object[] {7900, "JAMES", 30}),
      new GenericRow(new object[] {7934, "MILLER", 10}),
      new GenericRow(new object[] {7902, "FORD", 20}),
      new GenericRow(new object[] {7654, "MARTIN", 30})
};

// Save sample data in the Parquet format
var departmentSchema = new StructType(new List<StructField>()
{
    new StructField("deptId", new IntegerType()),
    new StructField("deptName", new StringType()),
    new StructField("location", new StringType())
});
var employeeSchema = new StructType(new List<StructField>()
{
    new StructField("empId", new IntegerType()),
    new StructField("empName", new StringType()),
    new StructField("deptId", new IntegerType())
});

DataFrame empData = spark.CreateDataFrame(employees, employeeSchema);
DataFrame deptData = spark.CreateDataFrame(departments, departmentSchema);

string empLocation = "/<yourpath>/employees.parquet";       //TODO ** customize this location path **
string deptLocation = "/<yourpath>/departments.parquet";     //TODO ** customize this location path **
empData.Write().Mode("overwrite").Parquet(empLocation);
deptData.Write().Mode("overwrite").Parquet(deptLocation);
```

Results in:

```
departments: Seq[(Int, String, String)] = List((10,Accounting,New York), (20,Research,Dallas),
(30,Sales,Chicago), (40,Operations,Boston))
employees: Seq[(Int, String, Int)] = List((7369,SMITH,20), (7499,ALLEN,30), (7521,WARD,30), (7566,JONES,20),
(7698,BLAKE,30), (7782,CLARK,10), (7788,SCOTT,20), (7839,KING,10), (7844,TURNER,30), (7876,ADAMS,20),
(7900,JAMES,30), (7934,MILLER,10), (7902,FORD,20), (7654,MARTIN,30))

empData: org.apache.spark.sql.DataFrame = [empId: int, empName: string ... 1 more field]
deptData: org.apache.spark.sql.DataFrame = [deptId: int, deptName: string ... 1 more field]
empLocation: String = /your-path/employees.parquet
deptLocation: String = /your-path/departments.parquet
```

Let's verify the contents of the Parquet files we created to make sure they contain expected records in the correct format. Later, we'll use these data files to create Hyperspace indexes and run sample queries.

Running the following cell produces and output that displays the rows in employee and department dataFrames in a tabular form. There should be 14 employees and 4 departments, each matching with one of triplets you created in the previous cell.

```
// empLocation and deptLocation are the user defined locations above to save parquet files
val empDF: DataFrame = spark.read.parquet(empLocation)
val deptDF: DataFrame = spark.read.parquet(deptLocation)

// Verify the data is available and correct
empDF.show()
deptDF.show()
```

```
# emp_Location and dept_Location are the user-defined locations above to save parquet files
emp_DF = spark.read.parquet(emp_Location)
dept_DF = spark.read.parquet(dept_Location)

# Verify the data is available and correct
emp_DF.show()
dept_DF.show()
```

```
// empLocation and deptLocation are the user-defined locations above to save parquet files
DataFrame empDF = spark.Read().Parquet(empLocation);
DataFrame deptDF = spark.Read().Parquet(deptLocation);

// Verify the data is available and correct
empDF.Show();
deptDF.Show();
```

Results in:

```
empDF: org.apache.spark.sql.DataFrame = [empId: int, empName: string ... 1 more field]
deptDF: org.apache.spark.sql.DataFrame = [deptId: int, deptName: string ... 1 more field]
```

```
|EmpId|EmpName|DeptId|
|-----|-------|------|
| 7499|  ALLEN|    30|
| 7521|   WARD|    30|
| 7369|  SMITH|    20|
| 7844| TURNER|    30|
| 7876|  ADAMS|    20|
| 7900|  JAMES|    30|
| 7934| MILLER|    10|
| 7839|   KING|    10|
| 7566|  JONES|    20|
| 7698|  BLAKE|    30|
| 7782|  CLARK|    10|
| 7788|  SCOTT|    20|
| 7902|   FORD|    20|
| 7654| MARTIN|    30|
```

```
|DeptId|  DeptName|Location|
|------|----------|--------|
|    10|Accounting|New York|
|    40|Operations|  Boston|
|    20|  Research|  Dallas|
|    30|     Sales| Chicago|
```

Indexes

Hyperspace lets you create indexes on records scanned from persisted data files. After they're successfully created, an entry that corresponds to the index is added to the Hyperspace's metadata. This metadata is later used by Apache Spark's optimizer (with our extensions) during query processing to find and use proper indexes.

After indexes are created, you can perform several actions:

- **Refresh if the underlying data changes.** You can refresh an existing index to capture the changes.
- **Delete if the index isn't needed.** You can perform a soft delete, that is, the index isn't physically deleted but is marked as "deleted" so that it's no longer used in your workloads.
- **Vacuum if an index is no longer required.** You can vacuum an index, which forces a physical deletion of the index contents and associated metadata completely from Hyperspace's metadata.

Refresh if the underlying data changes, you can refresh an existing index to capture that. Delete if the index is not needed, you can perform a soft-delete that is, index is not physically deleted but is marked as 'deleted' so it is no longer used in your workloads.

The following sections show how such index management operations can be done in Hyperspace.

First, you need to import the required libraries and create an instance of Hyperspace. Later, you'll use this instance to invoke different Hyperspace APIs to create indexes on your sample data and modify those indexes.

The output of running the following cell shows a reference to the created instance of Hyperspace.

```
// Create an instance of Hyperspace
import com.microsoft.hyperspace._

val hyperspace: Hyperspace = Hyperspace()
```

```
from hyperspace import *

# Create an instance of Hyperspace
hyperspace = Hyperspace(spark)
```

```
// Create an instance of Hyperspace
using Microsoft.Spark.Extensions.Hyperspace;

Hyperspace hyperspace = new Hyperspace(spark);
```

Results in:

```
hyperspace: com.microsoft.hyperspace.Hyperspace = com.microsoft.hyperspace.Hyperspace@1432f740
```

Create indexes

To create a Hyperspace index, you need to provide two pieces of information:

- A Spark DataFrame that references the data to be indexed.
- An index configuration object, IndexConfig, which specifies the index name and the indexed and included columns of the index.

You start by creating three Hyperspace indexes on our sample data: two indexes on the department dataset named "deptIndex1" and "deptIndex2" and one index on the employee dataset named "empIndex". For each index, you need a corresponding IndexConfig to capture the name along with columns lists for the indexed and included columns. Running the following cell creates these IndexConfigs, and its output lists them.

> **NOTE**
> An index column is a column that appears in your filters or join conditions. An included column is a column that appears in your select/project.

For instance, in the following query:

```
SELECT X
FROM T
WHERE Y = 2
```

Y can be an index column, and X can be an included column.

```
// Create index configurations
import com.microsoft.hyperspace.index.IndexConfig

val empIndexConfig: IndexConfig = IndexConfig("empIndex", Seq("deptId"), Seq("empName"))
val deptIndexConfig1: IndexConfig = IndexConfig("deptIndex1", Seq("deptId"), Seq("deptName"))
val deptIndexConfig2: IndexConfig = IndexConfig("deptIndex2", Seq("location"), Seq("deptName"))
```

```
# Create index configurations

emp_IndexConfig = IndexConfig("empIndex1", ["deptId"], ["empName"])
dept_IndexConfig1 = IndexConfig("deptIndex1", ["deptId"], ["deptName"])
dept_IndexConfig2 = IndexConfig("deptIndex2", ["location"], ["deptName"])
```

```
using Microsoft.Spark.Extensions.Hyperspace.Index;

var empIndexConfig = new IndexConfig("empIndex", new string[] {"deptId"}, new string[] {"empName"});
var deptIndexConfig1 = new IndexConfig("deptIndex1", new string[] {"deptId"}, new string[] {"deptName"});
var deptIndexConfig2 = new IndexConfig("deptIndex2", new string[] {"location"}, new string[] {"deptName"});
```

Results in:

```
empIndexConfig: com.microsoft.hyperspace.index.IndexConfig = [indexName: empIndex; indexedColumns: deptid;
includedColumns: empname]
deptIndexConfig1: com.microsoft.hyperspace.index.IndexConfig = [indexName: deptIndex1; indexedColumns:
deptid; includedColumns: deptname]
deptIndexConfig2: com.microsoft.hyperspace.index.IndexConfig = [indexName: deptIndex2; indexedColumns:
location; includedColumns: deptname]
```

Now, you create three indexes using your index configurations. For this purpose, you invoke "createIndex" command on our Hyperspace instance. This command requires an index configuration and the dataFrame containing rows to be indexed. Running the following cell creates three indexes.

```
// Create indexes from configurations
import com.microsoft.hyperspace.index.Index

hyperspace.createIndex(empDF, empIndexConfig)
hyperspace.createIndex(deptDF, deptIndexConfig1)
hyperspace.createIndex(deptDF, deptIndexConfig2)
```

```
# Create indexes from configurations

hyperspace.createIndex(emp_DF, emp_IndexConfig)
hyperspace.createIndex(dept_DF, dept_IndexConfig1)
hyperspace.createIndex(dept_DF, dept_IndexConfig2)
```

```
// Create indexes from configurations
hyperspace.CreateIndex(empDF, empIndexConfig);
hyperspace.CreateIndex(deptDF, deptIndexConfig1);
hyperspace.CreateIndex(deptDF, deptIndexConfig2);
```

List indexes

The code that follows shows how you can list all available indexes in a Hyperspace instance. It uses "indexes" API that returns information about existing indexes as a Spark DataFrame so you can perform additional operations.

For instance, you can invoke valid operations on this DataFrame for checking its content or analyzing it further (for example filtering specific indexes or grouping them according to some desired property).

The following cell uses DataFrame's 'show' action to fully print the rows and show details of our indexes in a tabular form. For each index, you can see all information Hyperspace has stored about it in the metadata. You will immediately notice the following:

- config.indexName, config.indexedColumns, config.includedColumns, and status.status are the fields that a user normally refers to.
- dfSignature is automatically generated by Hyperspace and is unique for each index. Hyperspace uses this signature internally to maintain the index and exploit it at query time.

In the following output, all three indexes should have "ACTIVE" as status and their name, indexed columns, and included columns should match with what we defined in index configurations above.

```
hyperspace.indexes.show
```

```
hyperspace.indexes().show()
```

```
hyperspace.Indexes().Show();
```

Results in:

```
|Config.IndexName|Config.IndexedColumns|Config.IncludedColumns|        SchemaString|   SignatureProvider|
DfSignature|      SerializedPlan|NumBuckets|             DirPath|Status.Value|Stats.IndexSize|
|----------------|---------------------|----------------------|--------------------|--------------------|---
----------------|--------------------|----------|--------------------|------------|---------------|
|      deptIndex1|             [deptId]|            [deptName]|`deptId`
INT,`dep...|com.microsoft.cha...|0effc1610ae2e7c49...|Relation[deptId#3...|       200|abfss://datasets@...|
ACTIVE|              0|
|      deptIndex2|           [location]|            [deptName]|`location`
STRING...|com.microsoft.cha...|0effc1610ae2e7c49...|Relation[deptId#3...|       200|abfss://datasets@...|
ACTIVE|              0|
|        empIndex|             [deptId]|             [empName]|`deptId`
INT,`emp...|com.microsoft.cha...|30768c6c9b2533004...|Relation[empId#32...|       200|abfss://datasets@...|
ACTIVE|              0|
```

Delete indexes

You can drop an existing index by using the "deleteIndex" API and providing the index name. Index deletion does a soft delete: It mainly updates index's status in the Hyperspace metadata from "ACTIVE" to "DELETED". This will exclude the dropped index from any future query optimization and Hyperspace no longer picks that index for any query.

However, index files for a deleted index still remain available (since it is a soft-delete), so that the index could be restored if user asks for.

The following cell deletes index with name "deptIndex2" and lists Hyperspace metadata after that. The output should be similar to above cell for "List Indexes" except for "deptIndex2", which now should have its status changed into "DELETED".

```
hyperspace.deleteIndex("deptIndex2")

hyperspace.indexes.show
```

```
hyperspace.deleteIndex("deptIndex2")
hyperspace.indexes().show()
```

```
hyperspace.DeleteIndex("deptIndex2");

hyperspace.Indexes().Show();
```

Results in:

```
|Config.IndexName|Config.IndexedColumns|Config.IncludedColumns|        SchemaString|   SignatureProvider|
DfSignature|      SerializedPlan|NumBuckets|             DirPath|Status.Value|Stats.IndexSize|
|----------------|---------------------|----------------------|--------------------|--------------------|---
----------------|--------------------|----------|--------------------|------------|---------------|
|      deptIndex1|             [deptId]|            [deptName]|`deptId`
INT,`dep...|com.microsoft.cha...|0effc1610ae2e7c49...|Relation[deptId#3...|       200|abfss://datasets@...|
ACTIVE|              0|
|      deptIndex2|           [location]|            [deptName]|`location`
STRING...|com.microsoft.cha...|0effc1610ae2e7c49...|Relation[deptId#3...|       200|abfss://datasets@...|
DELETED|              0|
|        empIndex|             [deptId]|             [empName]|`deptId`
INT,`emp...|com.microsoft.cha...|30768c6c9b2533004...|Relation[empId#32...|       200|abfss://datasets@...|
ACTIVE|              0|
```

Restore indexes

You can use the "restoreIndex" API to restore a deleted index. This will bring back the latest version of index into ACTIVE status and makes it usable again for queries. The following cell shows an example of "restoreIndex" usage. You delete "deptIndex1" and restore it. The output shows "deptIndex1" first went into the "DELETED" status after invoking "deleteIndex" command and came back to the "ACTIVE" status after calling "restoreIndex".

```
hyperspace.deleteIndex("deptIndex1")

hyperspace.indexes.show

hyperspace.restoreIndex("deptIndex1")

hyperspace.indexes.show
```

```
hyperspace.deleteIndex("deptIndex1")
hyperspace.indexes().show()
hyperspace.restoreIndex("deptIndex1")
hyperspace.indexes().show()
```

```
hyperspace.DeleteIndex("deptIndex1");
hyperspace.Indexes().Show();
hyperspace.RestoreIndex("deptIndex1");
hyperspace.Indexes().Show();
```

Results in:

```
|Config.IndexName|Config.IndexedColumns|Config.IncludedColumns|        SchemaString|   SignatureProvider|
DfSignature|      SerializedPlan|NumBuckets|             DirPath|Status.Value|Stats.indexSize|
|----------------|--------------------|----------------------|--------------------|--------------------|---
-----------------|--------------------|----------|--------------------|------------|---------------|
|      deptIndex1|             [deptId]|            [deptName]|`deptId`
INT,`dep...|com.microsoft.cha...|0effc1610ae2e7c49...|Relation[deptId#3...|       200|abfss://datasets@...|
DELETED|              0|
|      deptIndex2|           [location]|            [deptName]|`location`
STRING...|com.microsoft.cha...|0effc1610ae2e7c49...|Relation[deptId#3...|       200|abfss://datasets@...|
DELETED|              0|
|        empIndex|             [deptId]|             [empName]|`deptId`
INT,`emp...|com.microsoft.cha...|30768c6c9b2533004...|Relation[empId#32...|       200|abfss://datasets@...|
ACTIVE|              0|
```

```
|Config.IndexName|Config.IndexedColumns|Config.IncludedColumns|        SchemaString|   SignatureProvider|
DfSignature|      SerializedPlan|NumBuckets|             DirPath|Status.value|Stats.IndexSize|
|----------------|--------------------|----------------------|--------------------|--------------------|---
-----------------|--------------------|----------|--------------------|------------|---------------|
|      deptIndex1|             [deptId]|            [deptName]|`deptId`
INT,`dep...|com.microsoft.cha...|0effc1610ae2e7c49...|Relation[deptId#3...|       200|abfss://datasets@...|
ACTIVE|              0|
|      deptIndex2|           [location]|            [deptName]|`location`
STRING...|com.microsoft.cha...|0effc1610ae2e7c49...|Relation[deptId#3...|       200|abfss://datasets@...|
DELETED|              0|
|        empIndex|             [deptId]|             [empName]|`deptId`
INT,`emp...|com.microsoft.cha...|30768c6c9b2533004...|Relation[empId#32...|       200|abfss://datasets@...|
ACTIVE|              0|
```

Vacuum indexes

You can perform a hard delete, that is, fully remove files and the metadata entry for a deleted index by using the **vacuumIndex** command. This action is irreversible. It physically deletes all the index files, which is why it's a hard delete.

The following cell vacuums the "deptIndex2" index and shows Hyperspace metadata after vacuuming. You should see metadata entries for two indexes "deptIndex1" and "empIndex" both with "ACTIVE" status and no entry for "deptIndex2".

```
hyperspace.vacuumIndex("deptIndex2")

hyperspace.indexes.show
```

```
hyperspace.vacuumIndex("deptIndex2")
hyperspace.indexes().show()
```

```
hyperspace.VacuumIndex("deptIndex2");
hyperspace.Indexes().Show();
```

Results in:

```
|Config.IndexName|Config.IndexedColumns|Config.IncludedColumns|        SchemaString|   SignatureProvider|
DfSignature|      SerializedPlan|NumBuckets|             DirPath|Status.Value|Stats.IndexSize|
|----------------|---------------------|----------------------|--------------------|--------------------|---
-----------------|--------------------|----------|--------------------|------------|---------------|
|      deptIndex1|             [deptId]|            [deptName]|`deptId`
INT,`dep...|com.microsoft.cha...|0effc1610ae2e7c49...|Relation[deptId#3...|       200|abfss://datasets@...|
ACTIVE|              0|
|        empIndex|             [deptId]|             [empName]|`deptId`
INT,`emp...|com.microsoft.cha...|30768c6c9b2533004...|Relation[empId#32...|       200|abfss://datasets@...|
ACTIVE|              0|
```

Enable or disable Hyperspace

Hyperspace provides APIs to enable or disable index usage with Spark.

- By using the **enableHyperspace** command, Hyperspace optimization rules become visible to the Spark optimizer and exploit existing Hyperspace indexes to optimize user queries.
- By using the **disableHyperspace** command, Hyperspace rules no longer apply during query optimization. Disabling Hyperspace has no impact on created indexes because they remain intact.

The following cell shows how you can use these commands to enable or disable Hyperspace. The output shows a reference to the existing Spark session whose configuration is updated.

```
// Enable Hyperspace
spark.enableHyperspace

// Disable Hyperspace
spark.disableHyperspace
```

```
# Enable Hyperspace
Hyperspace.enable(spark)

# Disable Hyperspace
Hyperspace.disable(spark)
```

```
// Enable Hyperspace
spark.EnableHyperspace();

// Disable Hyperspace
spark.DisableHyperspace();
```

Results in:

```
res48: org.apache.spark.sql.Spark™Session = org.apache.spark.sql.SparkSession@39fe1ddb
res51: org.apache.spark.sql.Spark™Session = org.apache.spark.sql.SparkSession@39fe1ddb
```

Index usage

To make Spark use Hyperspace indexes during query processing, you need to make sure that Hyperspace is enabled.

The following cell enables Hyperspace and creates two DataFrames containing your sample data records, which you use for running example queries. For each DataFrame, a few sample rows are printed.

```
// Enable Hyperspace
spark.enableHyperspace

val empDFrame: DataFrame = spark.read.parquet(empLocation)
val deptDFrame: DataFrame = spark.read.parquet(deptLocation)

empDFrame.show(5)
deptDFrame.show(5)
```

```
# Enable Hyperspace
Hyperspace.enable(spark)

emp_DF = spark.read.parquet(emp_Location)
dept_DF = spark.read.parquet(dept_Location)

emp_DF.show(5)
dept_DF.show(5)
```

```
// Enable Hyperspace
spark.EnableHyperspace();

DataFrame empDFrame = spark.Read().Parquet(empLocation);
DataFrame deptDFrame = spark.Read().Parquet(deptLocation);

empDFrame.Show(5);
deptDFrame.Show(5);
```

Results in:

```
res53: org.apache.spark.sql.Spark™Session = org.apache.spark.sql.Spark™Session@39fe1ddb
empDFrame: org.apache.spark.sql.DataFrame = [empId: int, empName: string ... 1 more field]
deptDFrame: org.apache.spark.sql.DataFrame = [deptId: int, deptName: string ... 1 more field]
```

```
|empId|empName|deptId|
|-----|-------|------|
| 7499|  ALLEN|    30|
| 7521|   WARD|    30|
| 7369|  SMITH|    20|
| 7844| TURNER|    30|
| 7876|  ADAMS|    20|
```

This only shows the top 5 rows

```
|deptId|  deptName|location|
|------|----------|--------|
|    10|Accounting|New York|
|    40|Operations|  Boston|
|    20|  Research|  Dallas|
|    30|     Sales| Chicago|
```

Index types

Currently, Hyperspace has rules to exploit indexes for two groups of queries:

- Selection queries with lookup or range selection filtering predicates.
- Join queries with an equality join predicate (that is, equijoins).

Indexes for accelerating filters

The first example query does a lookup on department records, as shown in the following cell. In SQL, this query looks like the following example:

```
SELECT deptName
FROM departments
WHERE deptId = 20
```

The output of running the following cell shows:

- Query result, which is a single department name.
- Query plan that Spark used to run the query.

In the query plan, the **FileScan** operator at the bottom of the plan shows the data source where the records were read from. The location of this file indicates the path to the latest version of the "deptIndex1" index. This information shows that according to the query and using Hyperspace optimization rules, Spark decided to exploit the proper index at runtime.

```
// Filter with equality predicate

val eqFilter: DataFrame = deptDFrame.filter("deptId = 20").select("deptName")
eqFilter.show()

eqFilter.explain(true)
```

```
# Filter with equality predicate

eqFilter = dept_DF.filter("""deptId = 20""").select("""deptName""")
eqFilter.show()

eqFilter.explain(True)
```

```
DataFrame eqFilter = deptDFrame.Filter("deptId = 20").Select("deptName");
eqFilter.Show();

eqFilter.Explain(true);
```

Results in:

```
eqFilter: org.apache.spark.sql.DataFrame = [deptName: string]
```

```
|DeptName|
|--------|
|Research|
```

```
== Parsed Logical Plan ==
'Project [unresolvedalias('deptName, None)]
+- Filter (deptId#533 = 20)
   +- Relation[deptId#533,deptName#534,location#535] parquet

== Analyzed Logical Plan ==
deptName: string
Project [deptName#534]
+- Filter (deptId#533 = 20)
   +- Relation[deptId#533,deptName#534,location#535] parquet

== Optimized Logical Plan ==
Project [deptName#534]
+- Filter (isnotnull(deptId#533) && (deptId#533 = 20))
   +- Relation[deptId#533,deptName#534,location#535] parquet

== Physical Plan ==
*(1) Project [deptName#534]
+- *(1) Filter (isnotnull(deptId#533) && (deptId#533 = 20))
   +- *(1) FileScan parquet [deptId#533,deptName#534] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspaceon...,
PartitionFilters: [], PushedFilters: [IsNotNull(deptId), EqualTo(deptId,20)], ReadSchema:
struct<deptId:int,deptName:string>
```

The second example is a range selection query on department records. In SQL, this query looks like the following example:

```
SELECT deptName
FROM departments
WHERE deptId > 20
```

Similar to the first example, the output of the following cell shows the query results (names of two departments) and the query plan. The location of the data file in the **FileScan** operator shows that "deptIndex1" was used to run the query.

```
// Filter with range selection predicate

val rangeFilter: DataFrame = deptDFrame.filter("deptId > 20").select("deptName")
rangeFilter.show()

rangeFilter.explain(true)
```

```
# Filter with range selection predicate

rangeFilter = dept_DF.filter("""deptId > 20""").select("deptName")
rangeFilter.show()

rangeFilter.explain(True)
```

```
// Filter with range selection predicate
DataFrame rangeFilter = deptDFrame.Filter("deptId > 20").Select("deptName");
rangeFilter.Show();

rangeFilter.Explain(true);
```

Results in:

```
rangeFilter: org.apache.spark.sql.DataFrame = [deptName: string]
```

```
|  DeptName|
|----------|
|Operations|
|     Sales|
```

```
== Parsed Logical Plan ==
'Project [unresolvedalias('deptName, None)]
+- Filter (deptId#533 > 20)
   +- Relation[deptId#533,deptName#534,location#535] parquet

== Analyzed Logical Plan ==
deptName: string
Project [deptName#534]
+- Filter (deptId#533 > 20)
   +- Relation[deptId#533,deptName#534,location#535] parquet

== Optimized Logical Plan ==
Project [deptName#534]
+- Filter (isnotnull(deptId#533) && (deptId#533 > 20))
   +- Relation[deptId#533,deptName#534,location#535] parquet

== Physical Plan ==
*(1) Project [deptName#534]
+- *(1) Filter (isnotnull(deptId#533) && (deptId#533 > 20))
   +- *(1) FileScan parquet [deptId#533,deptName#534] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspaceon...,
PartitionFilters: [], PushedFilters: [IsNotNull(deptId), GreaterThan(deptId,20)], ReadSchema:
struct<deptId:int,deptName:string>
```

The third example is a query joining department and employee records on the department ID. The equivalent SQL statement is shown as follows:

```
SELECT employees.deptId, empName, departments.deptId, deptName
FROM   employees, departments
WHERE  employees.deptId = departments.deptId
```

The output of running the following cell shows the query results, which are the names of 14 employees and the name of the department each employee works in. The query plan is also included in the output. Notice how the file locations for two **FileScan** operators show that Spark used "empIndex" and "deptIndex1" indexes to run the query.

```
// Join

val eqJoin: DataFrame =
      empDFrame.
      join(deptDFrame, empDFrame("deptId") === deptDFrame("deptId")).
      select(empDFrame("empName"), deptDFrame("deptName"))

eqJoin.show()

eqJoin.explain(true)
```

```
# Join

eqJoin = emp_DF.join(dept_DF, emp_DF.deptId == dept_DF.deptId).select(emp_DF.empName, dept_DF.deptName)

eqJoin.show()

eqJoin.explain(True)
```

```
// Join
DataFrame eqJoin =
      empDFrame
      .Join(deptDFrame, empDFrame.Col("deptId") == deptDFrame.Col("deptId"))
      .Select(empDFrame.Col("empName"), deptDFrame.Col("deptName"));

eqJoin.Show();

eqJoin.Explain(true);
```

Results in:

```
eqJoin: org.apache.spark.sql.DataFrame = [empName: string, deptName: string]
```

```
|empName|  deptName|
|-------|----------|
|  SMITH|  Research|
|  JONES|  Research|
|   FORD|  Research|
|  ADAMS|  Research|
|  SCOTT|  Research|
|   KING|Accounting|
|  CLARK|Accounting|
| MILLER|Accounting|
|  JAMES|     Sales|
|  BLAKE|     Sales|
| MARTIN|     Sales|
|  ALLEN|     Sales|
|   WARD|     Sales|
| TURNER|     Sales|
```

```
== Parsed Logical Plan ==
Project [empName#528, deptName#534]
+- Join Inner, (deptId#529 = deptId#533)
   :- Relation[empId#527,empName#528,deptId#529] parquet
   +- Relation[deptId#533,deptName#534,location#535] parquet

== Analyzed Logical Plan ==
empName: string, deptName: string
Project [empName#528, deptName#534]
+- Join Inner, (deptId#529 = deptId#533)
   :- Relation[empId#527,empName#528,deptId#529] parquet
   +- Relation[deptId#533,deptName#534,location#535] parquet

== Optimized Logical Plan ==
Project [empName#528, deptName#534]
+- Join Inner, (deptId#529 = deptId#533)
   :- Project [empName#528, deptId#529]
   :  +- Filter isnotnull(deptId#529)
   :     +- Relation[empName#528,deptId#529] parquet
   +- Project [deptId#533, deptName#534]
      +- Filter isnotnull(deptId#533)
         +- Relation[deptId#533,deptName#534] parquet

== Physical Plan ==
*(3) Project [empName#528, deptName#534]
+- *(3) SortMergeJoin [deptId#529], [deptId#533], Inner
   :- *(1) Project [empName#528, deptId#529]
   :  +- *(1) Filter isnotnull(deptId#529)
   :     +- *(1) FileScan parquet [deptId#529,empName#528] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspaceon...,
PartitionFilters: [], PushedFilters: [IsNotNull(deptId)], ReadSchema: struct<deptId:int,empName:string>,
SelectedBucketsCount: 200 out of 200
   +- *(2) Project [deptId#533, deptName#534]
      +- *(2) Filter isnotnull(deptId#533)
         +- *(2) FileScan parquet [deptId#533,deptName#534] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspaceon...,
PartitionFilters: [], PushedFilters: [IsNotNull(deptId)], ReadSchema: struct<deptId:int,deptName:string>,
SelectedBucketsCount: 200 out of 200
```

Support for SQL semantics

The index usage is transparent to whether you use the DataFrame API or Spark SQL. The following example shows the same join example as before, in SQL form, showing the use of indexes if applicable.

```
empDFrame.createOrReplaceTempView("EMP")
deptDFrame.createOrReplaceTempView("DEPT")

val joinQuery = spark.sql("SELECT EMP.empName, DEPT.deptName FROM EMP, DEPT WHERE EMP.deptId = DEPT.deptId")

joinQuery.show()
joinQuery.explain(true)
```

```
from pyspark.sql import SparkSession

emp_DF.createOrReplaceTempView("EMP")
dept_DF.createOrReplaceTempView("DEPT")

joinQuery = spark.sql("SELECT EMP.empName, DEPT.deptName FROM EMP, DEPT WHERE EMP.deptId = DEPT.deptId")

joinQuery.show()
joinQuery.explain(True)
```

```
empDFrame.CreateOrReplaceTempView("EMP");
deptDFrame.CreateOrReplaceTempView("DEPT");

var joinQuery = spark.Sql("SELECT EMP.empName, DEPT.deptName FROM EMP, DEPT WHERE EMP.deptId =
DEPT.deptId");

joinQuery.Show();
joinQuery.Explain(true);
```

Results in:

```
joinQuery: org.apache.spark.sql.DataFrame = [empName: string, deptName: string]
```

```
|empName|  deptName|
|-------|----------|
|  SMITH|  Research|
|  JONES|  Research|
|   FORD|  Research|
|  ADAMS|  Research|
|  SCOTT|  Research|
|   KING|Accounting|
|  CLARK|Accounting|
| MILLER|Accounting|
|  JAMES|     Sales|
|  BLAKE|     Sales|
| MARTIN|     Sales|
|  ALLEN|     Sales|
|   WARD|     Sales|
| TURNER|     Sales|
```

```
== Parsed Logical Plan ==
'Project ['EMP.empName, 'DEPT.deptName]
+- 'Filter ('EMP.deptId = 'DEPT.deptId)
   +- 'Join Inner
      :- 'UnresolvedRelation `EMP`
      +- 'UnresolvedRelation `DEPT`

== Analyzed Logical Plan ==
empName: string, deptName: string
Project [empName#528, deptName#534]
+- Filter (deptId#529 = deptId#533)
   +- Join Inner
      :- SubqueryAlias `emp`
      :  +- Relation[empId#527,empName#528,deptId#529] parquet
      +- SubqueryAlias `dept`
         +- Relation[deptId#533,deptName#534,location#535] parquet

== Optimized Logical Plan ==
Project [empName#528, deptName#534]
+- Join Inner, (deptId#529 = deptId#533)
   :- Project [empName#528, deptId#529]
   :  +- Filter isnotnull(deptId#529)
   :     +- Relation[empId#527,empName#528,deptId#529] parquet
   +- Project [deptId#533, deptName#534]
      +- Filter isnotnull(deptId#533)
         +- Relation[deptId#533,deptName#534,location#535] parquet

== Physical Plan ==
*(5) Project [empName#528, deptName#534]
+- *(5) SortMergeJoin [deptId#529], [deptId#533], Inner
   :- *(2) Sort [deptId#529 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(deptId#529, 200)
   :     +- *(1) Project [empName#528, deptId#529]
   :        +- *(1) Filter isnotnull(deptId#529)
   :           +- *(1) FileScan parquet [deptId#529,empName#528] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspaceon...,
PartitionFilters: [], PushedFilters: [IsNotNull(deptId)], ReadSchema: struct<deptId:int,empName:string>
   +- *(4) Sort [deptId#533 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(deptId#533, 200)
         +- *(3) Project [deptId#533, deptName#534]
            +- *(3) Filter isnotnull(deptId#533)
               +- *(3) FileScan parquet [deptId#533,deptName#534] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/your-path/departments.parquet],
PartitionFilters: [], PushedFilters: [IsNotNull(deptId)], ReadSchema: struct<deptId:int,deptName:string>
```

Explain API

Indexes are great, but how do you know if they're being used? Hyperspace allows users to compare their original plan versus the updated index-dependent plan before running their query. You have an option to choose from HTML, plaintext, or console mode to display the command output.

The following cell shows an example with HTML. The highlighted section represents the difference between original and updated plans along with the indexes being used.

```
spark.conf.set("spark.hyperspace.explain.displayMode", "html")
hyperspace.explain(eqJoin)(displayHTML(_))
```

```
eqJoin = emp_DF.join(dept_DF, emp_DF.deptId == dept_DF.deptId).select(emp_DF.empName, dept_DF.deptName)

spark.conf.set("spark.hyperspace.explain.displayMode", "html")
hyperspace.explain(eqJoin, True, displayHTML)
```

```
spark.Conf().Set("spark.hyperspace.explain.displayMode", "html");
spark.Conf().Set("spark.hyperspace.explain.displayMode.highlight.beginTag", "<b
style=\"background:LightGreen\">");
spark.Conf().Set("spark.hyperspace.explain.displayMode.highlight.endTag", "</b>");

hyperspace.Explain(eqJoin, false, input => DisplayHTML(input));
```

Results in:

Plan with indexes

```
Project [empName#528, deptName#534]
+- SortMergeJoin [deptId#529], [deptId#533], Inner
   :- *(1) Project [empName#528, deptId#529]
   :  +- *(1) Filter isnotnull(deptId#529)
   :     +- *(1) FileScan parquet [deptId#529,empName#528] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspaceon...,
PartitionFilters: [], PushedFilters: [IsNotNull(deptId)], ReadSchema: struct
   +- *(2) Project [deptId#533, deptName#534]
      +- *(2) Filter isnotnull(deptId#533)
         +- *(2) FileScan parquet [deptId#533,deptName#534] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspaceon...,
PartitionFilters: [], PushedFilters: [IsNotNull(deptId)], ReadSchema: struct
```

Plan without indexes

```
Project [empName#528, deptName#534]
+- SortMergeJoin [deptId#529], [deptId#533], Inner
   :- *(2) Sort [deptId#529 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(deptId#529, 200)
   :     +- *(1) Project [empName#528, deptId#529]
   :        +- *(1) Filter isnotnull(deptId#529)
   :           +- *(1) FileScan parquet [empName#528,deptId#529] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/your-path/employees.parquet],
PartitionFilters: [], PushedFilters: [IsNotNull(deptId)], ReadSchema: struct
   +- *(4) Sort [deptId#533 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(deptId#533, 200)
         +- *(3) Project [deptId#533, deptName#534]
            +- *(3) Filter isnotnull(deptId#533)
               +- *(3) FileScan parquet [deptId#533,deptName#534] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/your-path/departments.parquet],
PartitionFilters: [], PushedFilters: [IsNotNull(deptId)], ReadSchema: struct
```

Indexes used

```
deptIndex1:abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/<container>/indexes/public/deptIndex1/v
__=0
empIndex:abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/<container>/indexes/public/empIndex/v__=0
```

Refresh indexes

If the original data on which an index was created changes, the index will no longer capture the latest state of data. You can refresh a stale index by using the **refreshIndex** command. This command causes the index to be fully rebuilt and updates it according to the latest data records. We'll show you how to incrementally refresh your index in other notebooks.

The following two cells show an example for this scenario:

- The first cell adds two more departments to the original departments data. It reads and prints a list of departments to verify new departments are added correctly. The output shows six departments in total: four old ones and two new. Invoking **refreshIndex** updates "deptIndex1" so that the index captures new departments.
- The second cell runs our range selection query example. The results should now contain four departments: two are the ones seen before when we ran the preceding query, and two are the new departments we added.

Specific index refresh

```
val extraDepartments = Seq(
      (50, "Inovation", "Seattle"),
      (60, "Human Resources", "San Francisco"))

val extraDeptData: DataFrame = extraDepartments.toDF("deptId", "deptName", "location")
extraDeptData.write.mode("Append").parquet(deptLocation)

val deptDFrameUpdated: DataFrame = spark.read.parquet(deptLocation)

deptDFrameUpdated.show(10)

hyperspace.refreshIndex("deptIndex1")
```

```


extra_Departments = [(50, "Inovation", "Seattle"), (60, "Human Resources", "San Francisco")]

extra_departments_df = spark.createDataFrame(extra_Departments, dept_schema)
extra_departments_df.write.mode("Append").parquet(dept_Location)


dept_DFrame_Updated = spark.read.parquet(dept_Location)

dept_DFrame_Updated.show(10)
```

```


var extraDepartments = new List<GenericRow>()
{
    new GenericRow(new object[] {50, "Inovation", "Seattle"}),
    new GenericRow(new object[] {60, "Human Resources", "San Francisco"})
};

DataFrame extraDeptData = spark.CreateDataFrame(extraDepartments, departmentSchema);
extraDeptData.Write().Mode("Append").Parquet(deptLocation);

DataFrame deptDFrameUpdated = spark.Read().Parquet(deptLocation);

deptDFrameUpdated.Show(10);

hyperspace.RefreshIndex("deptIndex1");
```

Results in:

```
extraDepartments: Seq[(Int, String, String)] = List((50,Inovation,Seattle), (60,Human Resources,San
Francisco))
extraDeptData: org.apache.spark.sql.DataFrame = [deptId: int, deptName: string ... 1 more field]
deptDFrameUpdated: org.apache.spark.sql.DataFrame = [deptId: int, deptName: string ... 1 more field]
```

```
|deptId|       deptName|     location|
|------|---------------|-------------|
|    60|Human Resources|San Francisco|
|    10|     Accounting|     New York|
|    50|      Inovation|      Seattle|
|    40|     Operations|       Boston|
|    20|       Research|       Dallas|
|    30|          Sales|      Chicago|
```

Range selection

```
val newRangeFilter: DataFrame = deptDFrameUpdated.filter("deptId > 20").select("deptName")
newRangeFilter.show()

newRangeFilter.explain(true)
```

```
newRangeFilter = dept_DFrame_Updated.filter("deptId > 20").select("deptName")
newRangeFilter.show()

newRangeFilter.explain(True)
```

```
DataFrame newRangeFilter = deptDFrameUpdated.Filter("deptId > 20").Select("deptName");
newRangeFilter.Show();

newRangeFilter.Explain(true);
```

Results in:

```
newRangeFilter: org.apache.spark.sql.DataFrame = [deptName: string]
```

```
|       DeptName|
|---------------|
|Human Resources|
|      Inovation|
|     Operations|
|          Sales|
```

```
== Parsed Logical Plan ==
'Project [unresolvedalias('deptName, None)]
+- Filter (deptId#674 > 20)
   +- Relation[deptId#674,deptName#675,location#676] parquet

== Analyzed Logical Plan ==
deptName: string
Project [deptName#675]
+- Filter (deptId#674 > 20)
   +- Relation[deptId#674,deptName#675,location#676] parquet

== Optimized Logical Plan ==
Project [deptName#675]
+- Filter (isnotnull(deptId#674) && (deptId#674 > 20))
   +- Relation[deptId#674,deptName#675,location#676] parquet

== Physical Plan ==
*(1) Project [deptName#675]
+- *(1) Filter (isnotnull(deptId#674) && (deptId#674 > 20))
   +- *(1) FileScan parquet [deptId#674,deptName#675] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspaceon...,
PartitionFilters: [], PushedFilters: [IsNotNull(deptId), GreaterThan(deptId,20)], ReadSchema:
struct<deptId:int,deptName:string>
```

Hybrid scan for mutable datasets

Often, if your underlying source data had some new files appended or existing files deleted, your index will get stale and Hyperspace decides not to use it. However, there are times when you just want to use the index without having to refresh it every time. There could be multiple reasons for doing so:

- You don't want to continuously refresh your index, but instead want to do it periodically because you understand your workloads the best.
- You added/removed only a few files and don't want to wait for yet another refresh job to finish.

To allow you to still use a stale index, Hyperspace introduces hybrid scan, a novel technique that allows users to utilize outdated or stale indexes (for example, the underlying source data had some new files appended or existing files deleted) without refreshing indexes.

To achieve this, when you set the appropriate configuration to enable hybrid scan, Hyperspace modifies the query plan to leverage the changes as following:

- Appended files can be merged to index data by using Union or BucketUnion (for join). Shuffling appended data can also be applied before merging, if needed.
- Deleted files can be handled by injecting Filter-NOT-IN condition on lineage column of index data, so that the indexed rows from the deleted files can be excluded at query time.

You can check the transformation of the query plan in following examples.

> **NOTE**
> Currently, hybrid scan is supported only for non-partitioned data.

Hybrid scan for appended files - non-partitioned data

Non-partitioned data is used in the following example. In this example, we expect that the Join index can be used for the query and BucketUnion is introduced for appended files.

```
val testData = Seq(
    ("orange", 3, "2020-10-01"),
    ("banana", 1, "2020-10-01"),
    ("carrot", 5, "2020-10-02"),
    ("beetroot", 12, "2020-10-02"),
    ("orange", 2, "2020-10-03"),
    ("banana", 11, "2020-10-03"),
    ("carrot", 3, "2020-10-03"),
    ("beetroot", 2, "2020-10-04"),
    ("cucumber", 7, "2020-10-05"),
    ("pepper", 20, "2020-10-06")
    ).toDF("name", "qty", "date")

val testDataLocation = s"$dataPath/productTable"

testData.write.mode("overwrite").parquet(testDataLocation)
val testDF = spark.read.parquet(testDataLocation)
```

```
testdata = [
    ("orange", 3, "2020-10-01"),
    ("banana", 1, "2020-10-01"),
    ("carrot", 5, "2020-10-02"),
    ("beetroot", 12, "2020-10-02"),
    ("orange", 2, "2020-10-03"),
    ("banana", 11, "2020-10-03"),
    ("carrot", 3, "2020-10-03"),
    ("beetroot", 2, "2020-10-04"),
    ("cucumber", 7, "2020-10-05"),
    ("pepper", 20, "2020-10-06")
]

testdata_location = data_path + "/productTable"
from pyspark.sql.types import StructField, StructType, StringType, IntegerType
testdata_schema = StructType([
    StructField('name', StringType(), True),
    StructField('qty', IntegerType(), True),
    StructField('date', StringType(), True)])

test_df = spark.createDataFrame(testdata, testdata_schema)
test_df.write.mode("overwrite").parquet(testdata_location)
test_df = spark.read.parquet(testdata_location)
```

```
using Microsoft.Spark.Sql.Types;

var products = new List<GenericRow>() {
    new GenericRow(new object[] {"orange", 3, "2020-10-01"}),
    new GenericRow(new object[] {"banana", 1, "2020-10-01"}),
    new GenericRow(new object[] {"carrot", 5, "2020-10-02"}),
    new GenericRow(new object[] {"beetroot", 12, "2020-10-02"}),
    new GenericRow(new object[] {"orange", 2, "2020-10-03"}),
    new GenericRow(new object[] {"banana", 11, "2020-10-03"}),
    new GenericRow(new object[] {"carrot", 3, "2020-10-03"}),
    new GenericRow(new object[] {"beetroot", 2, "2020-10-04"}),
    new GenericRow(new object[] {"cucumber", 7, "2020-10-05"}),
    new GenericRow(new object[] {"pepper", 20, "2020-10-06"})
};
var productsSchema = new StructType(new List<StructField>()
{
    new StructField("name", new StringType()),
    new StructField("qty", new IntegerType()),
    new StructField("date", new StringType())
});

DataFrame testData = spark.CreateDataFrame(products, productsSchema);
string testDataLocation = $"{dataPath}/productTable";
testData.Write().Mode("overwrite").Parquet(testDataLocation);
```

```
// CREATE INDEX
hyperspace.createIndex(testDF, IndexConfig("productIndex2", Seq("name"), Seq("date", "qty")))

spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)
val filter1 = testDF.filter("name = 'banana'")
val filter2 = testDF.filter("qty > 10")
val query = filter1.join(filter2, "name")

// Check Join index rule is applied properly.
hyperspace.explain(query)(displayHTML(_))
```

```
# CREATE INDEX
hyperspace.createIndex(test_df, IndexConfig("productIndex2", ["name"], ["date", "qty"]))

spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)
filter1 = test_df.filter("name = 'banana'")
filter2 = test_df.filter("qty > 10")
query = filter1.join(filter2, "name")

# Check Join index rule is applied properly.
hyperspace.explain(query, True, displayHTML)
```

```
// CREATE INDEX
DataFrame testDF = spark.Read().Parquet(testDataLocation);
var productIndex2Config = new IndexConfig("productIndex", new string[] {"name"}, new string[] {"date",
"qty"});
hyperspace.CreateIndex(testDF, productIndex2Config);

// Check Join index rule is applied properly.
DataFrame filter1 = testDF.Filter("name = 'banana'");
DataFrame filter2 = testDF.Filter("qty > 10");
DataFrame query = filter1.Join(filter2, filter1.Col("name") == filter2.Col("name"));

query.Show();

hyperspace.Explain(query, true, input => DisplayHTML(input));
```

Result in:

```
=============================================================
Plan with indexes:
=============================================================
Project [name#607, qty#608, date#609, qty#632, date#633]
+- SortMergeJoin [name#607], [name#631], Inner
   :- *(1) Project [name#607, qty#608, date#609]
   :  +- *(1) Filter (isnotnull(name#607) && (name#607 = banana))
   :     +- *(1) FileScan parquet [name#607,date#609,qty#608] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-777519/p...,
PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema: struct,
SelectedBucketsCount: 1 out of 200
   +- *(2) Project [name#631, qty#632, date#633]
      +- *(2) Filter (((isnotnull(qty#632) && (qty#632 > 10)) && isnotnull(name#631)) && (name#631 =
banana))
         +- *(2) FileScan parquet [name#631,date#633,qty#632] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-777519/p...,
PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10), IsNotNull(name),
EqualTo(name,banana)], ReadSchema: struct, SelectedBucketsCount: 1 out of 200

=============================================================
Plan without indexes:
=============================================================
Project [name#607, qty#608, date#609, qty#632, date#633]
+- SortMergeJoin [name#607], [name#631], Inner
   :- *(2) Sort [name#607 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(name#607, 200), [id=#453]
   :     +- *(1) Project [name#607, qty#608, date#609]
   :        +- *(1) Filter (isnotnull(name#607) && (name#607 = banana))
   :           +- *(1) FileScan parquet [name#607,qty#608,date#609] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema:
struct
   +- *(4) Sort [name#631 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(name#631, 200), [id=#459]
         +- *(3) Project [name#631, qty#632, date#633]
            +- *(3) Filter (((isnotnull(qty#632) && (qty#632 > 10)) && isnotnull(name#631)) && (name#631 =
banana))
               +- *(3) FileScan parquet [name#631,qty#632,date#633] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10), IsNotNull(name),
EqualTo(name,banana)], ReadSchema: struct

=============================================================
Indexes used:
=============================================================
productIndex2:abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-
777519/productIndex2/v__=0
```

```
// Append new files.
val appendData = Seq(
    ("orange", 13, "2020-11-01"),
    ("banana", 5, "2020-11-01")).toDF("name", "qty", "date")
appendData.write.mode("append").parquet(testDataLocation)
```

```
# Append new files.
append_data = [
    ("orange", 13, "2020-11-01"),
    ("banana", 5, "2020-11-01")
]
append_df = spark.createDataFrame(append_data, testdata_schema)
append_df.write.mode("append").parquet(testdata_location)
```

```
// Append new files.
var appendProducts = new List<GenericRow>()
{
    new GenericRow(new object[] {"orange", 13, "2020-11-01"}),
    new GenericRow(new object[] {"banana", 5, "2020-11-01"})
};

DataFrame appendData = spark.CreateDataFrame(appendProducts, productsSchema);
appendData.Write().Mode("Append").Parquet(testDataLocation);
```

Hybrid scan is disabled by default. Therefore, you will see that because we appended new data, Hyperspace will decide *not* to use the index.

In the output, you will see no plan differences (hence, no highlighting).

```
// Hybrid Scan configs are false by default.
spark.conf.set("spark.hyperspace.index.hybridscan.enabled", "false")
spark.conf.set("spark.hyperspace.index.hybridscan.delete.enabled", "false")

val testDFWithAppend = spark.read.parquet(testDataLocation)
val filter1 = testDFWithAppend.filter("name = 'banana'")
val filter2 = testDFWithAppend.filter("qty > 10")
val query = filter1.join(filter2, "name")
hyperspace.explain(query)(displayHTML(_))
query.show
```

```
# Hybrid Scan configs are false by default.
spark.conf.set("spark.hyperspace.index.hybridscan.enabled", "false")
spark.conf.set("spark.hyperspace.index.hybridscan.delete.enabled", "false")

test_df_with_append = spark.read.parquet(testdata_location)
filter1 = test_df_with_append.filter("name = 'banana'")
filter2 = test_df_with_append.filter("qty > 10")
query = filter1.join(filter2, "name")
hyperspace.explain(query, True, displayHTML)
query.show()
```

```
// Hybrid Scan configs are false by default.
spark.Conf().Set("spark.hyperspace.index.hybridscan.enabled", "false");
spark.Conf().Set("spark.hyperspace.index.hybridscan.delete.enabled", "false");

DataFrame testDFWithAppend = spark.Read().Parquet(testDataLocation);
DataFrame filter1 = testDFWithAppend.Filter("name = 'banana'");
DataFrame filter2 = testDFWithAppend.Filter("qty > 10");
DataFrame query = filter1.Join(filter2, filter1.Col("name") == filter2.Col("name"));

query.Show();

hyperspace.Explain(query, true, input => DisplayHTML(input));
```

Result in:

```
=============================================================
Plan with indexes:
=============================================================
Project [name#678, qty#679, date#680, qty#685, date#686]
+- SortMergeJoin [name#678], [name#684], Inner
   :- *(2) Sort [name#678 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(name#678, 200), [id=#589]
   :     +- *(1) Project [name#678, qty#679, date#680]
   :        +- *(1) Filter (isnotnull(name#678) && (name#678 = banana))
   :           +- *(1) FileScan parquet [name#678,qty#679,date#680] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema:
struct
   +- *(4) Sort [name#684 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(name#684, 200), [id=#595]
         +- *(3) Project [name#684, qty#685, date#686]
            +- *(3) Filter (((isnotnull(qty#685) && (qty#685 > 10)) && (name#684 = banana)) &&
isnotnull(name#684))
               +- *(3) FileScan parquet [name#684,qty#685,date#686] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10),
EqualTo(name,banana), IsNotNull(name)], ReadSchema: struct

=============================================================
Plan without indexes:
=============================================================
Project [name#678, qty#679, date#680, qty#685, date#686]
+- SortMergeJoin [name#678], [name#684], Inner
   :- *(2) Sort [name#678 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(name#678, 200), [id=#536]
   :     +- *(1) Project [name#678, qty#679, date#680]
   :        +- *(1) Filter (isnotnull(name#678) && (name#678 = banana))
   :           +- *(1) FileScan parquet [name#678,qty#679,date#680] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema:
struct
   +- *(4) Sort [name#684 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(name#684, 200), [id=#542]
         +- *(3) Project [name#684, qty#685, date#686]
            +- *(3) Filter (((isnotnull(qty#685) && (qty#685 > 10)) && (name#684 = banana)) &&
isnotnull(name#684))
               +- *(3) FileScan parquet [name#684,qty#685,date#686] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10),
EqualTo(name,banana), IsNotNull(name)], ReadSchema: struct

+------+---+----------+---+----------+
|  name|qty|      date|qty|      date|
+------+---+----------+---+----------+
|banana| 11|2020-10-03| 11|2020-10-03|
|banana|  5|2020-11-01| 11|2020-10-03|
|banana|  1|2020-10-01| 11|2020-10-03|
+------+---+----------+---+----------
```

Enable hybrid scan

In plan with indexes, you can see Exchange hash partitioning required only for appended files so that we could still utilize the "shuffled" index data with appended files. BucketUnion is used to merge "shuffled" appended files with the index data.

```
// Enable Hybrid Scan config. "delete" config is not necessary since we only appended data.
spark.conf.set("spark.hyperspace.index.hybridscan.enabled", "true")
spark.enableHyperspace
// Need to redefine query to recalculate the query plan.
val query = filter1.join(filter2, "name")
hyperspace.explain(query)(displayHTML(_))
query.show
```

```
# Enable Hybrid Scan config. "delete" config is not necessary.
spark.conf.set("spark.hyperspace.index.hybridscan.enabled", "true")

# Need to redefine query to recalculate the query plan.
query = filter1.join(filter2, "name")
hyperspace.explain(query, True, displayHTML)
query.show()
```

```
// Enable Hybrid Scan config. "delete" config is not necessary.
spark.Conf().Set("spark.hyperspace.index.hybridscan.enabled", "true");
spark.EnableHyperspace();
// Need to redefine query to recalculate the query plan.
DataFrame query = filter1.Join(filter2, filter1.Col("name") == filter2.Col("name"));

query.Show();

hyperspace.Explain(query, true, input => DisplayHTML(input));
```

Result in:

```
=============================================================
Plan with indexes:
=============================================================
Project [name#678, qty#679, date#680, qty#732, date#733]
+- SortMergeJoin [name#678], [name#731], Inner
   :- *(3) Sort [name#678 ASC NULLS FIRST], false, 0
   :  +- BucketUnion 200 buckets, bucket columns: [name]
   :     :- *(1) Project [name#678, qty#679, date#680]
   :     :  +- *(1) Filter (isnotnull(name#678) && (name#678 = banana))
   :     :     +- *(1) FileScan parquet [name#678,date#680,qty#679] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-
777519/p..., PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema:
struct, SelectedBucketsCount: 1 out of 200
   :     +- Exchange hashpartitioning(name#678, 200), [id=#775]
   :        +- *(2) Project [name#678, qty#679, date#680]
   :           +- *(2) Filter (isnotnull(name#678) && (name#678 = banana))
   :              +- *(2) FileScan parquet [name#678,date#680,qty#679] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema:
struct
   +- *(6) Sort [name#731 ASC NULLS FIRST], false, 0
      +- BucketUnion 200 buckets, bucket columns: [name]
         :- *(4) Project [name#731, qty#732, date#733]
         :  +- *(4) Filter (((isnotnull(qty#732) && (qty#732 > 10)) && isnotnull(name#731)) && (name#731 =
banana))
         :     +- *(4) FileScan parquet [name#731,date#733,qty#732] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-
777519/p..., PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10), IsNotNull(name),
EqualTo(name,banana)], ReadSchema: struct, SelectedBucketsCount: 1 out of 200
         +- Exchange hashpartitioning(name#731, 200), [id=#783]
            +- *(5) Project [name#731, qty#732, date#733]
               +- *(5) Filter (((isnotnull(qty#732) && (qty#732 > 10)) && isnotnull(name#731)) && (name#731
= banana))
                  +- *(5) FileScan parquet [name#731,date#733,qty#732] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
```

```
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10), IsNotNull(name),
EqualTo(name,banana)], ReadSchema: struct

=============================================================
Plan without indexes:
=============================================================
Project [name#678, qty#679, date#680, qty#732, date#733]
+- SortMergeJoin [name#678], [name#731], Inner
   :- *(2) Sort [name#678 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(name#678, 200), [id=#701]
   :     +- *(1) Project [name#678, qty#679, date#680]
   :        +- *(1) Filter (isnotnull(name#678) && (name#678 = banana))
   :           +- *(1) FileScan parquet [name#678,qty#679,date#680] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema:
struct
   +- *(4) Sort [name#731 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(name#731, 200), [id=#707]
         +- *(3) Project [name#731, qty#732, date#733]
            +- *(3) Filter (((isnotnull(qty#732) && (qty#732 > 10)) && isnotnull(name#731)) && (name#731 =
banana))
               +- *(3) FileScan parquet [name#731,qty#732,date#733] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10), IsNotNull(name),
EqualTo(name,banana)], ReadSchema: struct

=============================================================
Indexes used:
=============================================================
productIndex2:abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-
777519/productIndex2/v__=0


+------+---+----------+---+----------+
|  name|qty|      date|qty|      date|
+------+---+----------+---+----------+
|banana|  1|2020-10-01| 11|2020-10-03|
|banana| 11|2020-10-03| 11|2020-10-03|
|banana|  5|2020-11-01| 11|2020-10-03|
+------+---+----------+---+----------+
```

Incremental index refresh

When you're ready to update your indexes but don't want to rebuild your entire index, Hyperspace supports updating indexes in an incremental manner using the `hs.refreshIndex("name", "incremental")` API. This will eliminates the need for a full rebuild of index from scratch, utilizing previously created index files as well as updating indexes on only the newly added data.

Of course, be sure to use the complementary `optimizeIndex` API (shown below) periodically to make sure you do not see performance regressions. We recommend calling optimize at least once for every 10 times you call `refreshIndex(..., "incremental")`, assuming the data you added/removed is < 10% of the original dataset. For instance, if your original dataset is 100 GB, and you've added/removed data in increments/decrements of 1 GB, you can call `refreshIndex` 10 times before calling `optimizeIndex`. Please note that this example is simply used for illustration and you have to adapt this for your workloads.

In the example below, notice the addition of a Sort node in the query plan when indexes are used. This is because partial indexes are created on the appended data files, causing Spark to introduce a `Sort`. Please also note that `Shuffle` i.e. Exchange is still eliminated from the plan, giving you the appropriate acceleration.

```
def query(): DataFrame = {
    val testDFWithAppend = spark.read.parquet(testDataLocation)
    val filter1 = testDFWithAppend.filter("name = 'banana'")
    val filter2 = testDFWithAppend.filter("qty > 10")
    filter1.join(filter2, "name")
}

hyperspace.refreshIndex("productIndex2", "incremental")

hyperspace.explain(query())(displayHTML(_))
query().show
```

```
def query():
    test_df_with_append = spark.read.parquet(testdata_location)
    filter1 = test_df_with_append.filter("name = 'banana'")
    filter2 = test_df_with_append.filter("qty > 10")
    return filter1.join(filter2, "name")

hyperspace.refreshIndex("productIndex2", "incremental")

hyperspace.explain(query(), True, displayHTML)
query().show()
```

Result in:

```
=============================================================
Plan with indexes:
=============================================================
Project [name#820, qty#821, date#822, qty#827, date#828]
+- SortMergeJoin [name#820], [name#826], Inner
   :- *(1) Sort [name#820 ASC NULLS FIRST], false, 0
   :  +- *(1) Project [name#820, qty#821, date#822]
   :     +- *(1) Filter (isnotnull(name#820) && (name#820 = banana))
   :        +- *(1) FileScan parquet [name#820,date#822,qty#821] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-777519/p...,
PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema: struct,
SelectedBucketsCount: 1 out of 200
   +- *(2) Sort [name#826 ASC NULLS FIRST], false, 0
      +- *(2) Project [name#826, qty#827, date#828]
         +- *(2) Filter (((isnotnull(qty#827) && (qty#827 > 10)) && (name#826 = banana)) &&
isnotnull(name#826))
            +- *(2) FileScan parquet [name#826,date#828,qty#827] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-777519/p...,
PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10), EqualTo(name,banana),
IsNotNull(name)], ReadSchema: struct, SelectedBucketsCount: 1 out of 200

=============================================================
Plan without indexes:
=============================================================
Project [name#820, qty#821, date#822, qty#827, date#828]
+- SortMergeJoin [name#820], [name#826], Inner
   :- *(2) Sort [name#820 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(name#820, 200), [id=#927]
   :     +- *(1) Project [name#820, qty#821, date#822]
   :        +- *(1) Filter (isnotnull(name#820) && (name#820 = banana))
   :           +- *(1) FileScan parquet [name#820,qty#821,date#822] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema:
struct
   +- *(4) Sort [name#826 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(name#826, 200), [id=#933]
         +- *(3) Project [name#826, qty#827, date#828]
            +- *(3) Filter (((isnotnull(qty#827) && (qty#827 > 10)) && (name#826 = banana)) &&
isnotnull(name#826))
               +- *(3) FileScan parquet [name#826,qty#827,date#828] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10),
EqualTo(name,banana), IsNotNull(name)], ReadSchema: struct

+------+---+----------+---+----------+
|  name|qty|      date|qty|      date|
+------+---+----------+---+----------+
|banana|  1|2020-10-01| 11|2020-10-03|
|banana| 11|2020-10-03| 11|2020-10-03|
|banana|  5|2020-11-01| 11|2020-10-03|
+------+---+----------+---+----------+
```

Optimize index layout

After calling incremental refreshes multiple times on newly appended data (e.g. if the user writes to data in small batches or in case of streaming scenarios), the number of index files tend to become large affecting the performance of the index (large number of small files problem). Hyperspace provides `hyperspace.optimizeIndex("indexName")` API to optimize the index layout and reduce the large files problem.

In the plan below, notice that Hyperspace has removed the additional Sort node in the query plan. Optimize can help avoiding sorting for any index bucket which contains only one file. However, this will only be true if ALL the index buckets have at most 1 file per bucket, after `optimizeIndex`.

```
// Append some more data and call refresh again.
val appendData = Seq(
    ("orange", 13, "2020-11-01"),
    ("banana", 5, "2020-11-01")).toDF("name", "qty", "date")
appendData.write.mode("append").parquet(testDataLocation)

hyperspace.refreshIndex("productIndex2", "incremental")
```

```
# Append some more data and call refresh again.
append_data = [
    ("orange", 13, "2020-11-01"),
    ("banana", 5, "2020-11-01")
]
append_df = spark.createDataFrame(append_data, testdata_schema)
append_df.write.mode("append").parquet(testdata_location)

hyperspace.refreshIndex("productIndex2", "incremental"
```

```
// Call optimize. Ensure that Sort is removed after optimization (This is possible here because after
optimize, in this case, every bucket contains only 1 file.).
hyperspace.optimizeIndex("productIndex2")

hyperspace.explain(query())(displayHTML(_))
```

```
# Call optimize. Ensure that Sort is removed after optimization (This is possible here because after
optimize, in this case, every bucket contains only 1 file.).
hyperspace.optimizeIndex("productIndex2")

hyperspace.explain(query(), True, displayHTML)
```

Result in:

```
=============================================================
Plan with indexes:
=============================================================
Project [name#954, qty#955, date#956, qty#961, date#962]
+- SortMergeJoin [name#954], [name#960], Inner
   :- *(1) Project [name#954, qty#955, date#956]
   :  +- *(1) Filter (isnotnull(name#954) && (name#954 = banana))
   :     +- *(1) FileScan parquet [name#954,date#956,qty#955] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-777519/p...,
PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema: struct,
SelectedBucketsCount: 1 out of 200
   +- *(2) Project [name#960, qty#961, date#962]
      +- *(2) Filter (((isnotnull(qty#961) && (qty#961 > 10)) && isnotnull(name#960)) && (name#960 =
banana))
         +- *(2) FileScan parquet [name#960,date#962,qty#961] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-777519/p...,
PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10), IsNotNull(name),
EqualTo(name,banana)], ReadSchema: struct, SelectedBucketsCount: 1 out of 200

=============================================================
Plan without indexes:
=============================================================
Project [name#954, qty#955, date#956, qty#961, date#962]
+- SortMergeJoin [name#954], [name#960], Inner
   :- *(2) Sort [name#954 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(name#954, 200), [id=#1070]
   :     +- *(1) Project [name#954, qty#955, date#956]
   :        +- *(1) Filter (isnotnull(name#954) && (name#954 = banana))
   :           +- *(1) FileScan parquet [name#954,qty#955,date#956] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema:
struct
   +- *(4) Sort [name#960 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(name#960, 200), [id=#1076]
         +- *(3) Project [name#960, qty#961, date#962]
            +- *(3) Filter (((isnotnull(qty#961) && (qty#961 > 10)) && isnotnull(name#960)) && (name#960 =
banana))
               +- *(3) FileScan parquet [name#960,qty#961,date#962] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10), IsNotNull(name),
EqualTo(name,banana)], ReadSchema: struct

=============================================================
Indexes used:
=============================================================
productIndex2:abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-
777519/productIndex2/v__=3
```

Optimize modes

The default mode for optimization is "quick" mode where files smaller than a predefined threshold are picked for optimization. To maximize the effect of optimization, Hyperspace allows another optimize mode "full" as shown below. This mode picks ALL index files for optimization irrespective of their file size and creates the best possible layout of the index. This is also slower than the default optimize mode because more data is being processed here.

```
hyperspace.optimizeIndex("productIndex2", "full")

hyperspace.explain(query())(displayHTML(_))
```

```
hyperspace.optimizeIndex("productIndex2", "full")

hyperspace.explain(query(), True, displayHTML)
```

Result in:

```
=============================================================
Plan with indexes:
=============================================================
Project [name#1000, qty#1001, date#1002, qty#1007, date#1008]
+- SortMergeJoin [name#1000], [name#1006], Inner
   :- *(1) Project [name#1000, qty#1001, date#1002]
   :  +- *(1) Filter (isnotnull(name#1000) && (name#1000 = banana))
   :     +- *(1) FileScan parquet [name#1000,date#1002,qty#1001] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-777519/p...,
PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema: struct,
SelectedBucketsCount: 1 out of 200
   +- *(2) Project [name#1006, qty#1007, date#1008]
      +- *(2) Filter (((isnotnull(qty#1007) && (qty#1007 > 10)) && isnotnull(name#1006)) && (name#1006 =
banana))
         +- *(2) FileScan parquet [name#1006,date#1008,qty#1007] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-777519/p...,
PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10), IsNotNull(name),
EqualTo(name,banana)], ReadSchema: struct, SelectedBucketsCount: 1 out of 200

=============================================================
Plan without indexes:
=============================================================
Project [name#1000, qty#1001, date#1002, qty#1007, date#1008]
+- SortMergeJoin [name#1000], [name#1006], Inner
   :- *(2) Sort [name#1000 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(name#1000, 200), [id=#1160]
   :     +- *(1) Project [name#1000, qty#1001, date#1002]
   :        +- *(1) Filter (isnotnull(name#1000) && (name#1000 = banana))
   :           +- *(1) FileScan parquet [name#1000,qty#1001,date#1002] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,banana)], ReadSchema:
struct
   +- *(4) Sort [name#1006 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(name#1006, 200), [id=#1166]
         +- *(3) Project [name#1006, qty#1007, date#1008]
            +- *(3) Filter (((isnotnull(qty#1007) && (qty#1007 > 10)) && isnotnull(name#1006)) && (name#1006
= banana))
               +- *(3) FileScan parquet [name#1006,qty#1007,date#1008] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/data-
777519/prod..., PartitionFilters: [], PushedFilters: [IsNotNull(qty), GreaterThan(qty,10), IsNotNull(name),
EqualTo(name,banana)], ReadSchema: struct

=============================================================
Indexes used:
=============================================================
productIndex2:abfss://datasets@hyperspacebenchmark.dfs.core.windows.net/hyperspace/indexes-
777519/productIndex2/v__=4
```

Next steps

- Project Hyperspace
- Azure Synapse Analytics

Apache Spark GPU-accelerated pools in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

Apache Spark is a parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. Apache Spark in Azure Synapse Analytics is one of Microsoft's implementations of Apache Spark in the cloud.

Azure Synapse now offers the ability to create Azure Synapse GPU-enabled pools to run Spark workloads using underlying RAPIDS libraries that use the massive parallel processing power of GPUs to accelerate processing. The RAPIDS Accelerator for Apache Spark allows you to run your existing Spark applications **without any code change** by just enabling a configuration setting, which comes pre-configured for a GPU-enabled pool. You can choose to turn on/off the RAPIDS-based GPU acceleration for your workload or parts of your workload by setting this configuration:

```
spark.conf.set('spark.rapids.sql.enabled','true/false')
```

NOTE

Azure Synapse GPU-enabled pools are currently in Public Preview.

RAPIDS Accelerator for Apache Spark

The Spark RAPIDS accelerator is a plugin that works by overriding the physical plan of a Spark job by supported GPU operations, and running those operations on the GPUs, thereby accelerating processing. This library is currently in preview and doesn't support all Spark operations (here is a list of currently supported operators, and more support is being added incrementally through new releases).

Cluster configuration options

The RAPIDS Accelerator plugin only supports a one-to-one mapping between GPUs and executors. This means a Spark job would need to request executor and driver resources that can be accommodated by the pool resources (according to the number of available GPU and CPU cores). In order to meet this condition and ensure optimal utilization of all the pool resources, we require the following configuration of drivers and executors for a Spark application running on GPU-enabled pools:

POOL SIZE	DRIVER SIZE OPTIONS	DRIVER CORES	DRIVER MEMORY (GB)	EXECUTOR CORES	EXECUTOR MEMORY (GB)	NUMBER OF EXECUTORS
GPU-Large	Small driver	4	30	12	60	Number of nodes in pool
GPU-Large	Medium driver	7	30	9	60	Number of nodes in pool

POOL SIZE	DRIVER SIZE OPTIONS	DRIVER CORES	DRIVER MEMORY (GB)	EXECUTOR CORES	EXECUTOR MEMORY (GB)	NUMBER OF EXECUTORS
GPU-XLarge	Medium driver	8	40	14	80	4 * Number of nodes in pool
GPU-XLarge	Large driver	12	40	13	80	4 * Number of nodes in pool

Any workload that does not meet one of the above configurations will not be accepted. This is done to make sure Spark jobs are being run with the most efficient and performant configuration utilizing all available resources on the pool.

The user can set the above configuration through their workload. For notebooks, the user can use the `%%configure` magic command to set one of the above configurations as shown below. For example, using a large pool with three nodes:

```
%%configure -f
{
    "driverMemory": "30g",
    "driverCores": 4,
    "executorMemory": "60g",
    "executorCores": 12,
    "numExecutors": 3
}
```

Run a sample Spark job through notebook on an Azure Synapse GPU-accelerated pool

It would be good to be familiar with the basic concepts of how to use a notebook in Azure Synapse Analytics before proceeding with this section. Let's walk through the steps to run a Spark application utilizing GPU acceleration. You can write a Spark application in all the four languages supported inside Synapse, PySpark (Python), Spark (Scala), SparkSQL, and .NET for Spark (C#).

1. Create a GPU-enabled pool as described in this quickstart.
2. Create a notebook and attach it to the GPU-enabled pool you created in the first step.
3. Set the configurations as explained in the previous section.
4. Create a sample dataframe by copying the below code in the first cell of your notebook:

- Scala
- Python
- C#

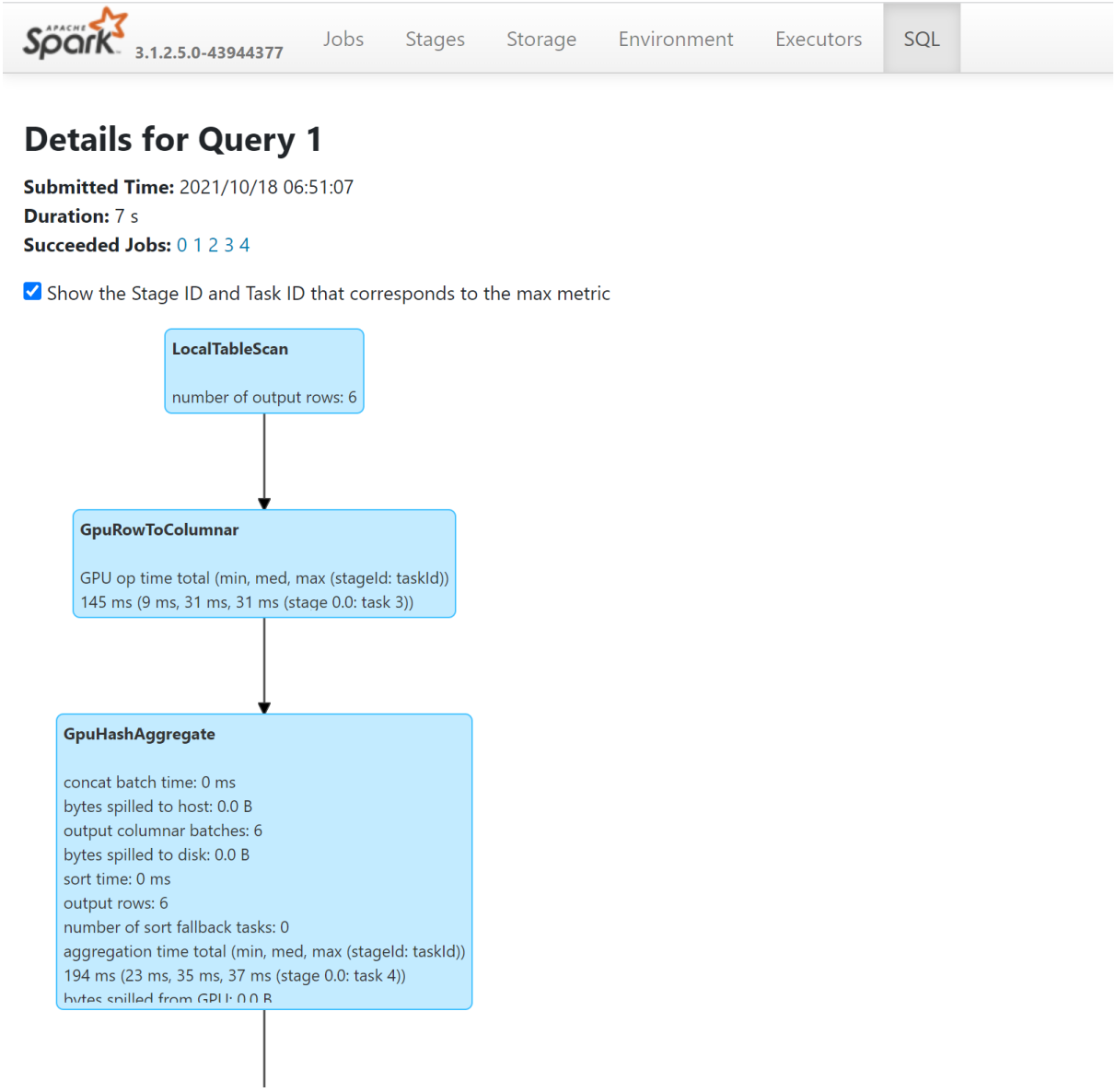
```
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}
import org.apache.spark.sql.Row
import scala.collection.JavaConversions._

val schema = StructType( Array(
  StructField("emp_id", IntegerType),
  StructField("name", StringType),
  StructField("emp_dept_id", IntegerType),
  StructField("salary", IntegerType)
))

val emp = Seq(Row(1, "Smith", 10, 100000),
    Row(2, "Rose", 20, 97600),
    Row(3, "Williams", 20, 110000),
    Row(4, "Jones", 10, 80000),
    Row(5, "Brown", 40, 60000),
    Row(6, "Brown", 30, 78000)
  )

val empDF = spark.createDataFrame(emp, schema)
```

5. Now let's do an aggregate by getting the maximum salary per department ID and display the result:

- Scala
- Python
- C#

```
val resultDF = empDF.groupBy("emp_dept_id").max("salary")
resultDF.show()
```

6. You can see the operations in your query that ran on GPUs by looking into the SQL plan through the Spark History Server:



How to tune your application for GPUs

Most Spark jobs can see improved performance through tuning configuration settings from defaults, and the same holds true for jobs leveraging the RAPIDS accelerator plugin for Apache Spark. This documentation provides guidelines on how to tune a Spark job to run on GPUs using the RAPIDS plugin.

Quotas and resource constraints in Azure Synapse GPU-enabled pools

Workspace level

Every Azure Synapse workspace comes with a default quota of 50 GPU vCores. In order to increase your quota of GPU cores, please submit a support request through the Azure portal.

Next steps

- Azure Synapse Analytics

Use extended Apache Spark history server to debug and diagnose Apache Spark applications

8/19/2022 • 7 minutes to read • Edit Online

This article provides guidance on how to use the extended Apache Spark history server to debug and diagnose completed and running Spark applications.

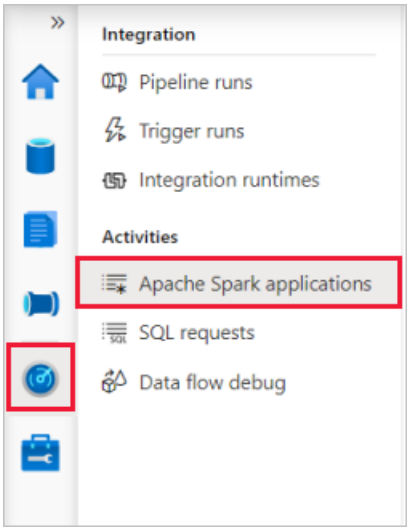
The extension includes a data tab, graph tab, and diagnosis tab. Use the **Data** tab to check the input and output data of the Spark job. The **Graph** tab shows you the data flow and replay of the job graph. The **Diagnosis** tab shows you **Data Skew**, **Time Skew**, and **Executor Usage Analysis**.

Access the Apache Spark history server

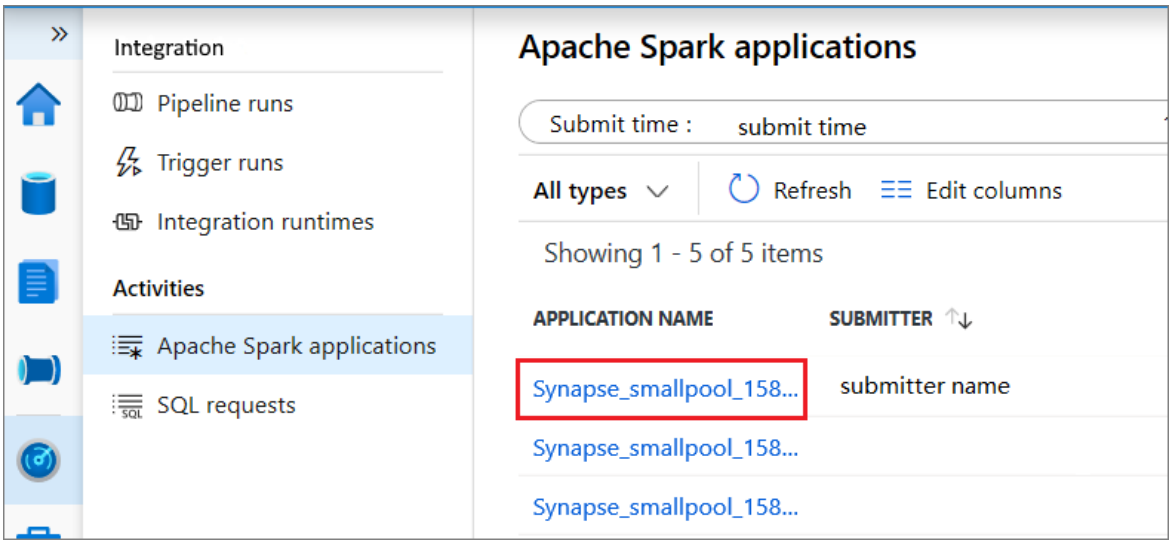
Apache Spark history server is the web user interface for completed and running Spark applications. You can open the Apache Spark history server web interface from the Azure Synapse Analytics.

Open the Spark History Server web UI from Apache spark applications node

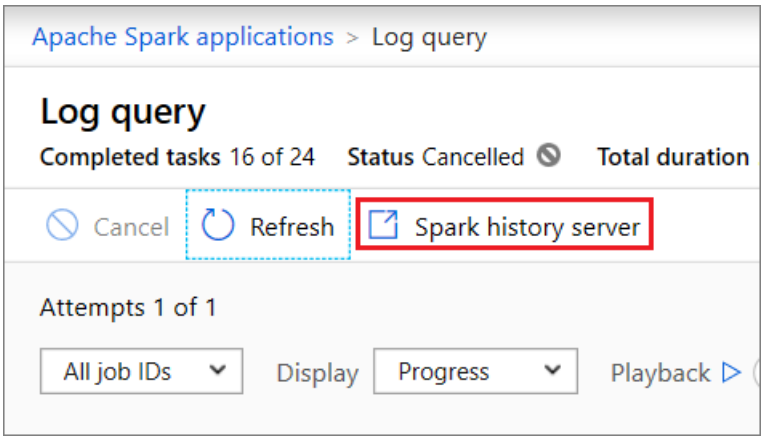
1. Open Azure Synapse Analytics.
2. Select **Monitor**, then select **Apache Spark Applications**.



3. Select an application, then open **Log query** by selecting it.

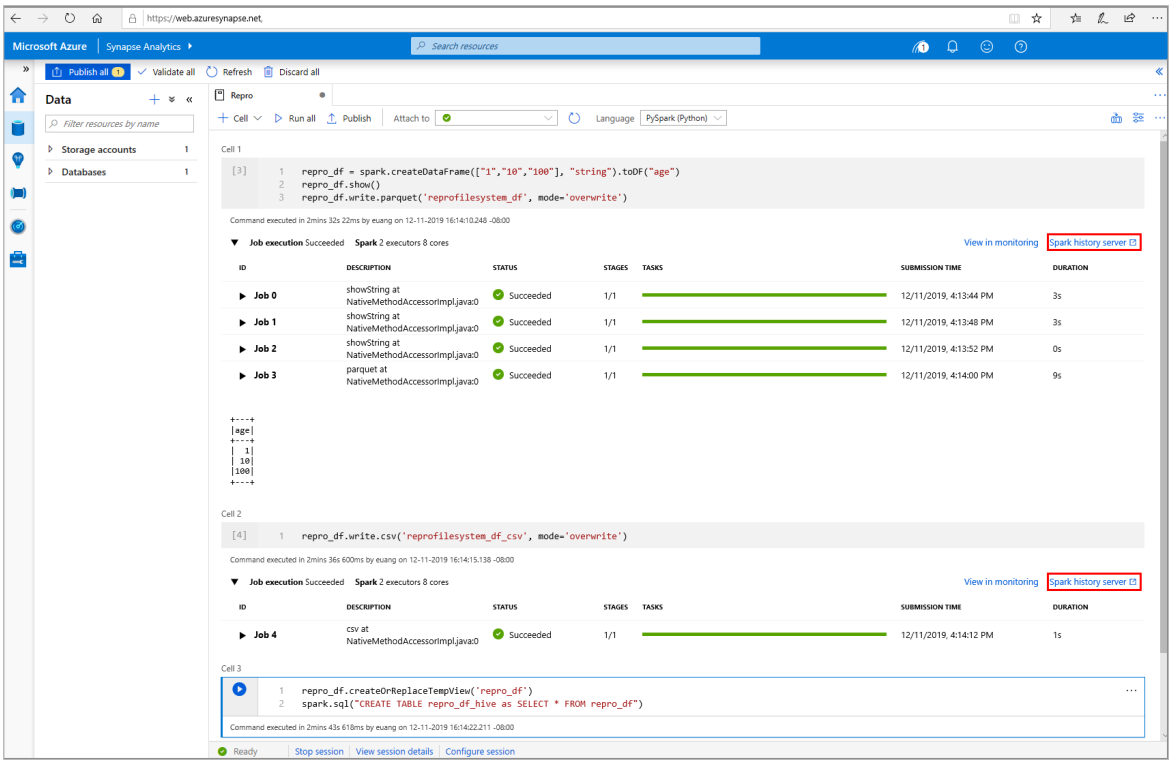


4. Select **Spark history server**, then the Spark History Server web UI will show up.

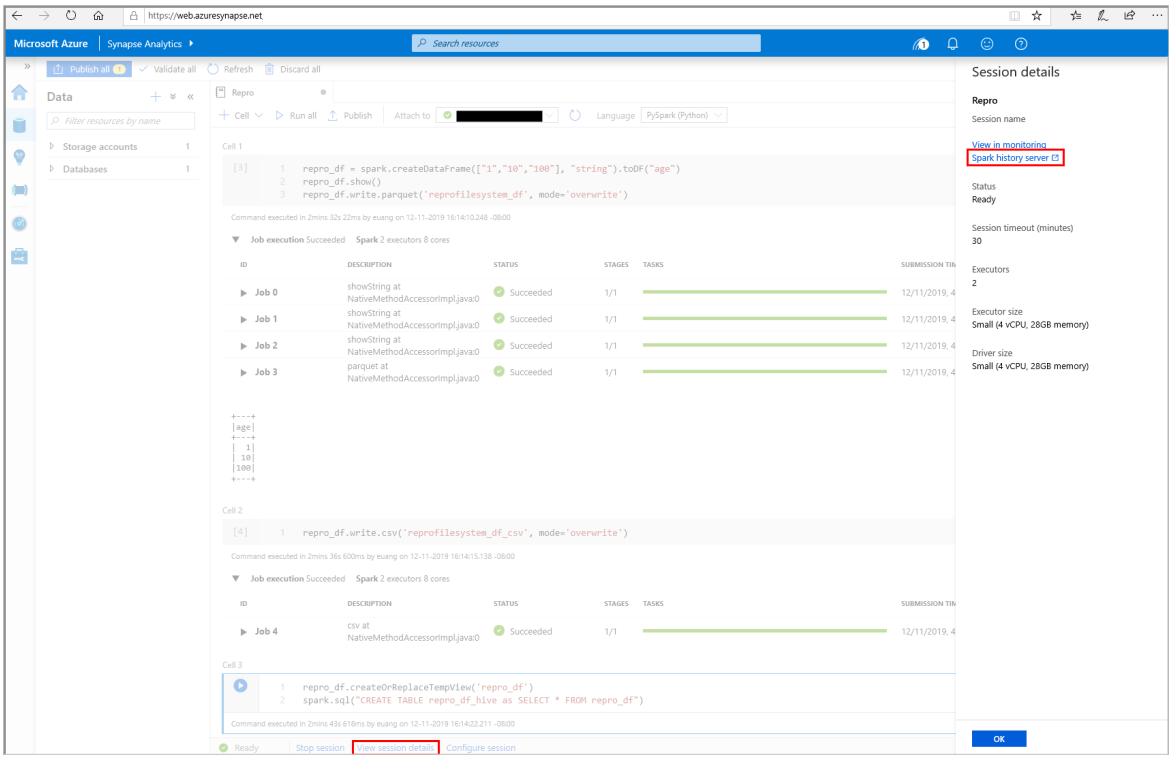


Open the Spark History Server web UI from Data node

1. From your Synapse Studio notebook, select **Spark history server** from the job execution output cell or from the status panel at the bottom of the notebook document. Select **Session details**.



2. Select **Spark history server** from the slide out panel.

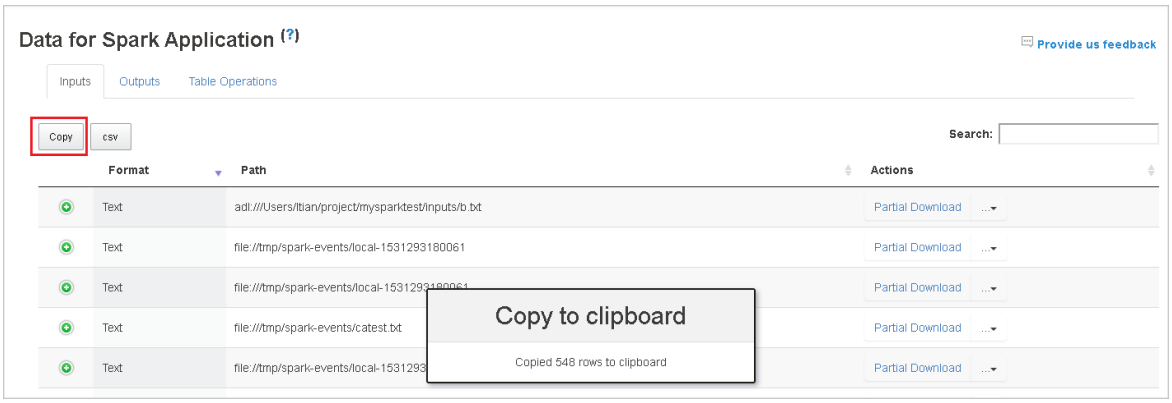


Explore the Data tab in Spark history server

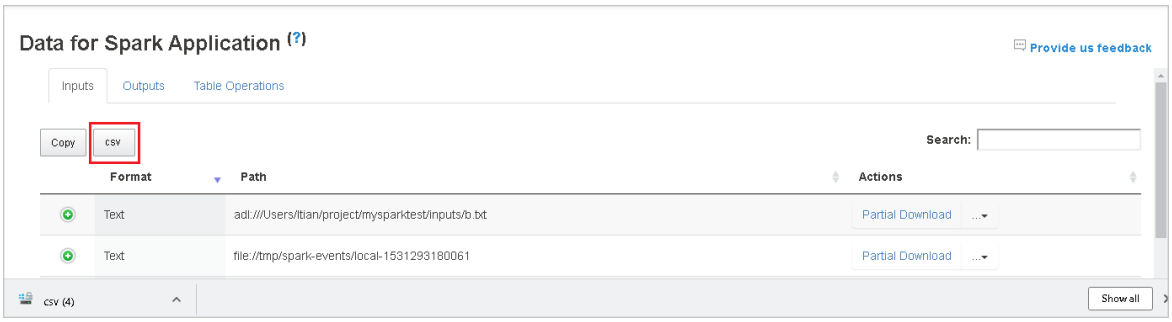
Select the Job ID for the job you want to view. Then select **Data** on the tool menu to get the data view. This section shows you how to do various tasks in the Data tab.

- Check the **Inputs**, **Outputs**, and **Table Operations** by selecting the tabs separately.

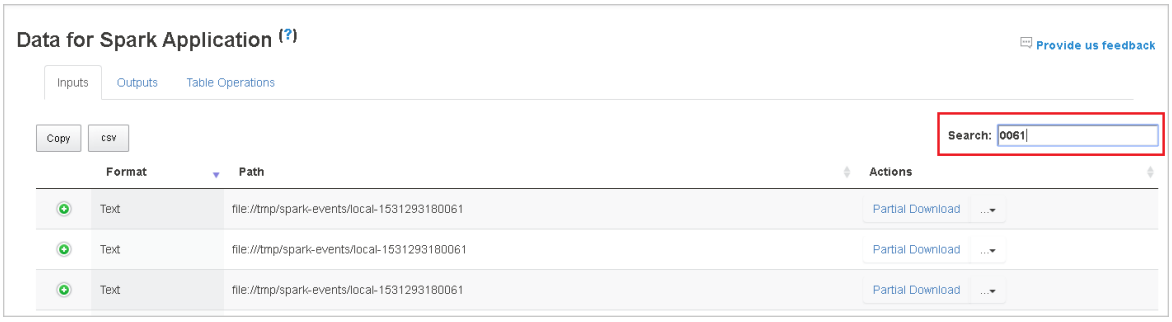
- Copy all rows by selecting **Copy**.



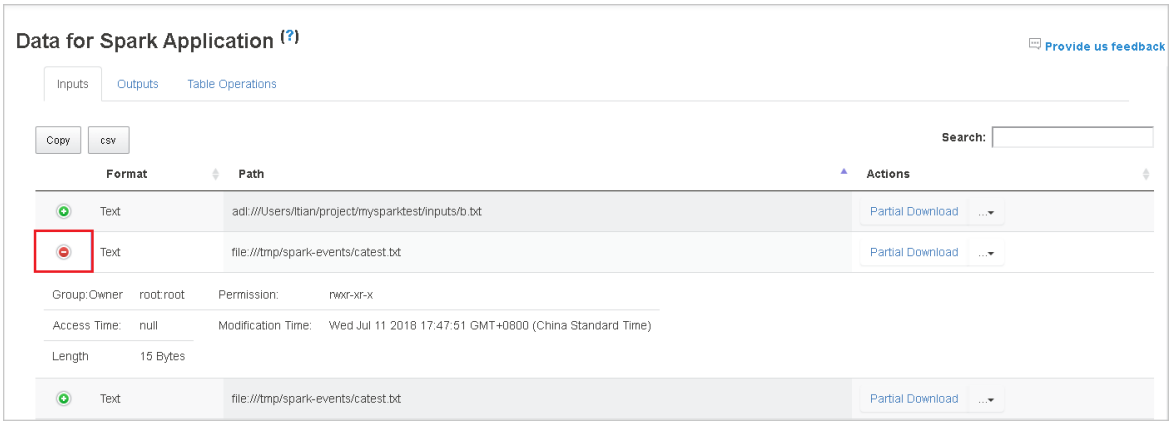
- Save all data as CSV file by selecting **csv**.



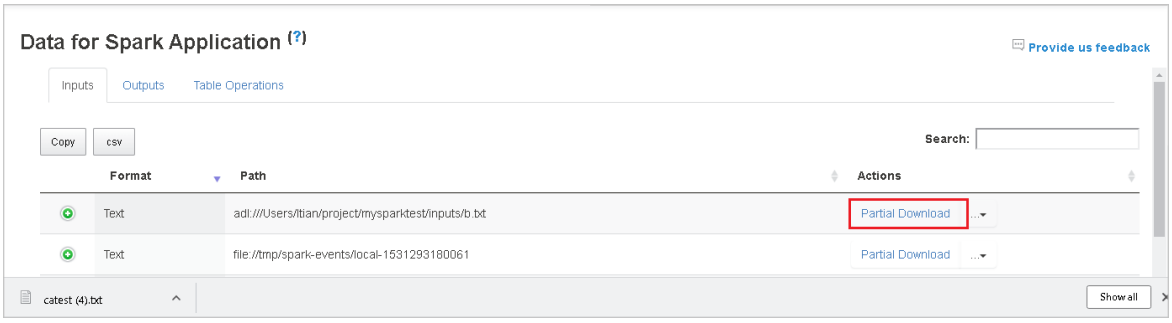
- Search by entering keywords in field **Search**. The search results display immediately.



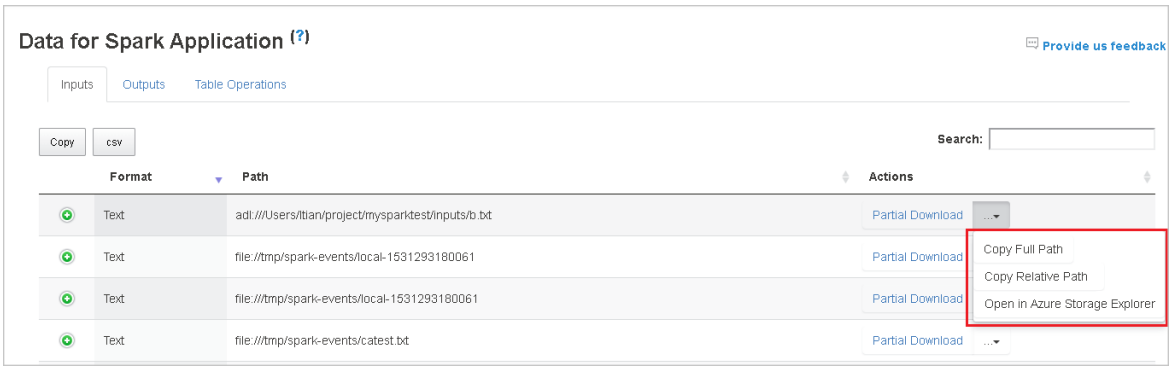
- Select the column header to sort table, select the plus sign to expand a row to show more details, or select the minus sign to collapse a row.



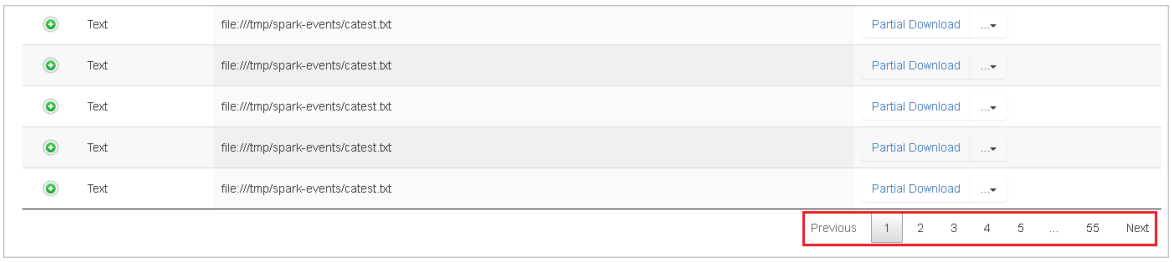
- Download a single file by selecting **Partial Download**. The selected file is downloaded to local. If the file no longer exists, a new tab appears with an error message.



- To copy a full path or relative path, select the **Copy Full Path** or **Copy Relative Path** options that expand from the drop-down menu. For Azure Data Lake Storage files, **Open in Azure Storage Explorer** launches Azure Storage Explorer and locates the folder when you are signed in.

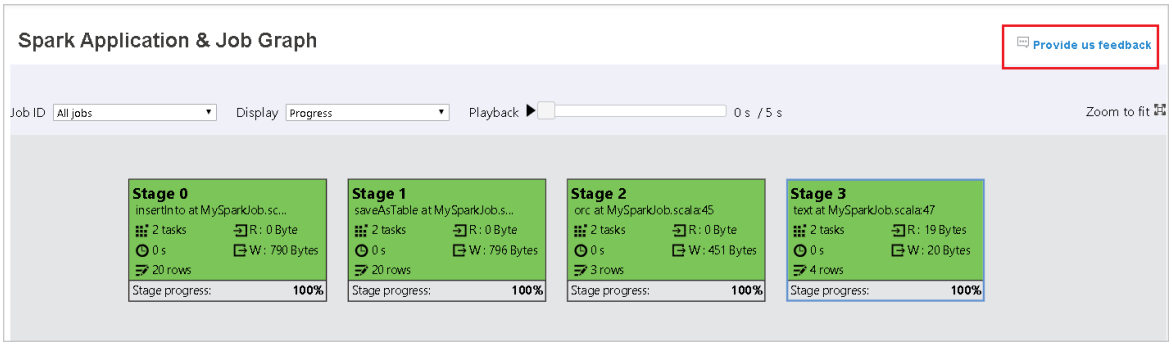


- Select page numbers below the table to navigate pages when there are too many rows to display in one page.



- Hover on the question mark beside **Data** to show the tooltip, or select the question mark to get more information.

- Send feedback with issues by selecting **Provide us feedback**.

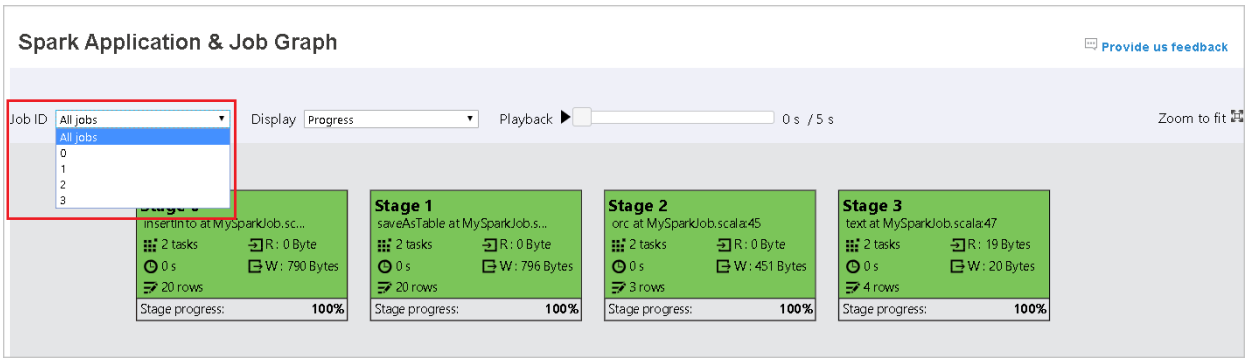


Graph tab in Apache Spark history server

Select the Job ID for the job you want to view. Then, select **Graph** on the tool menu to get the job graph view.

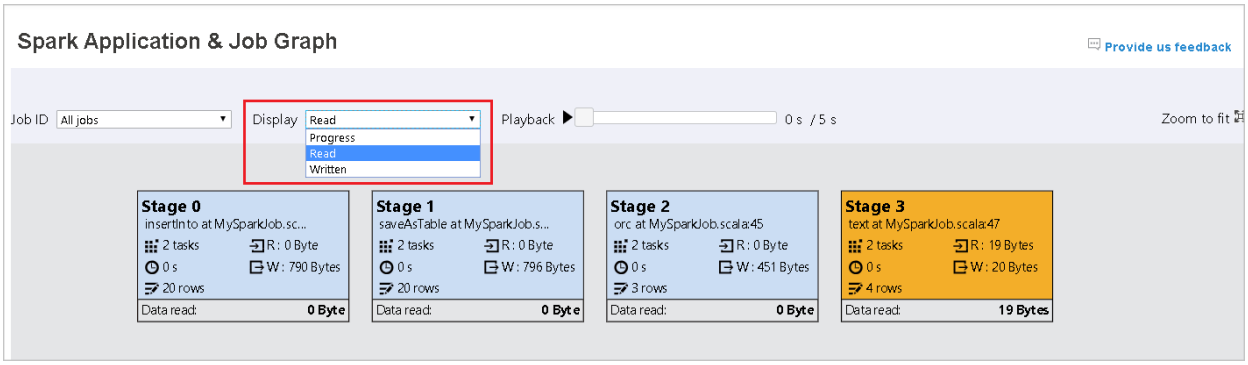
Overview

You can see an overview of your job in the generated job graph. By default, the graph shows all jobs. You can filter this view by **Job ID**.

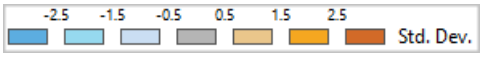


Display

By default, the **Progress** display is selected. You can check the data flow by selecting **Read** or **Written** in the **Display** dropdown list.



The graph node displays the colors shown in the heatmap legend.

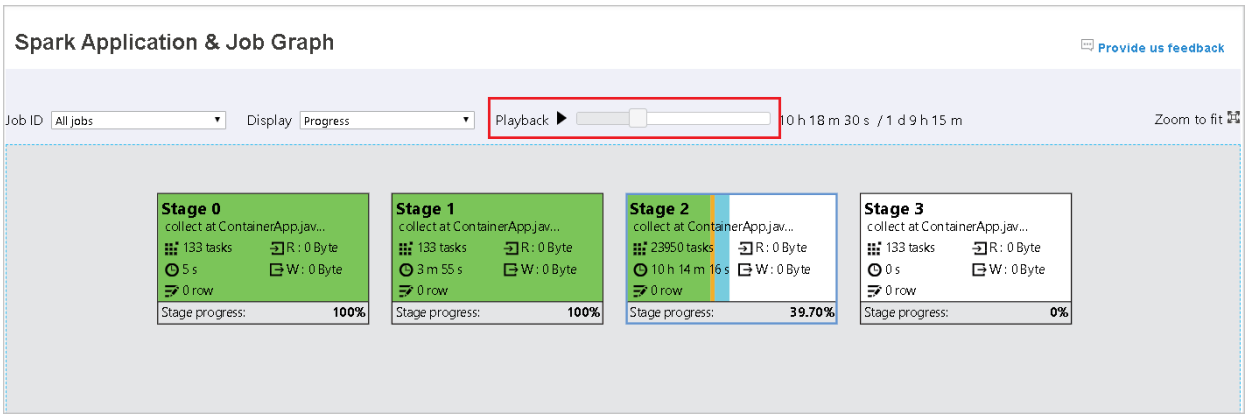


Playback

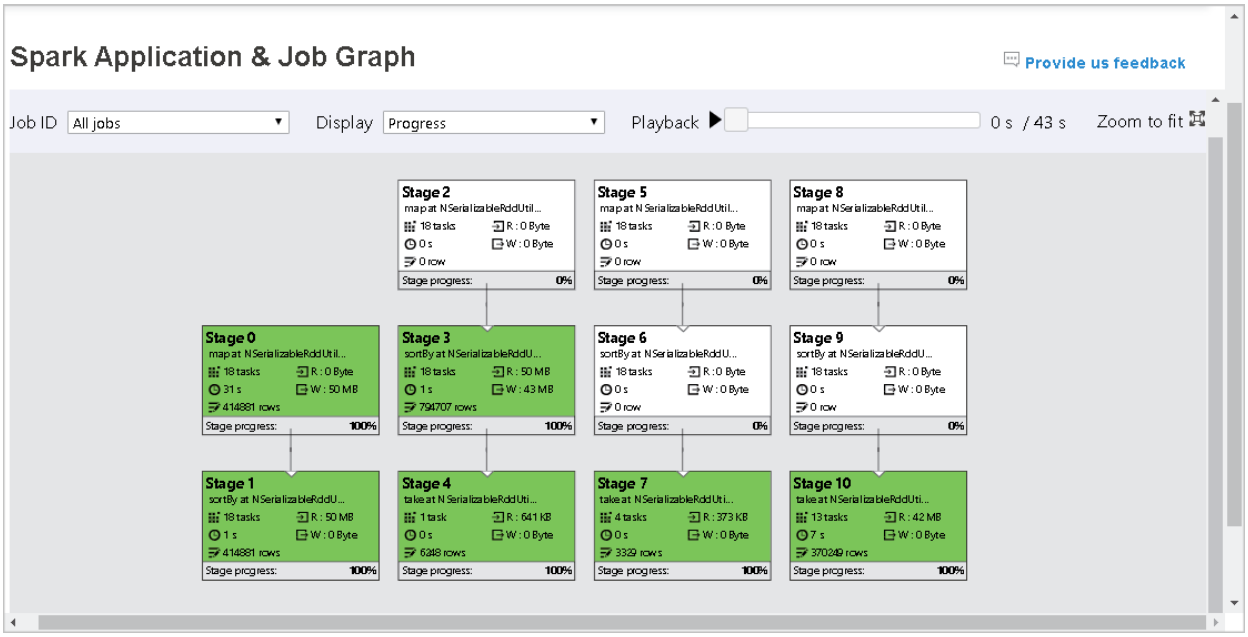
To playback the job, select **Playback**. You can select **Stop** at any time to stop. The task colors show different statuses when playing back:

COLOR	MEANING
Green	Succeeded: The job has completed successfully.
Orange	Retried: Instances of tasks that failed but do not affect the final result of the job. These tasks had duplicate or retry instances that may succeed later.
Blue	Running: The task is running.
White	Waiting or skipped: The task is waiting to run, or the stage has skipped.
Red	Failed: The task has failed.

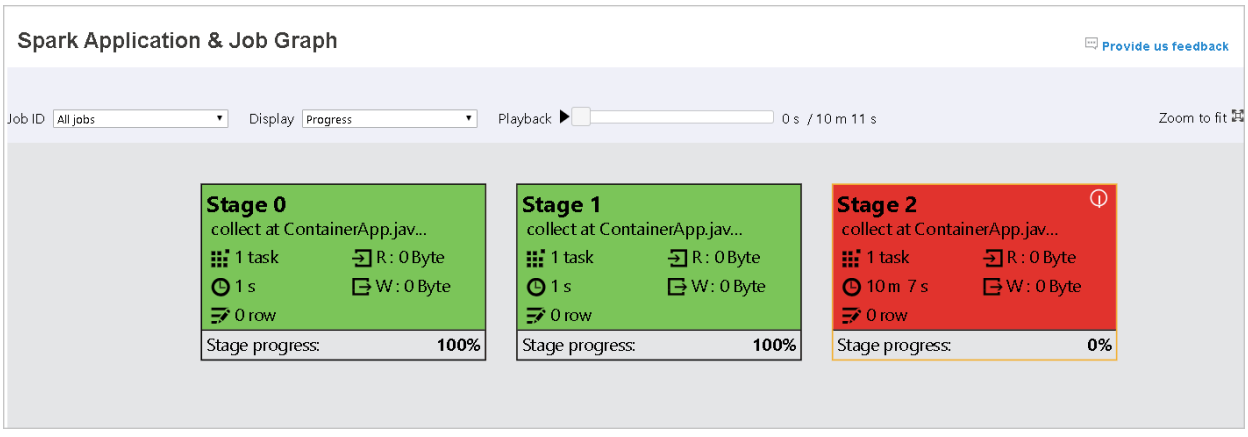
The following image shows Green, Orange, and Blue status colors.



The following image shows Green and White status colors.



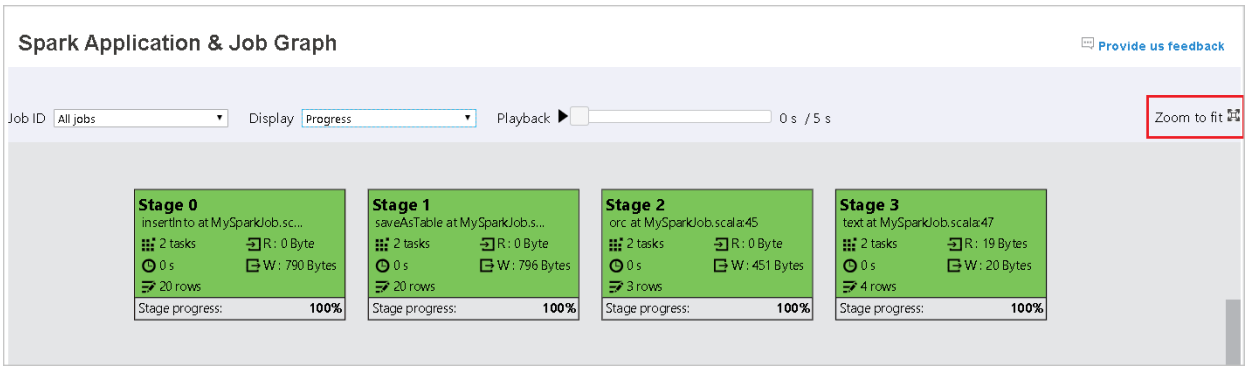
The following image shows Red and Green status colors.



> **NOTE**
>
> Playback for each job is allowed. For incomplete jobs, playback is not supported.

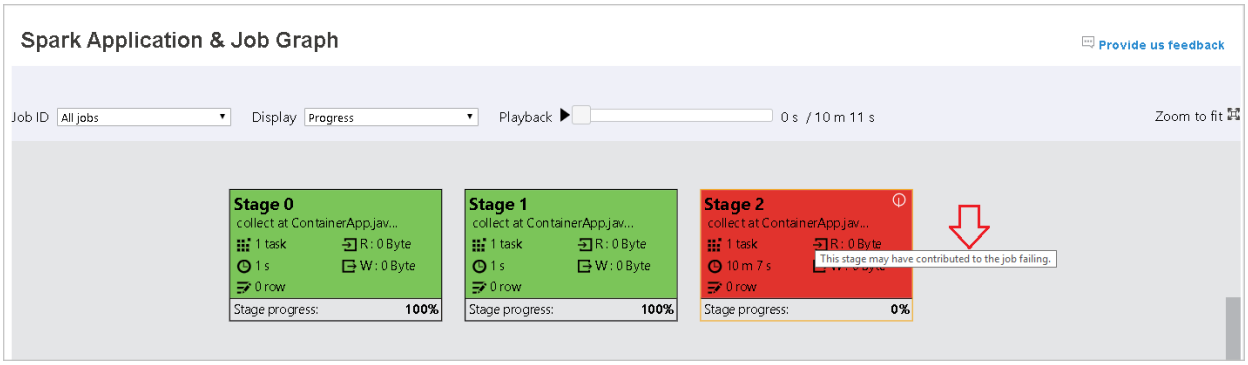
Zoom

Use your mouse scroll to zoom in and out on the job graph, or select **Zoom to fit** to make it fit to screen.



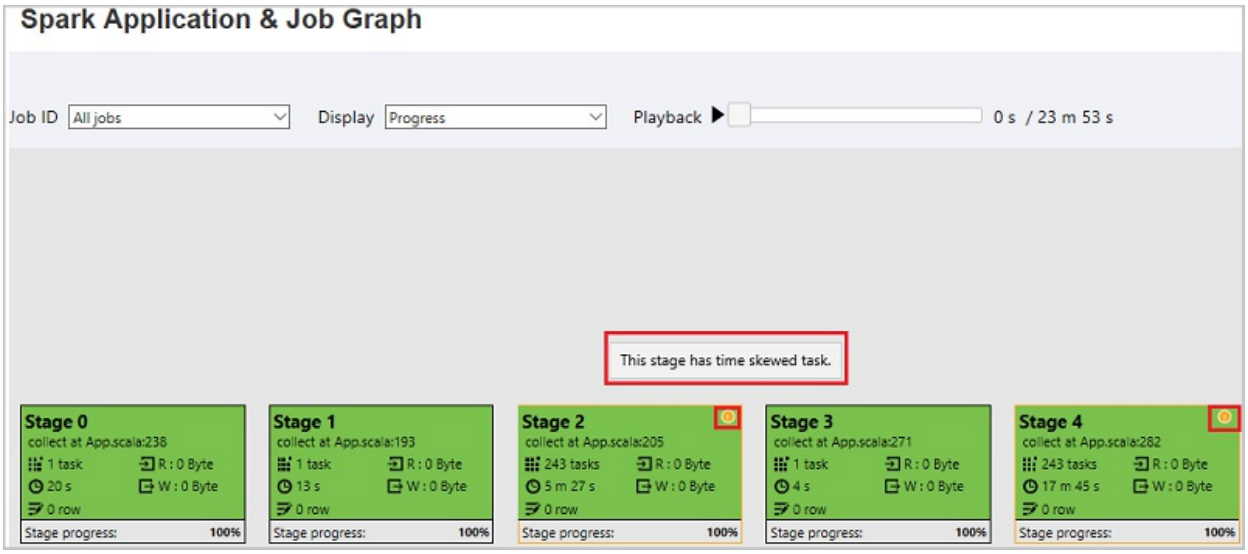
Tooltips

Hover on graph node to see the tooltip when there are failed tasks, and select a stage to open its stage page.



On the job graph tab, stages have a tooltip and a small icon displayed if they have tasks that meet the following conditions:

CONDITION	DESCRIPTION
Data skew	data read size > average data read size of all tasks inside this stage * 2 and data read size > 10 MB
Time skew	execution time > average execution time of all tasks inside this stage * 2 and execution time > 2 minutes



Graph node description

The job graph node displays the following information of each stage:

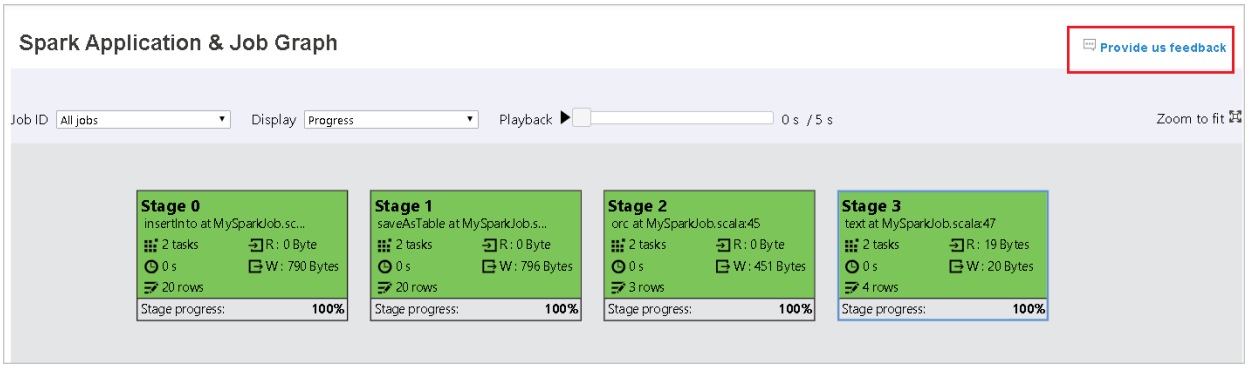
- ID.
- Name or description.

- Total task number.
- Data read: the sum of input size and shuffle read size.
- Data write: the sum of output size and shuffle writes size.
- Execution time: the time between start time of the first attempt and completion time of the last attempt.
- Row count: the sum of input records, output records, shuffle read records and shuffle write records.
- Progress.

> **NOTE**
>
> By default, the job graph node displays information from the last attempt of each stage (except for stage execution time). However, during playback, the graph node shows information of each attempt.
>
> The data size of read and write is 1MB = 1000 KB = 1000 * 1000 Bytes.

Provide feedback

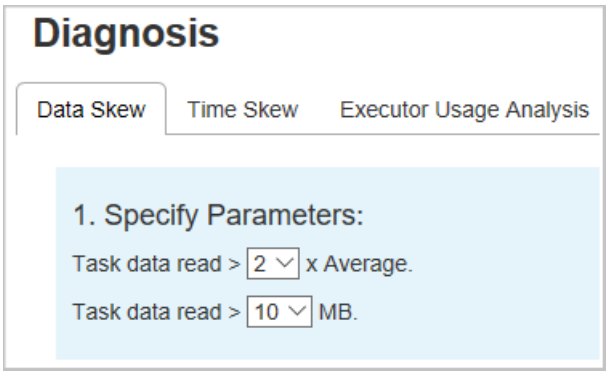
Send feedback with issues by selecting **Provide us feedback**.



Explore the Diagnosis tab in Apache Spark history server

To access the Diagnosis tab, select a job ID. Then select **Diagnosis** on the tool menu to get the job Diagnosis view. The diagnosis tab includes **Data Skew**, **Time Skew**, and **Executor Usage Analysis**.

Check the **Data Skew**, **Time Skew**, and **Executor Usage Analysis** by selecting the tabs respectively.



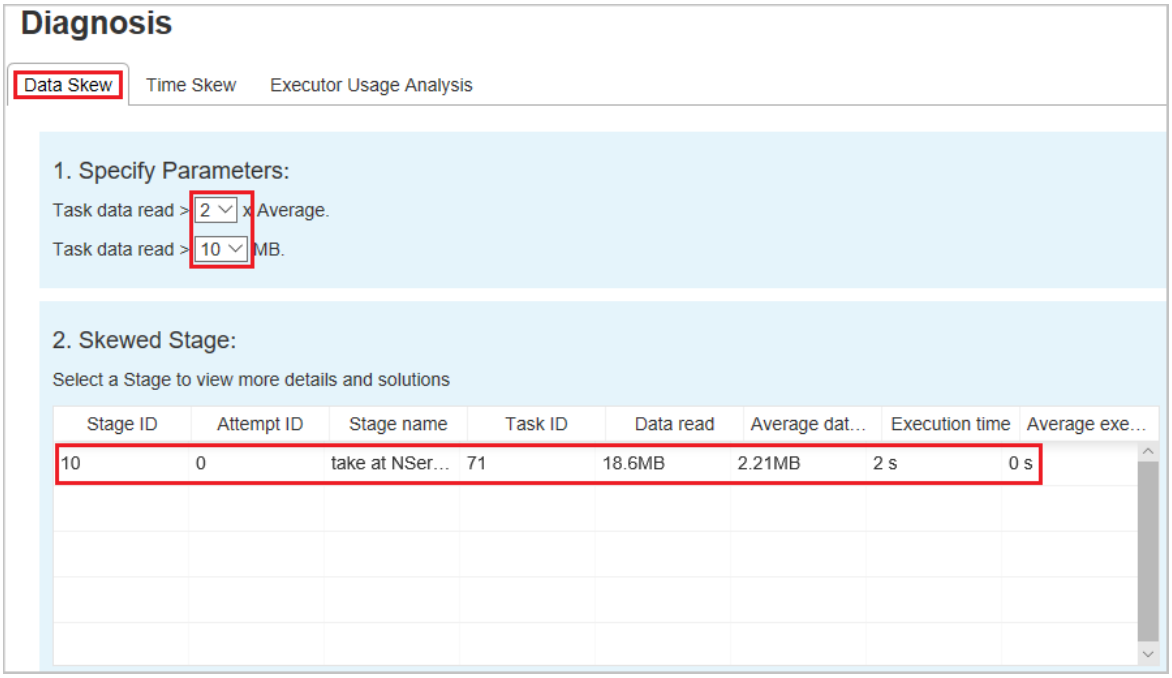
Data Skew

When you select the **Data Skew** tab, the corresponding skewed tasks are displayed based on the specified parameters.

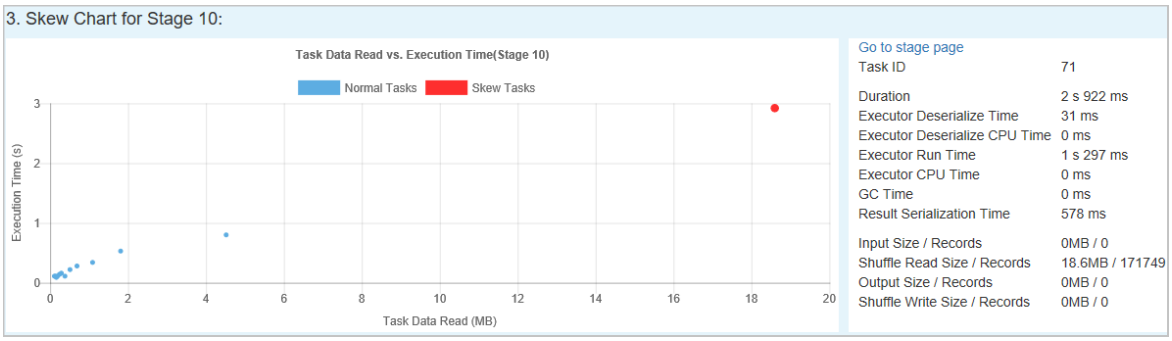
- **Specify Parameters** - The first section displays the parameters, which are used to detect Data Skew. The default rule is: Task Data Read is greater than three times of the average task data read, and the task data read is more than 10 MB. If you want to define your own rule for skewed tasks, you can choose your

parameters. The **Skewed Stage** and **Skew Char** sections are refreshed accordingly.

- **Skewed Stage** - The second section displays stages, which have skewed tasks meeting the criteria specified above. If there is more than one skewed task in a stage, the skewed stage table only displays the most skewed task (for example, the largest data for data skew).



- **Skew Chart** – When a row in the skew stage table is selected, the skew chart displays more task distribution details based on data read and execution time. The skewed tasks are marked in red and the normal tasks are marked in blue. The chart displays up to 100 sample tasks, and the task details are displayed in right bottom panel.

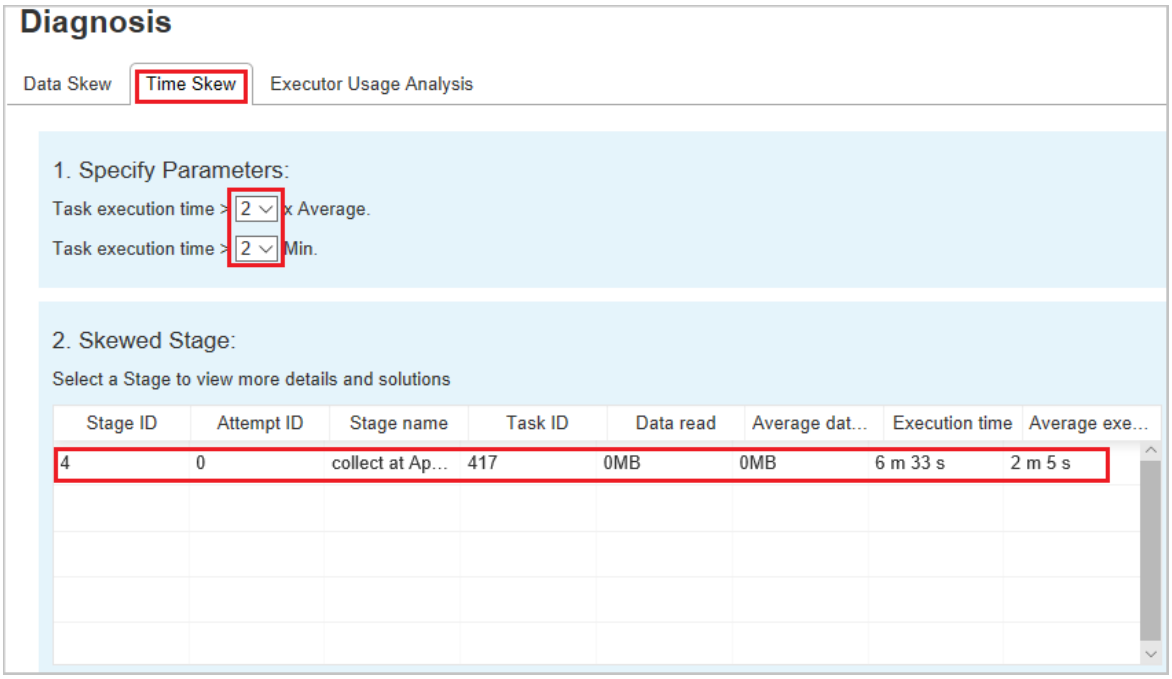


Time Skew

The **Time Skew** tab displays skewed tasks based on task execution time.

- **Specify Parameters** - The first section displays the parameters, which are used to detect Time Skew. The default criteria to detect time skew is: task execution time is greater than three times of average execution time and task execution time is greater than 30 seconds. You can change the parameters based on your needs. The **Skewed Stage** and **Skew Chart** display the corresponding stages and tasks information just like the **Data Skew** tab above.

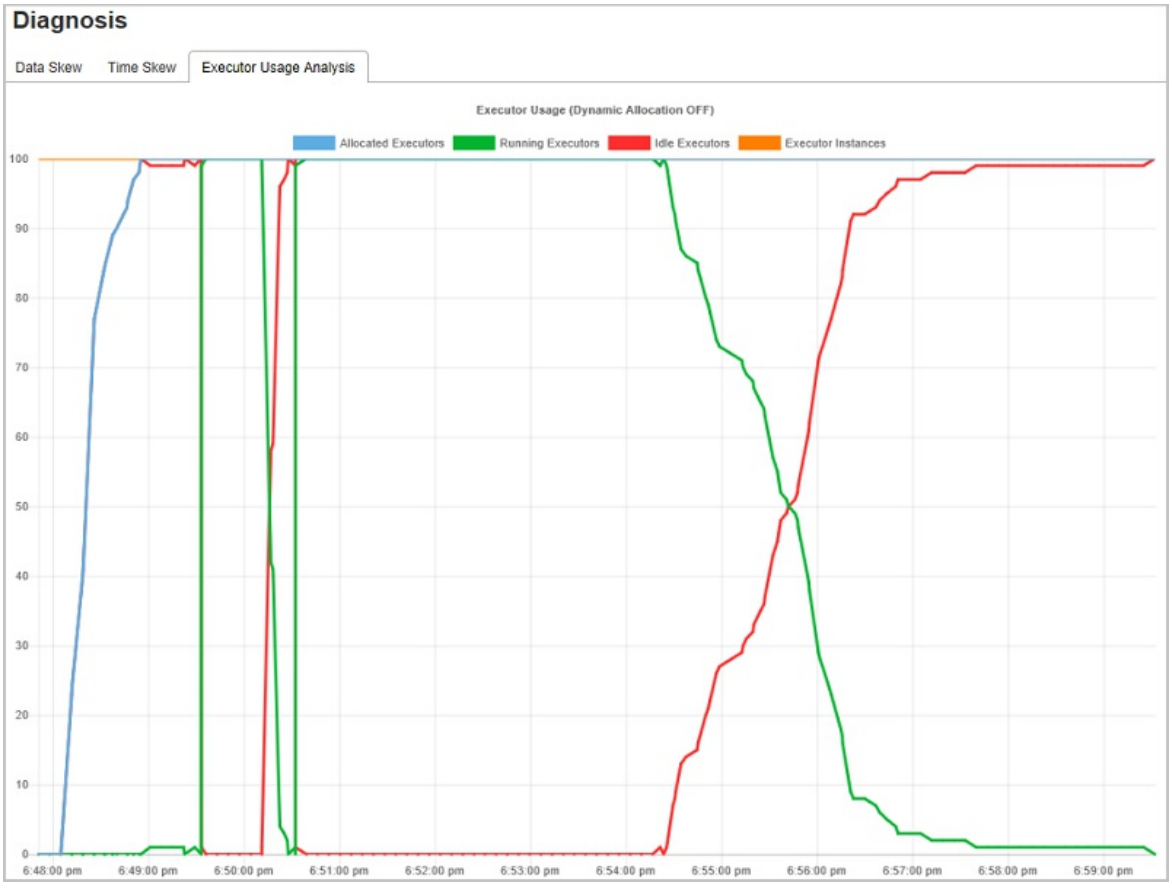
- Select **Time Skew**, then filtered result is displayed in **Skewed Stage** section according to the parameters set in section **Specify Parameters**. Select one item in **Skewed Stage** section, then the corresponding chart is drafted in section3, and the task details are displayed in right bottom panel.



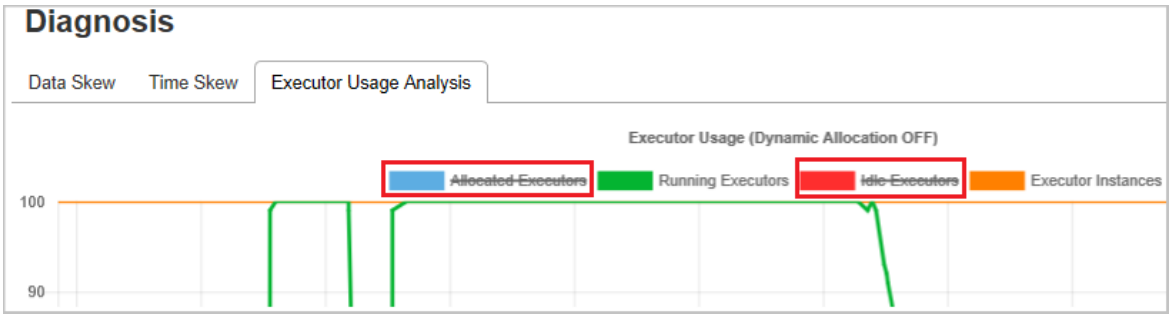
Executor Usage Analysis

The Executor Usage Graph visualizes the Spark job executor's allocation and running status.

1. Select **Executor Usage Analysis**, then four types curves about executor usage are drafted, including **Allocated Executors**, **Running Executors**, **Idle Executors**, and **Max Executor Instances**. For allocated executors, each "Executor added" or "Executor removed" event increases or decreases the allocated executors. You can check "Event Timeline" in the "Jobs" tab for more comparison.



2. Select the color icon to select or unselect the corresponding content in all drafts.



Known issues

Input/output data using Resilient Distributed Datasets (RDDs) doesn't show in the data tab.

Next steps

- Azure Synapse Analytics
- .NET for Apache Spark documentation

Use Synapse Studio to monitor your Apache Spark applications

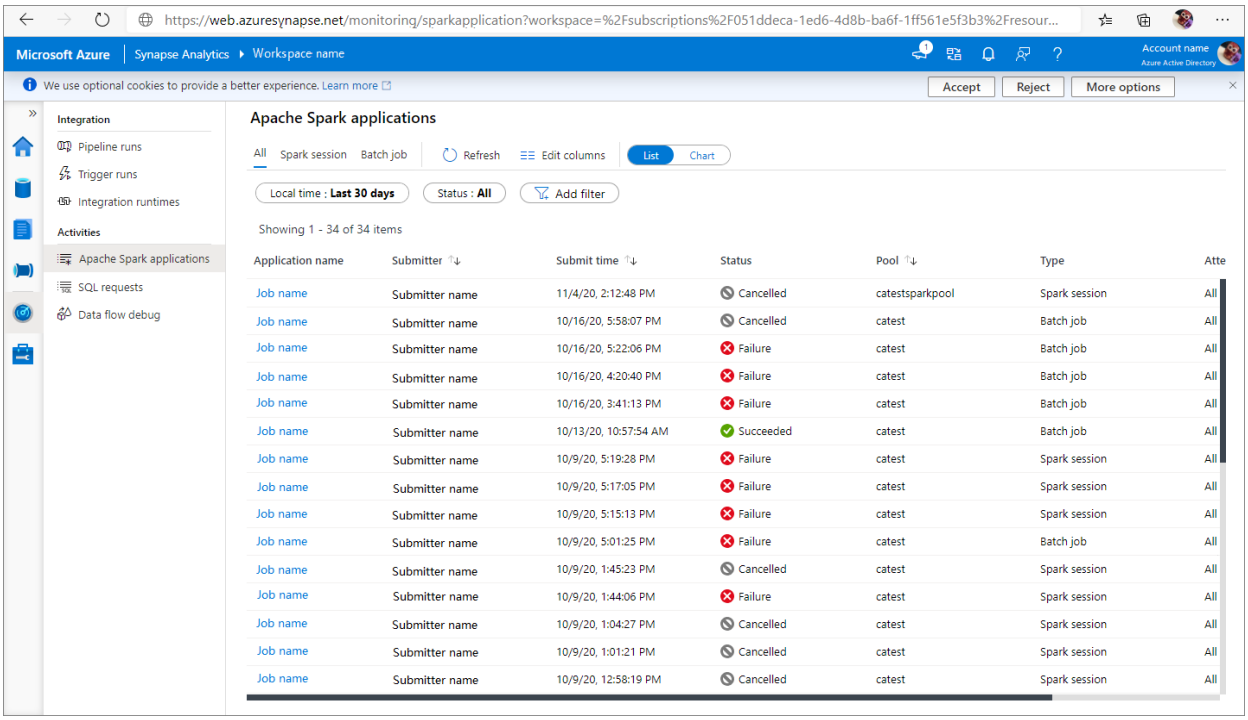
8/19/2022 • 5 minutes to read • Edit Online

With Azure Synapse Analytics, you can use Apache Spark to run notebooks, jobs, and other kinds of applications on your Apache Spark pools in your workspace.

This article explains how to monitor your Apache Spark applications, allowing you to keep an eye on the latest status, issues, and progress.

View Apache Spark applications

You can view all Apache Spark applications from **Monitor** -> **Apache Spark applications**.

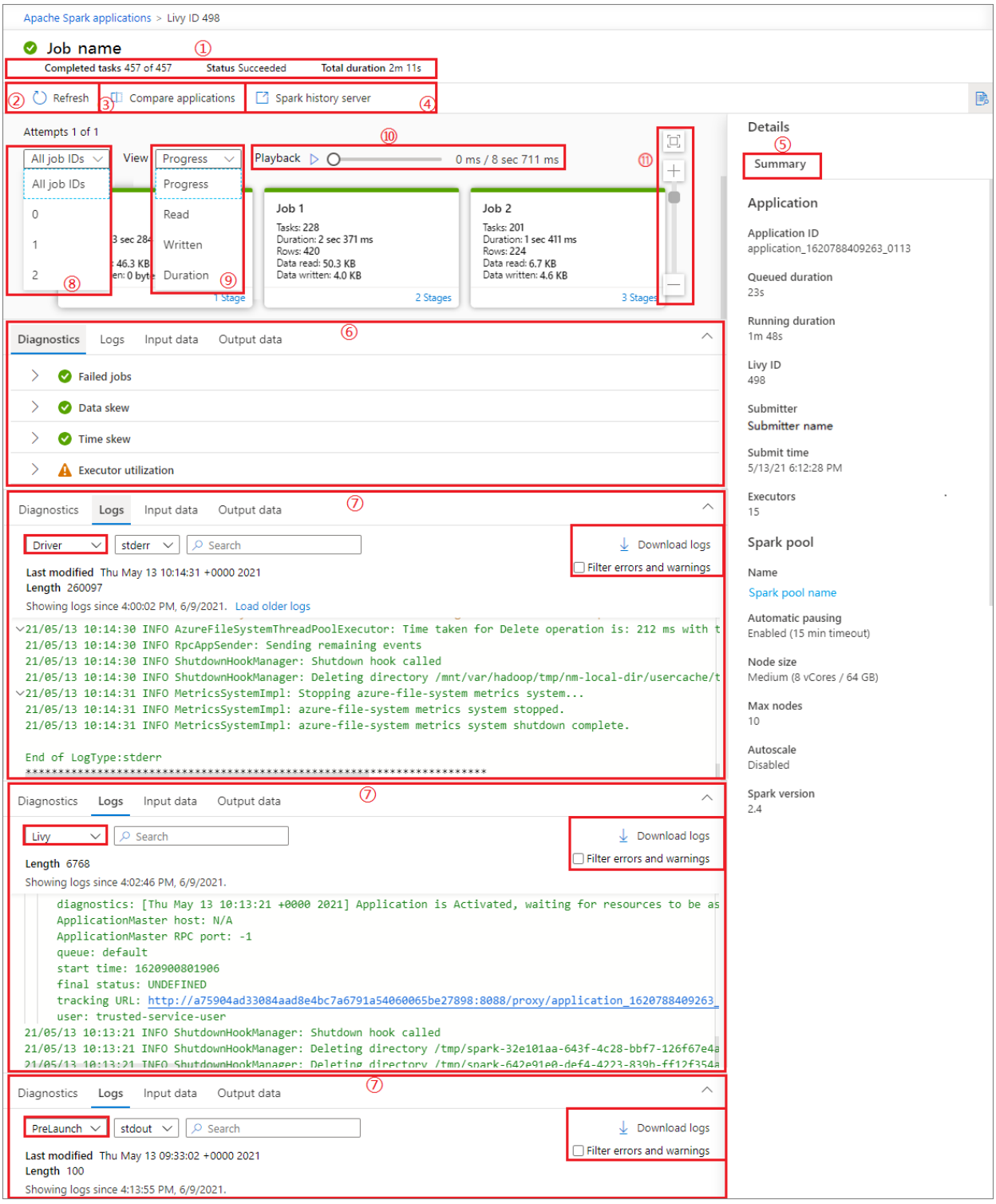


View completed Apache Spark applications

Open **Monitor**, then select **Apache Spark applications**. To view the details about the completed Apache Spark applications, select the Apache Spark application.

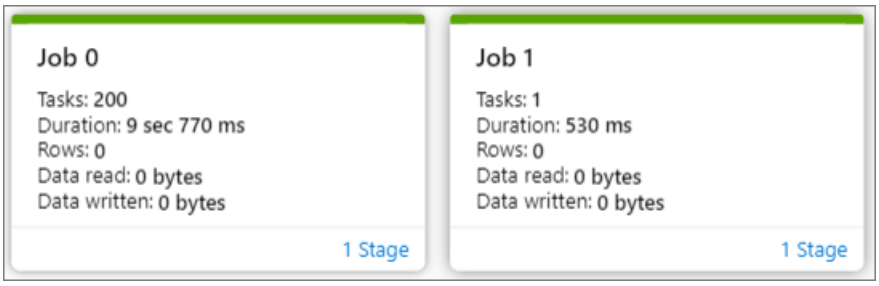
1. Check the **Completed tasks**, **Status**, and **Total duration**.
2. **Refresh** the job.
3. Click on **Compare applications** to use the comparison feature, for more information on this feature, see the Compare Apache Spark applications.
4. Click on **Spark history server** to open the History Server page.
5. Check the **Summary** info.

6. Check the diagnostics in **Diagnostic** tab.
7. Check the **Logs**. You can view full log of **Livy**, **Prelaunch**, and **Driver** logs via selecting different options in the drop-down list. And you can directly retrieve the required log information by searching keywords. Click **Download log** to download the log information to the local, and select the **Filter errors and warnings** check box to filter the errors and warnings you need.
8. You can see an overview of your job in the generated job graph. By default, the graph shows all jobs. You can filter this view by **Job ID**.
9. By default, the **Progress** display is selected. You can check the data flow by selecting **Progress/Read/Written/Duration** in the **View** dropdown list.
10. To play back the job, click on **Playback** button. You can click on **Stop** button at any time to stop.
11. Use scroll bar to zoom in and zoom out the job graph, you can also select **Zoom to Fit** to make it fit the screen.



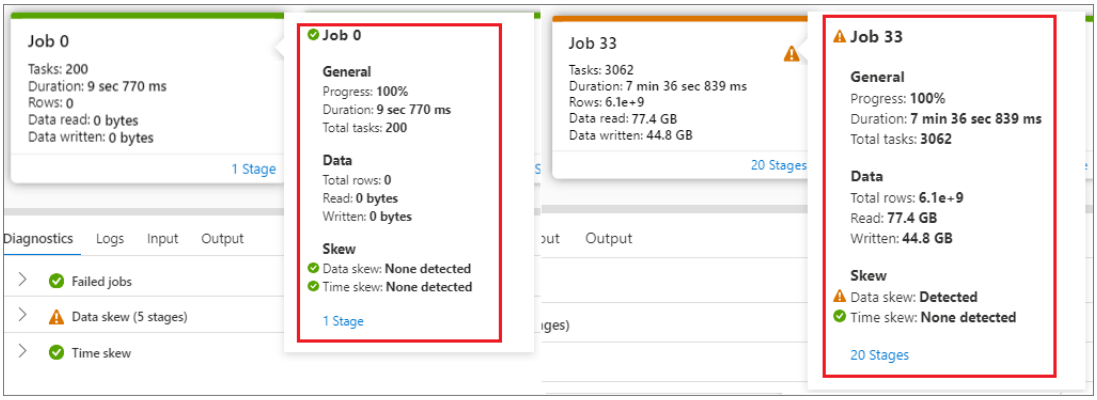
12. The job graph node displays the following information of each stage:

    - Job ID
    - Task number
    - Duration time
    - Row count
    - Data read: the sum of input size and shuffle read size
    - Data written: the sum of output size and shuffle writes size
    - Stage number



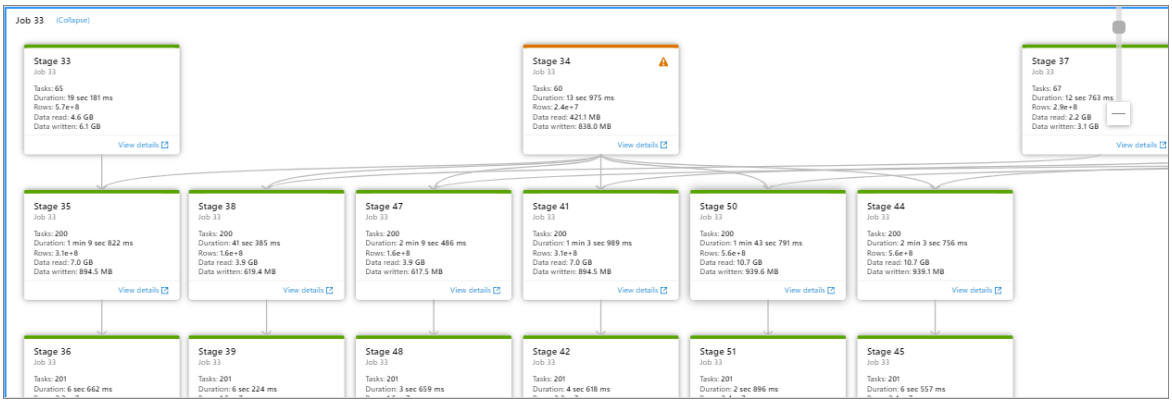
13. Hover the mouse over a job, and the job details will be displayed in the tooltip:

    - Icon of job status: If the job status is successful, it will be displayed as a green "Ã"; if the job detects a problem, it will display a yellow "!"
    - Job ID
    - General part:
        - Progress
        - Duration time
        - Total tasks number
    - Data part:
        - Total rows number
        - Read size
        - Written size
    - Skew part:
        - Data skew
        - Time skew
    - Stage number



14. Click **Stage number** to expand all the stages contained in the job. Click **Collapse** next to the Job ID to collapse all the stages in the job.

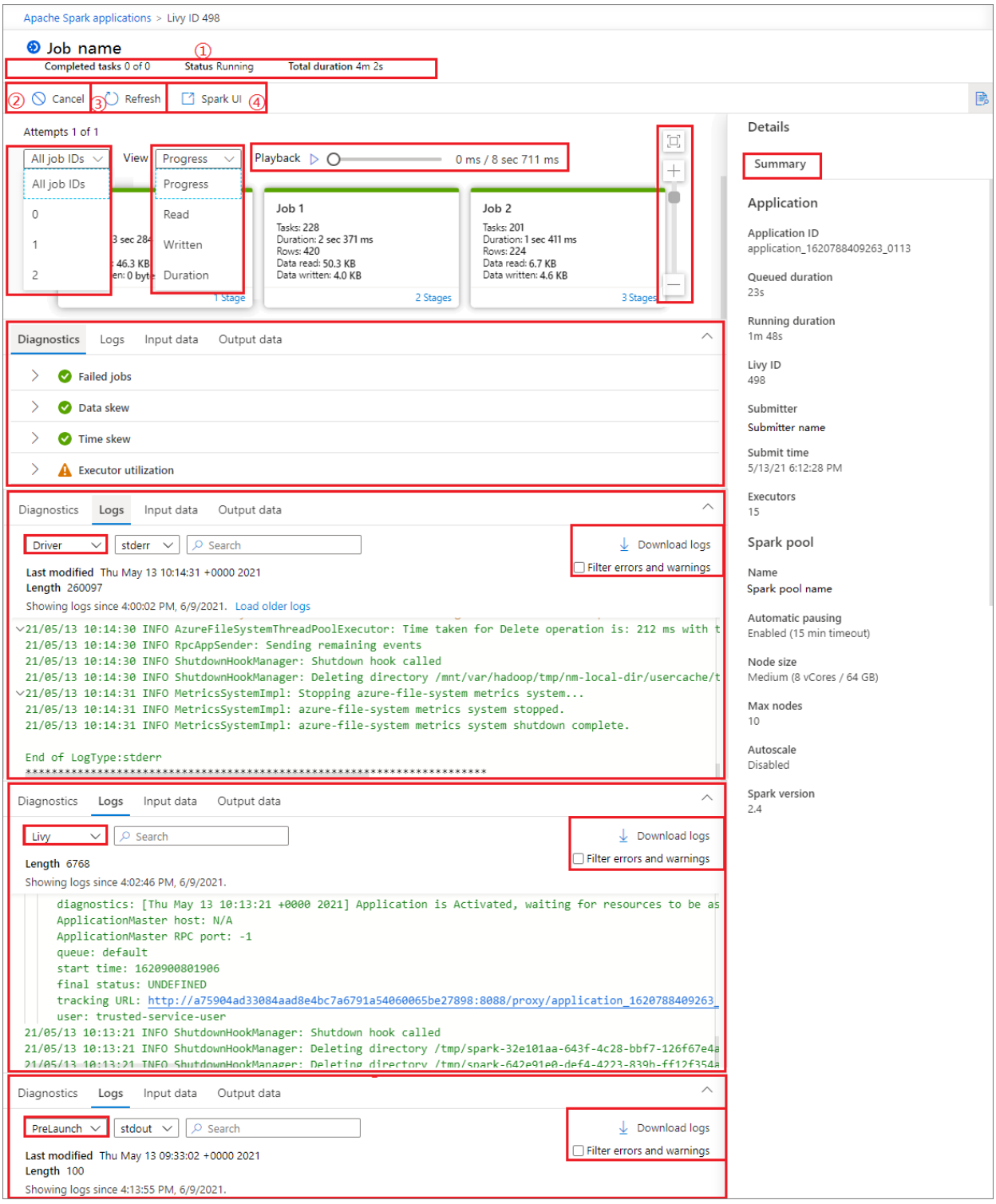
15. Click on **View details** in a stage graph, then the details for a stage will appear.



Monitor Apache Spark application progress

Open **Monitor**, then select **Apache Spark applications**. To view the details about the Apache Spark applications that are running, select the submitted Apache Spark application. If the Apache Spark application is still running, you can monitor the progress.

1. Check the **Completed tasks**, **Status**, and **Total duration**.
2. **Cancel** the Apache Spark application.
3. **Refresh** the Job.
4. Click on **Spark UI** button to go to Spark Job page.
5. For **Job graph**, **Summary**, **Diagnostics**, **Logs**. You can see an overview of your job in the generated job graph. Refer to steps 5 - 15 of View completed Apache Spark applications.



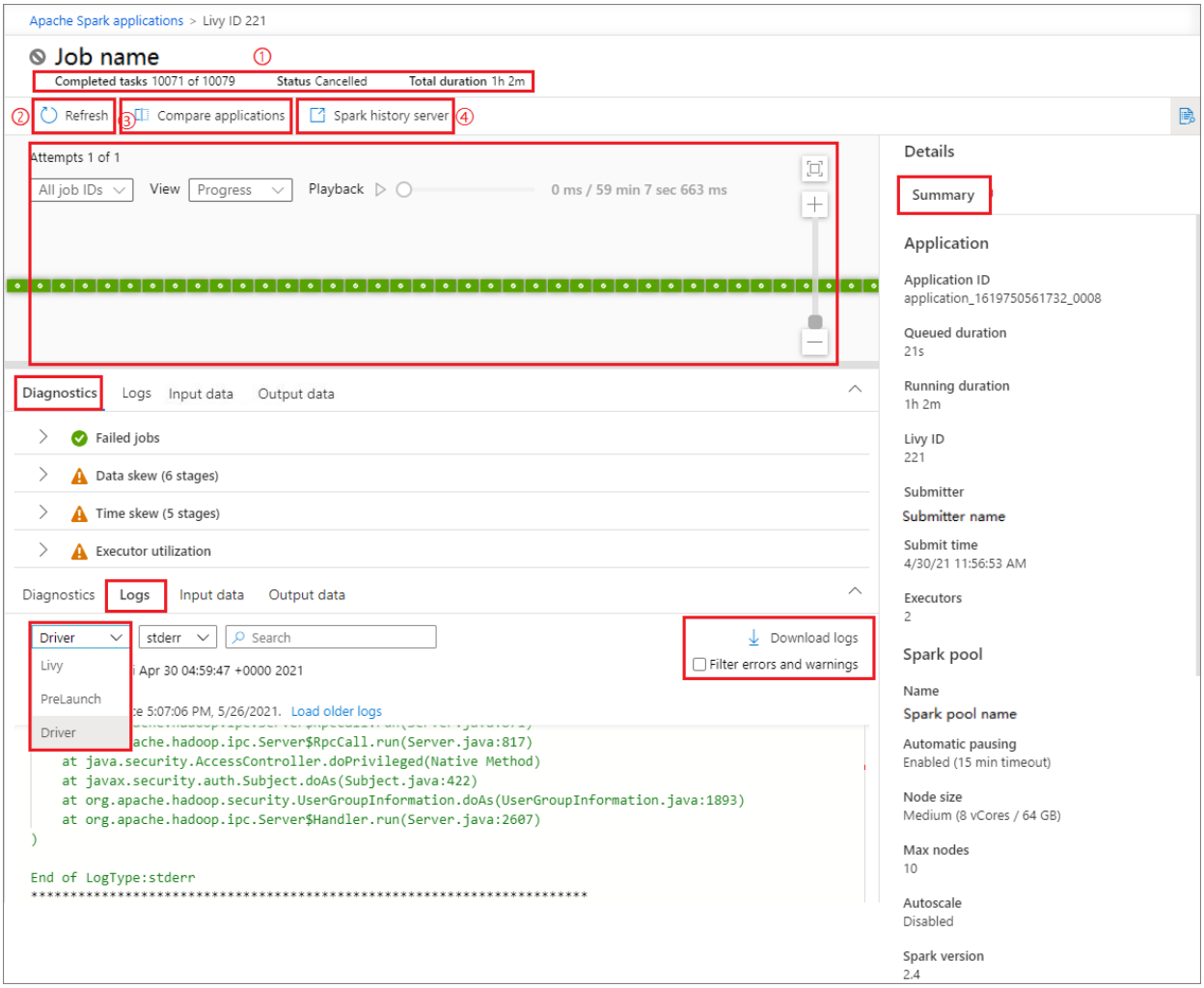
View canceled Apache Spark applications

Open **Monitor**, then select **Apache Spark applications**. To view the details about the canceled Apache Spark applications, select the Apache Spark application.

1. Check the **Completed tasks**, **Status**, and **Total duration**.
2. **Refresh** the Job.
3. Click on **Compare applications** to use the comparison feature, for more information on this feature, see the Compare Apache Spark applications.

4. Open Apache history server link by clicking **Spark history server**.

5. View the graph. You can see an overview of your job in the generated job graph. Refer to steps 5 - 15 of View completed Apache Spark applications.



Debug failed Apache Spark application

Open **Monitor**, then select **Apache Spark applications**. To view the details about the failed Apache Spark applications, select the Apache Spark application.

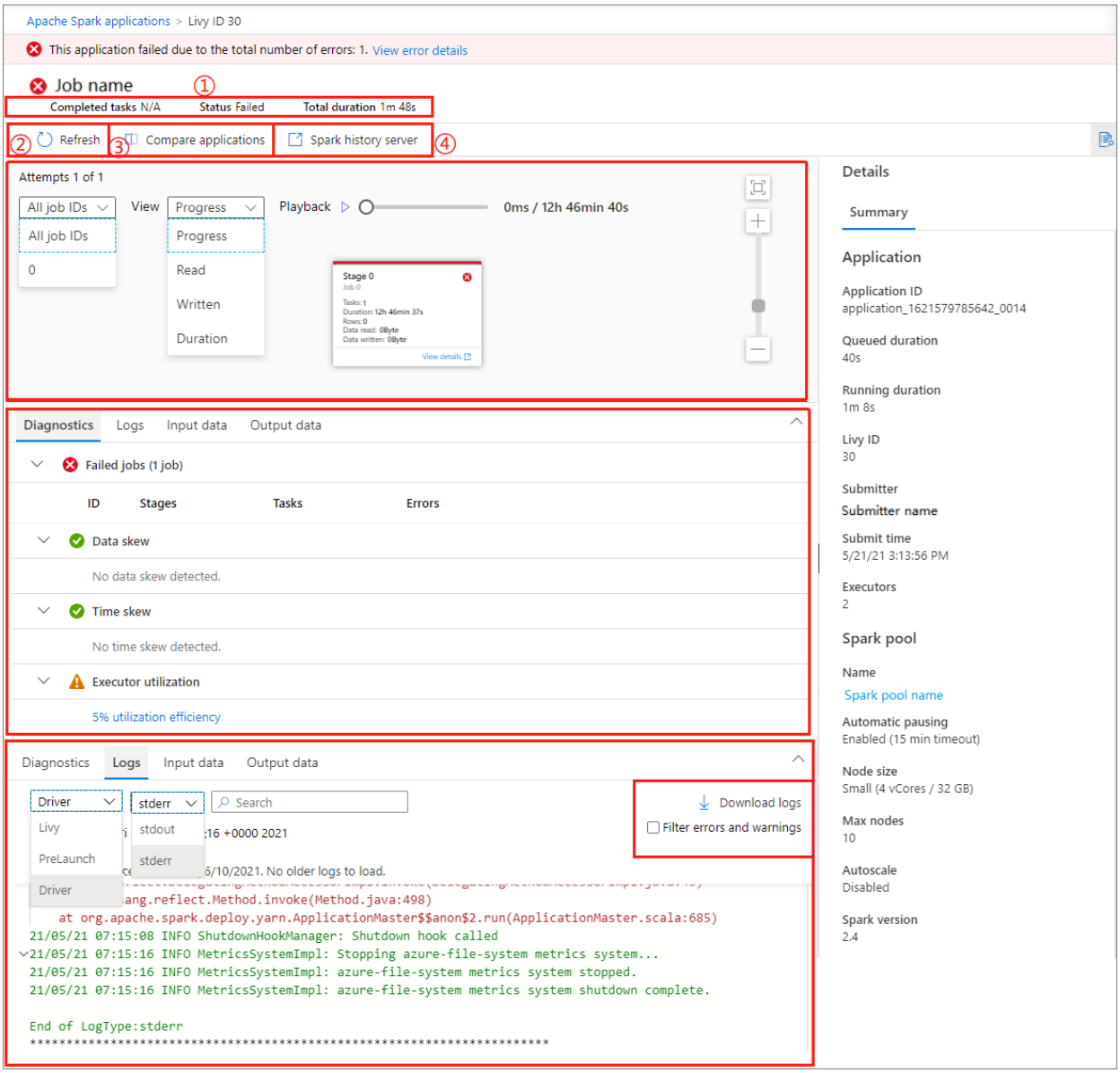
1. Check the **Completed tasks**, **Status**, and **Total duration**.

2. **Refresh** the Job.

3. Click on **Compare applications** to use the comparison feature, for more information on this feature, see the Compare Apache Spark applications.

4. Open Apache history server link by clicking **Spark history server**.

5. View the graph. You can see an overview of your job in the generated job graph. Refer to steps 5 - 15 of View completed Apache Spark applications.

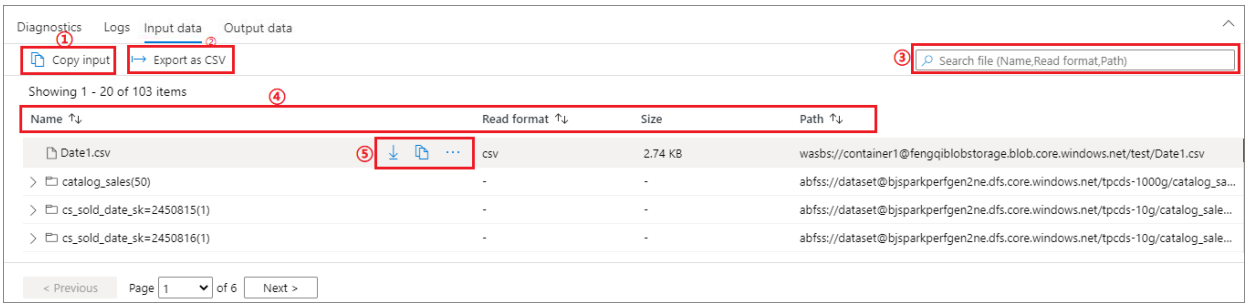


View input data/output data

Select an Apache Spark application, and click on **Input data/Output data tab** to view dates of the input and output for Apache Spark application. This function can help you debug the Spark job. And the data source supports three storage methods: gen1, gen2, and blob.

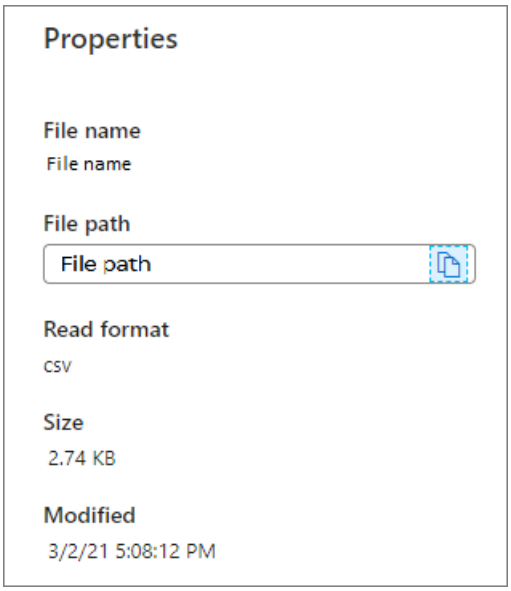
Input data tab

1. Click on the **Copy input** button to paste the input file to the local.
2. Click the **Export to CSV** button to export the input file in CSV format.
3. You can search files by input keywords in **Search box** (keywords include file name, read format and path).
4. You can sort the input files by clicking **Name**, **Read format**, and **path**.
5. Use the mouse to hover over an input file, the icon of the **Download/Copy path/More** button will appear.



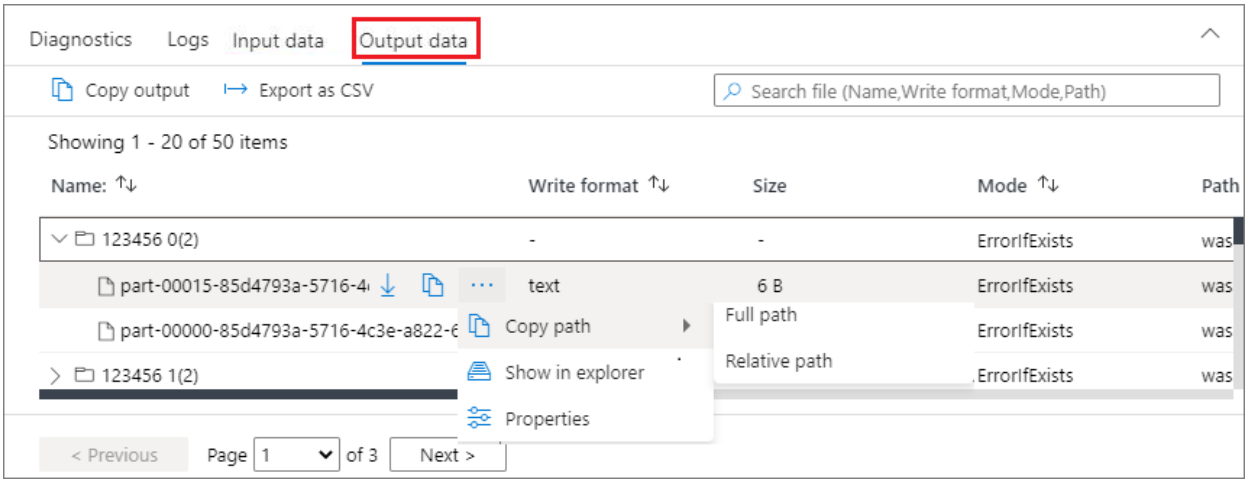
6. Click on **More** button. The **Copy path/Show in explorer/Properties** will appear in the context menu.

- Copy path: can copy **Full path** and **Relative path**.
- Show in explorer: can jump to the linked storage account (Data->Linked).
- Properties: show the basic properties of the file (File name/File path/Read format/Size/Modified).



Output data tab

Displays the same features as the input tab.

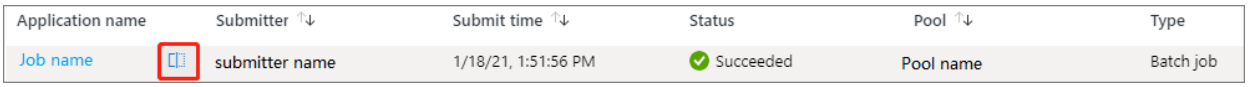


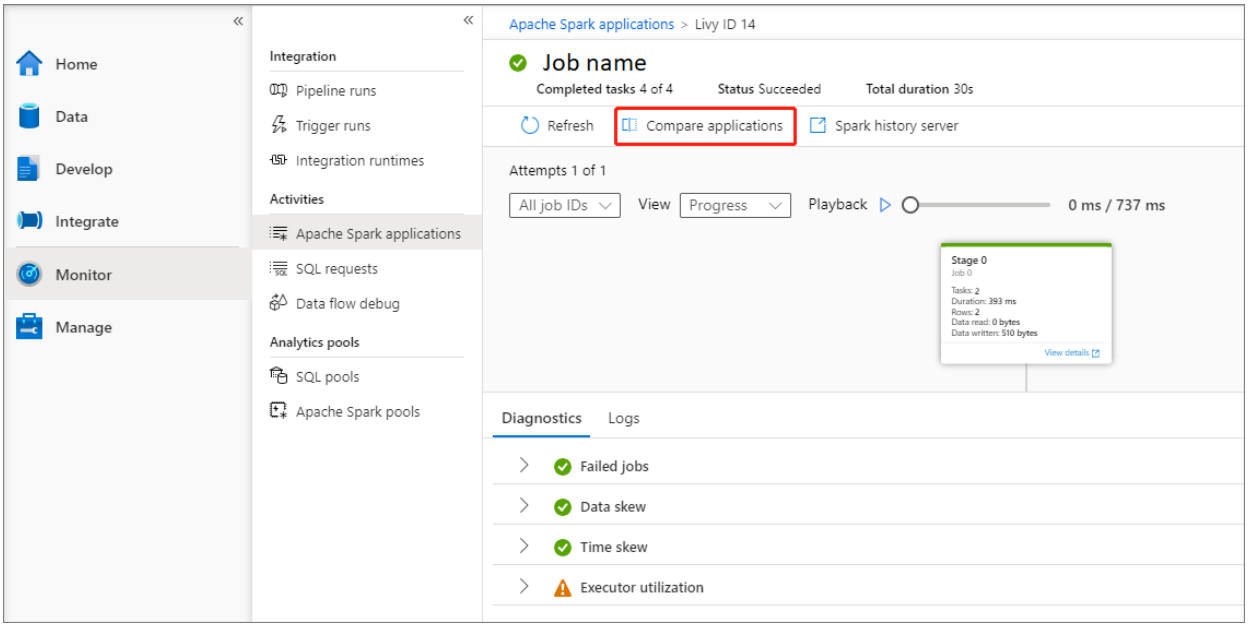
Compare Apache Spark Applications

There are two ways to compare applications. You can compare by choosing **Compare Application**, or click the **Compare in notebook** button to view it in the notebook.

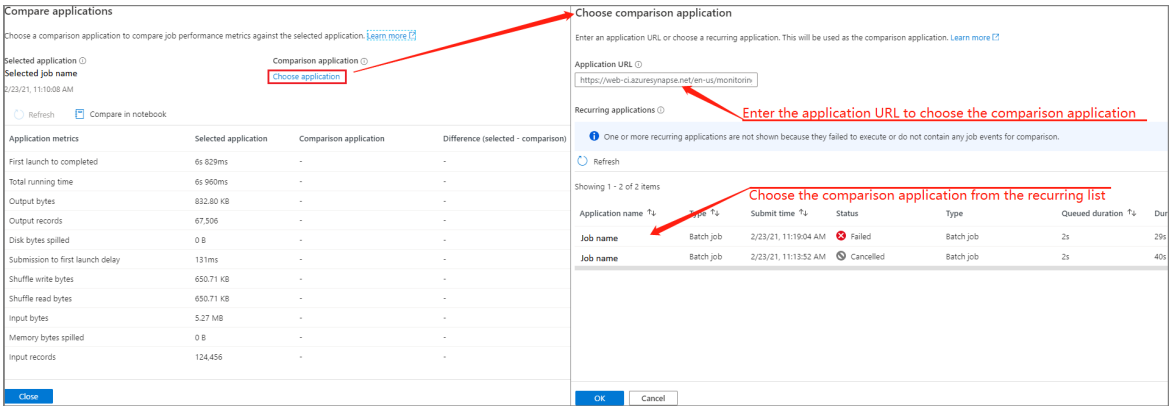
Compare by application

Click on **Compare applications** button and choose an application to compare performance. You can see the difference between the two applications.

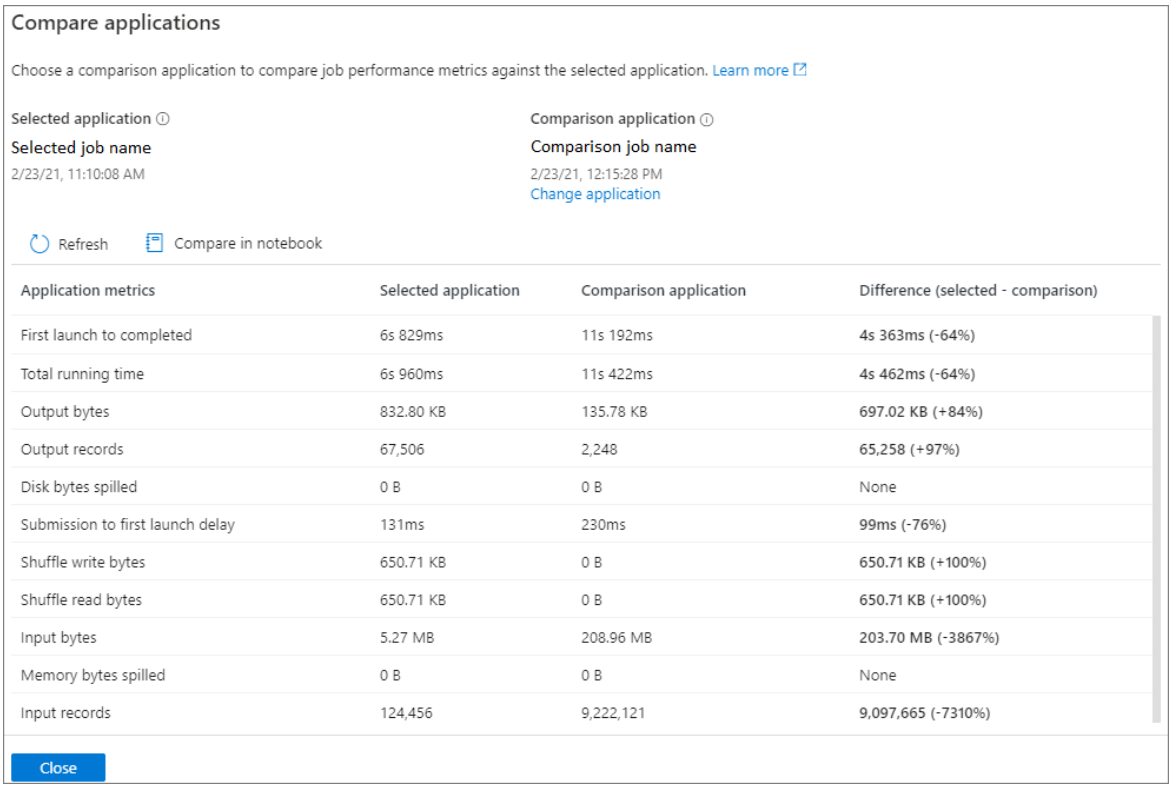




1. Use the mouse to hover over an application, and then the **Compare applications** icon is displayed.
2. Click on the **Compare applications** icon, and the Compare applications page will pop up.
3. Click on **Choose application** button to open **Choose comparison application** page.
4. When choosing the comparison application, you need to either enter the application URL, or choose from the recurring list. Then, click **OK** button.

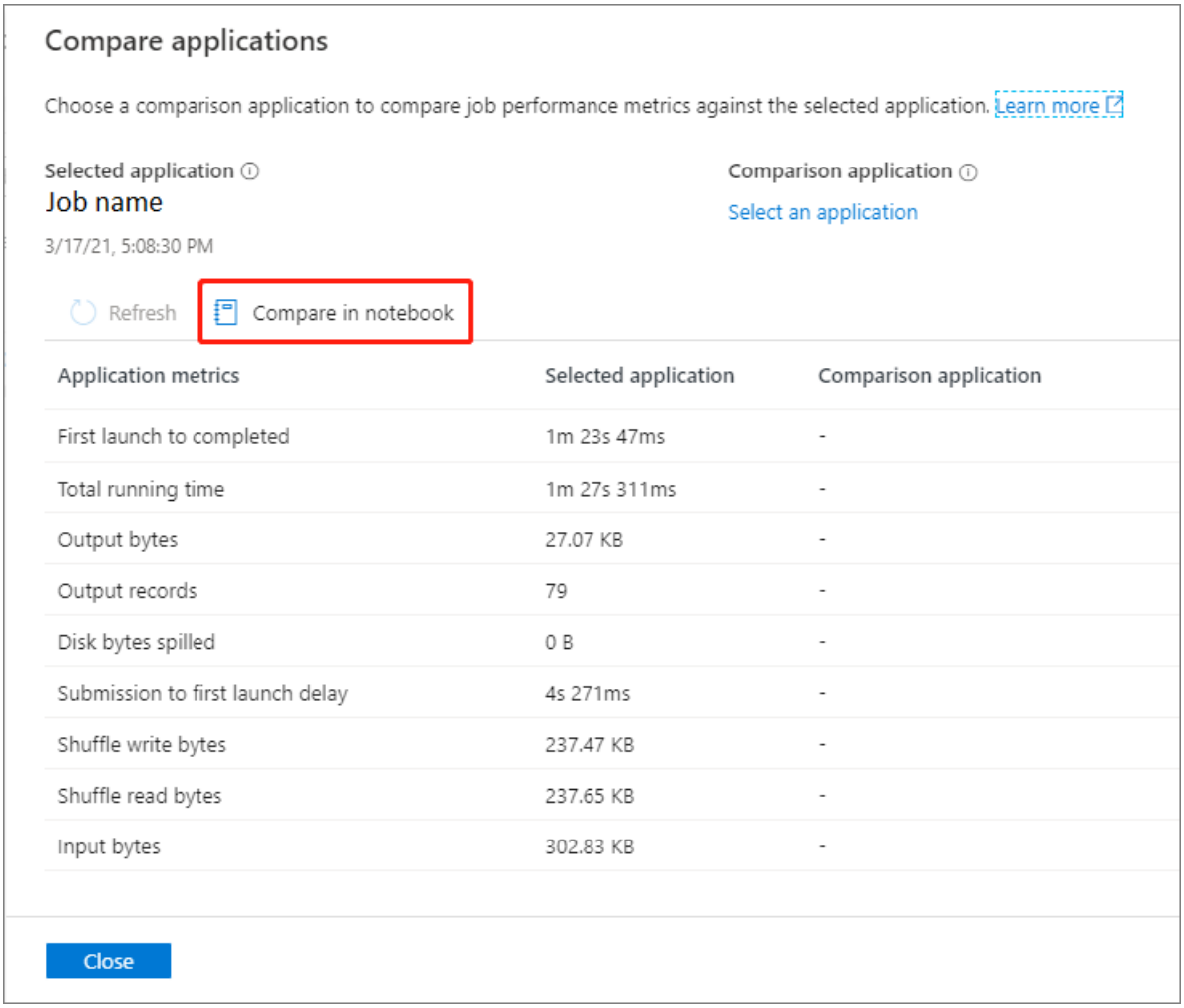


5. The comparison result will be displayed on the compare applications page.

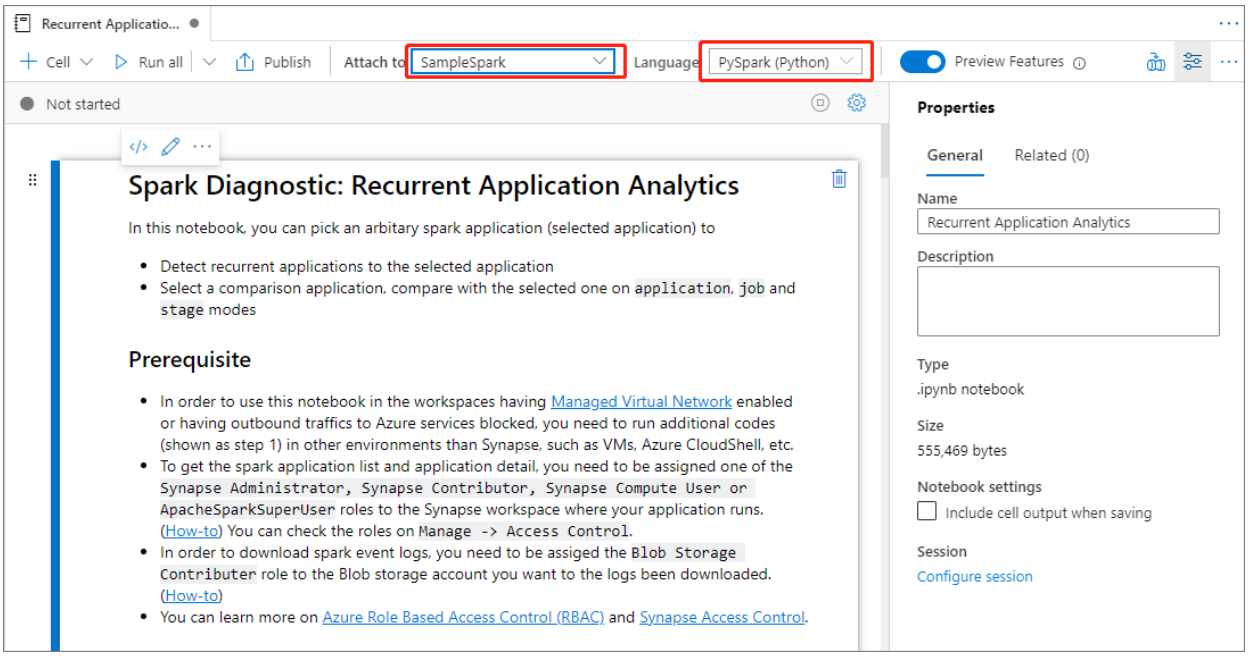


Compare in notebook

Click the **Compare in Notebook** button on the **Compare applications** page to open the notebook. The default name of the *.ipynb* file is **Recurrent Application Analytics**.



In the Notebook: Recurrent Application Analytics file, you can run it directly after setting the Spark pool and Language.



Next steps

For more information on monitoring pipeline runs, see the Monitor pipeline runs using Synapse Studio article.

Use Synapse Studio to monitor your Apache Spark pools

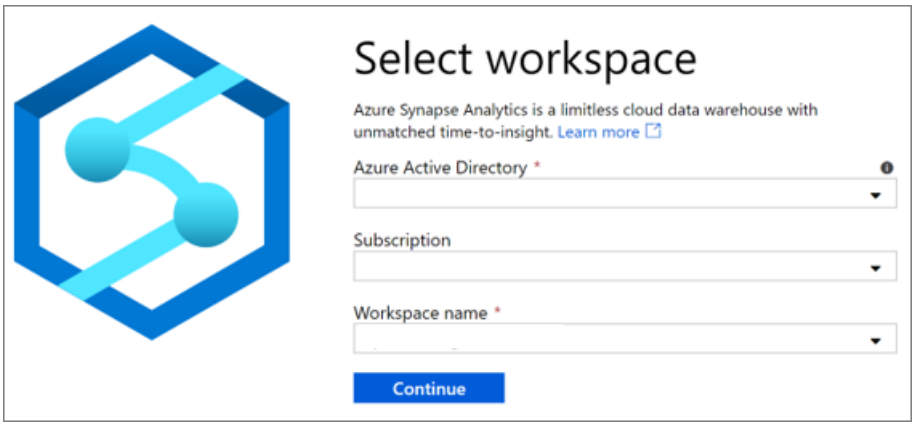
8/19/2022 • 2 minutes to read • Edit Online

With Azure Synapse Analytics, you can use Apache Spark to run notebooks, jobs, and other kinds of applications on Apache Spark pools in your workspace.

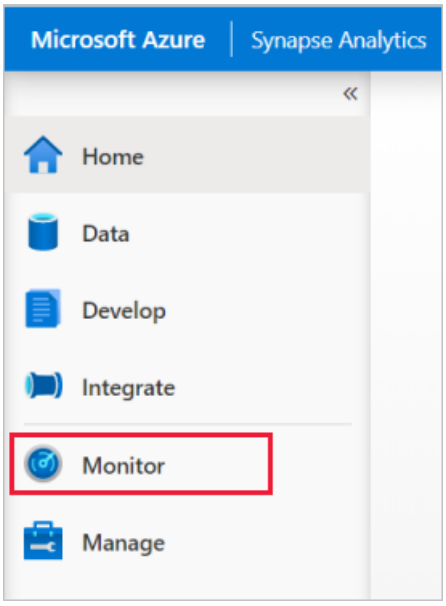
This article explains how to monitor your Apache Spark pools, allowing you to keep an eye on the status of your pools, including how many vCores are in use by different workspace users.

Access Apache Spark pools list

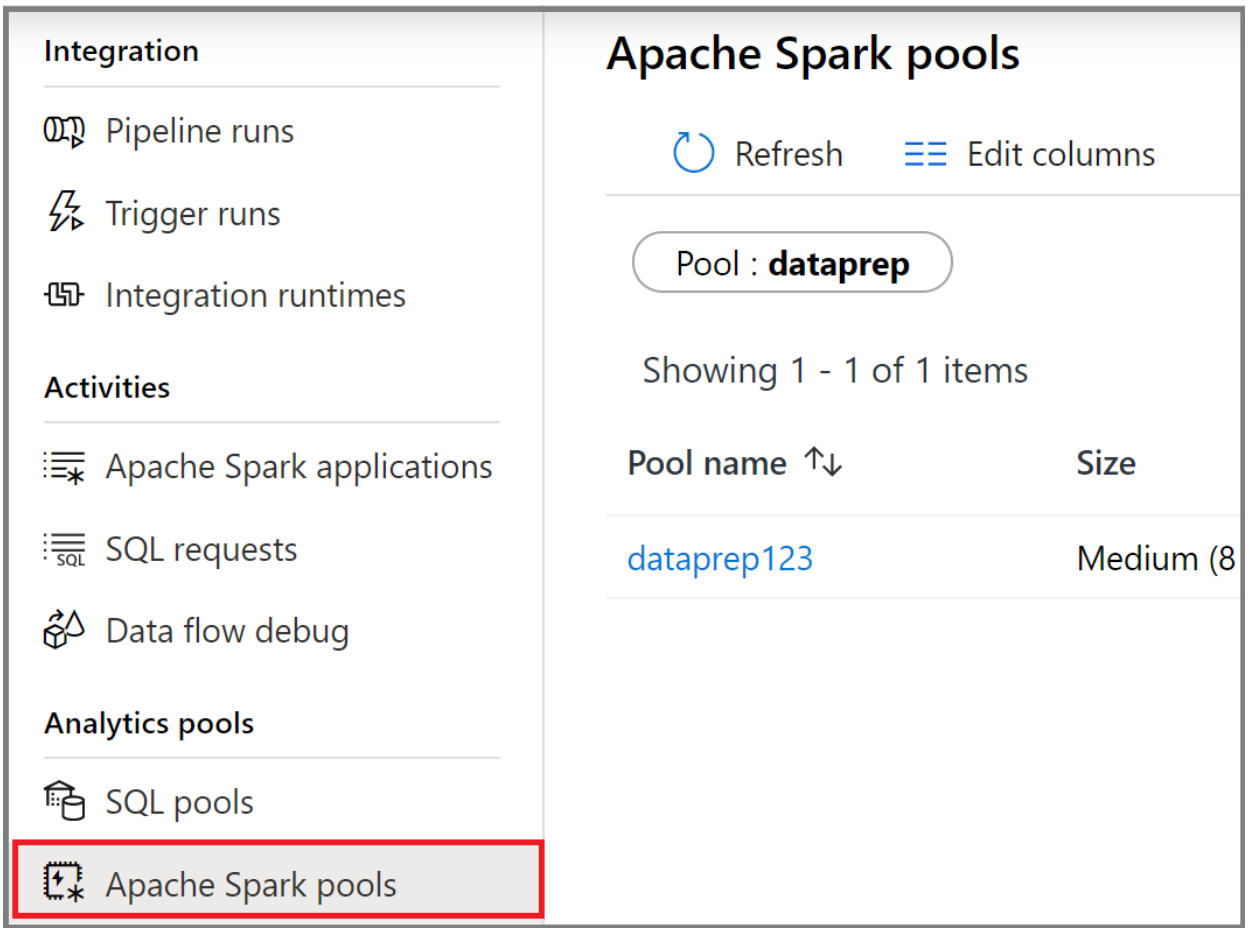
To see the list of Apache Spark pools in your workspace, first open the Synapse Studio and select your workspace.



Once you've opened your workspace, select the **Monitor** section on the left.



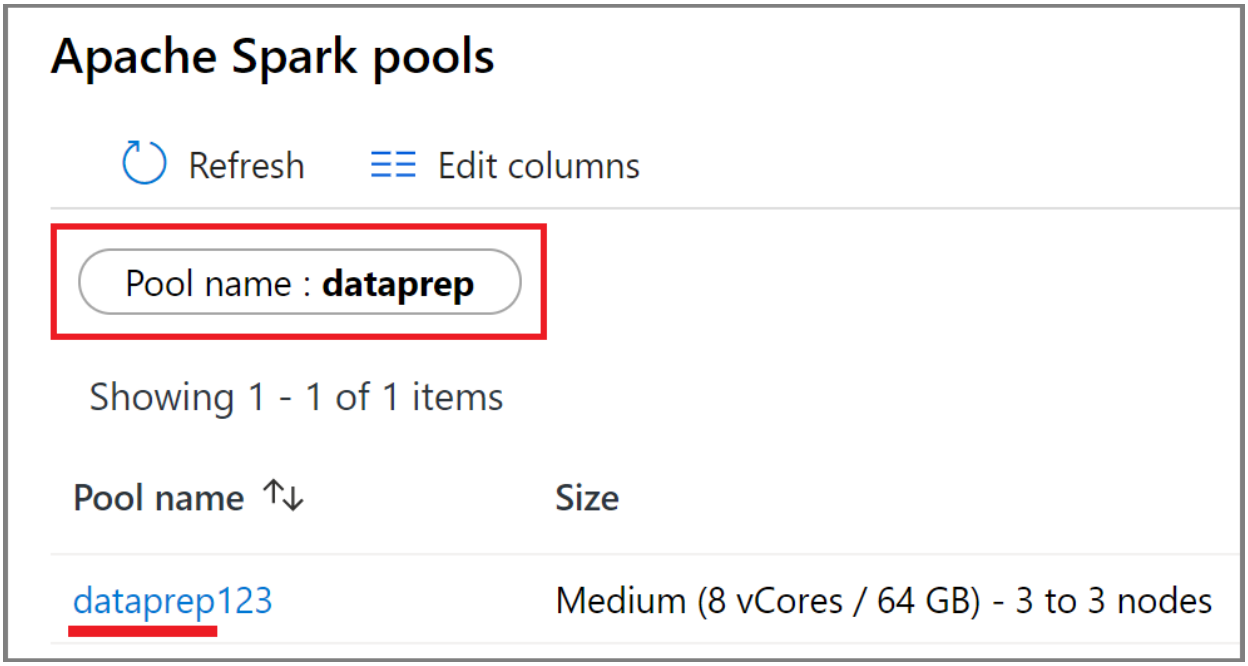
Select **Apache Spark pools** to view the list of Apache Spark pools.



Filter your Apache Spark pools

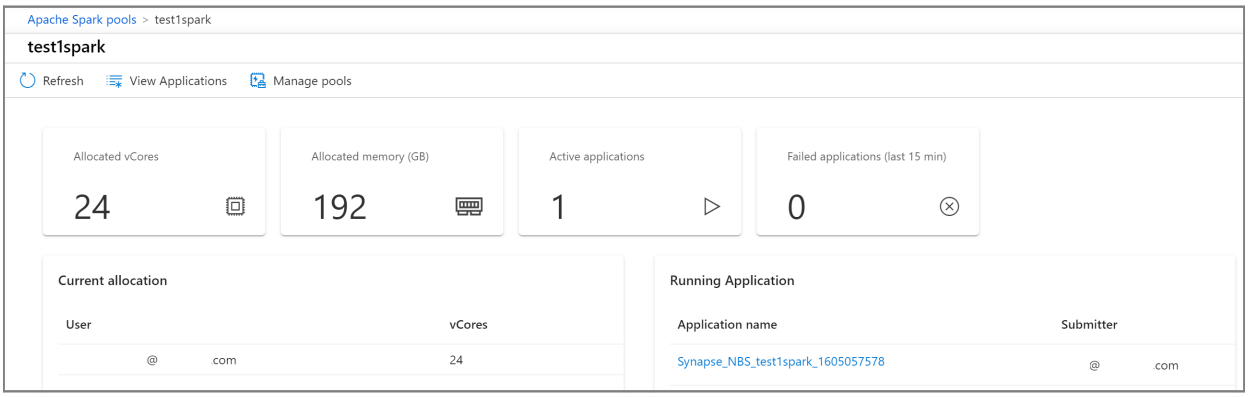
You can filter the list of Apache Spark pools to the ones that interest you. The filters at the top of the screen allow you to specify a field on which you'd like to filter.

For example, you can filter the view to see only the Apache Spark pools containing the name "dataprep":



View details about a specific Apache Spark pool

To view the details about one of your Apache Spark pools, select the Apache Spark pool to view the details.



Next steps

For more information on monitoring pipeline runs, see the Monitor pipeline runs in Synapse Studio article.

For more information on monitoring Apache Spark applications, see the Monitor Apache Spark applications in Synapse Studio article.

Collect Apache Spark applications metrics using APIs

8/19/2022 • 4 minutes to read • Edit Online

Overview

In this tutorial, you will learn how to integrate your existing on-premises Prometheus server with Azure Synapse workspace for near real-time Apache Spark application metrics using the Synapse Prometheus connector.

This tutorial also introduces the Azure Synapse REST metrics APIs. You can fetch Apache Spark application metrics data through the REST APIs to build your own monitoring and diagnosis toolkit or integrate with your monitoring systems.

Use Azure Synapse Prometheus connector for your on-premises Prometheus servers

Azure Synapse Prometheus connector is an open-source project. The Synapse Prometheus connector uses a file-based service discovery method to allow you to:

- Authenticate to Synapse workspace via an AAD service principal.
- Fetch workspace Apache Spark applications list.
- Pull Apache Spark application metrics through Prometheus file-based configuration.

1. Prerequisite

You need to have a Prometheus server deployed on a Linux VM.

2. Create a service principal

To use the Azure Synapse Prometheus connector in your on-premises Prometheus server, you should follow the steps below to create a service principal.

2.1 Create a service principal:

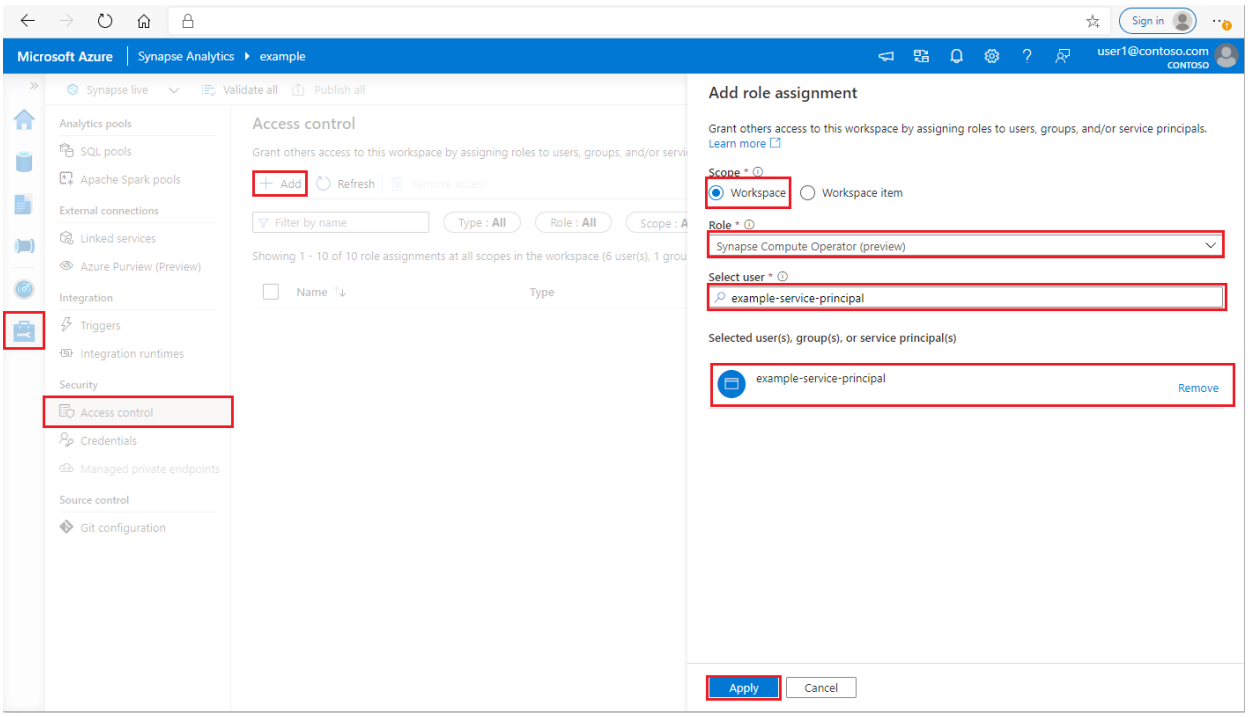
```
az ad sp create-for-rbac --name <service_principal_name> --role Contributor --scopes
/subscriptions/<subscription_id>
```

The result should look like:

```
{
  "appId": "abcdef...",
  "displayName": "<service_principal_name>",
  "name": "http://<service_principal_name>",
  "password": "abc....",
  "tenant": "<tenant_id>"
}
```

Note down the appId, password, and tenant ID.

2.2 Add corresponding permissions to the service principal created in the above step.



1. Log in to your Azure Synapse Analytics workspace as Synapse Administrator
2. In Synapse Studio, on the left-side pane, select **Manage > Access control**
3. Click the Add button on the upper left to **add a role assignment**
4. For Scope, choose **Workspace**
5. For Role, choose **Synapse Compute Operator**
6. For Select user, input your **<service_principal_name>** and click your service principal
7. Click **Apply** (Wait 3 minutes for permission to take effect.)

3. Download the Azure Synapse Prometheus Connector

Use the commands to install the Azure Synapse Prometheus Connector.

```
git clone https://github.com/microsoft/azure-synapse-spark-metrics.git
cd ./azure-synapse-spark-metrics/synapse-prometheus-connector/src
python pip install -r requirements.txt
```

4. Create a config file for Azure Synapse workspaces

Create a config.yaml file in config folder, and fill in the following fields: workspace_name, tenant_id, service_principal_name and service_principal_password. You can add multiple workspaces in the yaml config.

```
workspaces:
  - workspace_name: <your_workspace_name>
    tenant_id: <tenant_id>
    service_principal_name: <service_principal_app_id>
    service_principal_password: "<service_principal_password>"
```

5. Update the Prometheus config

Add following config section in your Prometheus scrape_config, and replace the <your_workspace_name> to your workspace name and the <path_to_synapse_connector> to your cloned synapse-prometheus-connector folder

```
- job_name: synapse-prometheus-connector
  static_configs:
  - labels:
      __metrics_path__: /metrics
      __scheme__: http
    targets:
    - localhost:8000
- job_name: synapse-workspace-<your_workspace_name>
  bearer_token_file: <path_to_synapse_connector>/output/workspace/<your_workspace_name>/bearer_token
  file_sd_configs:
  - files:
    - <path_to_synapse_connector>/output/workspace/<your_workspace_name>/application_discovery.json
    refresh_interval: 10s
  metric_relabel_configs:
  - source_labels: [ __name__ ]
    target_label: __name__
    regex: metrics_application_[0-9]+_[0-9]+_(.+)
    replacement: spark_$1
  - source_labels: [ __name__ ]
    target_label: __name__
    regex: metrics_(.+)
    replacement: spark_$1
```

6. Start the connector in the Prometheus server VM

Start a connector server in the Prometheus server VM as follows.

```
python main.py
```

Wait for a few seconds and the connector should start working. And you can see the "synapse-prometheus-connector" in the Prometheus service discovery page.

Use Azure Synapse Prometheus or REST metrics APIs to collect metrics data

1. Authentication

You can use the client credentials flow to get an access token. To access the metrics API, you should get an Azure AD access token for the service principal, which has proper permission to access the APIs.

PARAMETER	REQUIRED	DESCRIPTION
tenant_id	True	Your Azure service principal (application) tenant ID
grant_type	True	Specifies the requested grant type. In a Client Credentials Grant flow, the value must be client_credentials.
client_id	True	The application (service principal) ID of the application you registered in Azure portal or Azure CLI.
client_secret	True	The secret generated for the application (service principal)
resource	True	Synapse resource uri, should be 'https://dev.azuresynapse.net'

```
curl -X GET -H 'Content-Type: application/x-www-form-urlencoded' \
  -d 'grant_type=client_credentials&client_id=<service_principal_app_id>&resource=
<azure_synapse_resource_id>&client_secret=<service_principal_secret>' \
  https://login.microsoftonline.com/<tenant_id>/oauth2/token
```

Response looks like:

```
{
  "token_type": "Bearer",
  "expires_in": "599",
  "ext_expires_in": "599",
  "expires_on": "1575500666",
  "not_before": "1575499766",
  "resource": "2ff8...f879c1d",
  "access_token": "ABC0eXAiOiJKV1Q......un_f1mSgCHlA"
}
```

2. List running applications in the Azure Synapse workspace

To get list of Apache Spark applications for a Synapse workspace, you can follow this document Monitoring - Get Apache Spark Job List.

3. Collect Apache Spark application metrics with the Prometheus or REST APIs

Collect Apache Spark application metrics with the Prometheus API

Get latest metrics of the specified Apache Spark application by Prometheus API

```
GET
https://{endpoint}/livyApi/versions/{livyApiVersion}/sparkpools/{sparkPoolName}/sessions/{sessionId}/applica
tions/{sparkApplicationId}/metrics/executors/prometheus?format=html
```

PARAMETER	REQUIRED	DESCRIPTION
endpoint	True	The workspace development endpoint, for example `https://myworkspace.dev.azuresynapse.net.`
livyApiVersion	True	Valid api-version for the request. Currently, it is 2019-11-01-preview
sparkPoolName	True	Name of the spark pool.
sessionId	True	Identifier for the session.
sparkApplicationId	True	Spark Application ID

Sample Request:

```
GET https://myworkspace.dev.azuresynapse.net/livyApi/versions/2019-11-01-
preview/sparkpools/mysparkpool/sessions/1/applications/application_1605509647837_0001/metrics/executors/prom
etheus?format=html
```

Sample Response:

Status code: 200 Response looks like:

```
metrics_executor_rddBlocks{application_id="application_1605509647837_0001",
application_name="mynotebook_mysparkpool_1605509570802", executor_id="driver"} 0
metrics_executor_memoryUsed_bytes{application_id="application_1605509647837_0001",
application_name="mynotebook_mysparkpool_1605509570802", executor_id="driver"} 74992
metrics_executor_diskUsed_bytes{application_id="application_1605509647837_0001",
application_name="mynotebook_mysparkpool_1605509570802", executor_id="driver"} 0
metrics_executor_totalCores{application_id="application_1605509647837_0001",
application_name="mynotebook_mysparkpool_1605509570802", executor_id="driver"} 0
metrics_executor_maxTasks{application_id="application_1605509647837_0001",
application_name="mynotebook_mysparkpool_1605509570802", executor_id="driver"} 0
metrics_executor_activeTasks{application_id="application_1605509647837_0001",
application_name="mynotebook_mysparkpool_1605509570802", executor_id="driver"} 1
metrics_executor_failedTasks_total{application_id="application_1605509647837_0001",
application_name="mynotebook_mysparkpool_1605509570802", executor_id="driver"} 0
metrics_executor_completedTasks_total{application_id="application_1605509647837_0001",
application_name="mynotebook_mysparkpool_1605509570802", executor_id="driver"} 2
...
```

Collect Apache Spark application metrics with the REST API

```
GET
https://{endpoint}/livyApi/versions/{livyApiVersion}/sparkpools/{sparkPoolName}/sessions/{sessionId}/applica
tions/{sparkApplicationId}/executors
```

PARAMETER	REQUIRED	DESCRIPTION
endpoint	True	The workspace development endpoint, for example `https://myworkspace.dev.azuresynapse.net.`
livyApiVersion	True	Valid api-version for the request. Currently, it is 2019-11-01-preview
sparkPoolName	True	Name of the spark pool.
sessionId	True	Identifier for the session.
sparkApplicationId	True	Spark Application ID

Sample Request

```
GET https://myworkspace.dev.azuresynapse.net/livyApi/versions/2019-11-01-
preview/sparkpools/mysparkpool/sessions/1/applications/application_1605509647837_0001/executors
```

Sample Response Status code: 200

```
[
    {
        "id": "driver",
        "hostPort": "f98b8fc2aea84e9095bf2616208eb672007bde57624:45889",
        "isActive": true,
        "rddBlocks": 0,
        "memoryUsed": 75014,
        "diskUsed": 0,
        "totalCores": 0,
        "maxTasks": 0,
        "activeTasks": 0,
        "failedTasks": 0,
        "completedTasks": 0,
        "totalTasks": 0,
        "totalDuration": 0,
        "totalGCTime": 0,
        "totalInputBytes": 0,
        "totalShuffleRead": 0,
        "totalShuffleWrite": 0,
        "isBlacklisted": false,
        "maxMemory": 15845975654,
        "addTime": "2020-11-16T06:55:06.718GMT",
        "executorLogs": {
            "stdout":
"http://f98b8fc2aea84e9095bf2616208eb672007bde57624:8042/node/containerlogs/container_1605509647837_0001_01_
000001/trusted-service-user/stdout?start=-4096",
            "stderr":
"http://f98b8fc2aea84e9095bf2616208eb672007bde57624:8042/node/containerlogs/container_1605509647837_0001_01_
000001/trusted-service-user/stderr?start=-4096"
        },
        "memoryMetrics": {
            "usedOnHeapStorageMemory": 75014,
            "usedOffHeapStorageMemory": 0,
            "totalOnHeapStorageMemory": 15845975654,
            "totalOffHeapStorageMemory": 0
        },
        "blacklistedInStages": []
    },
    // ...
]
```


4. Build your own diagnosis and monitoring tools

The Prometheus API and the REST APIs provide rich metrics data about the Apache Spark application running information.You can collect the application-related metrics data through the Prometheus API and the REST APIs. And build your own diagnosis and monitoring tools that are more suitable for your needs.

Monitor Apache Spark Applications metrics with Prometheus and Grafana

8/19/2022 • 4 minutes to read • Edit Online

Overview

In this tutorial, you will learn how to deploy the Apache Spark application metrics solution to an Azure Kubernetes Service (AKS) cluster and learn how to integrate the Grafana dashboards.

You can use this solution to collect and query the Apache Spark metrics data near real time. The integrated Grafana dashboards allow you to diagnose and monitor your Apache Spark application. The source code and the configurations have been open-sourced on GitHub.

Prerequisites

1. Azure CLI
2. Helm client 3.30+
3. kubectl
4. Azure Kubernetes Service (AKS)

Or use the Azure Cloud Shell, which already includes the Azure CLI, Helm client and kubectl out of the box.

Log in to Azure

```
az login
az account set --subscription "<subscription_id>"
```

Create an Azure Kubernetes service instance (AKS)

Use the Azure CLI command to create a Kubernetes cluster in your subscription.

```
az aks create --name <kubernetes_name> --resource-group <kubernetes_resource_group> --location <location> --
node-vm-size Standard_D2s_v3
az aks get-credentials --name <kubernetes_name> --resource-group <kubernetes_resource_group>
```

Note: This step can be skipped if you already have an AKS cluster.

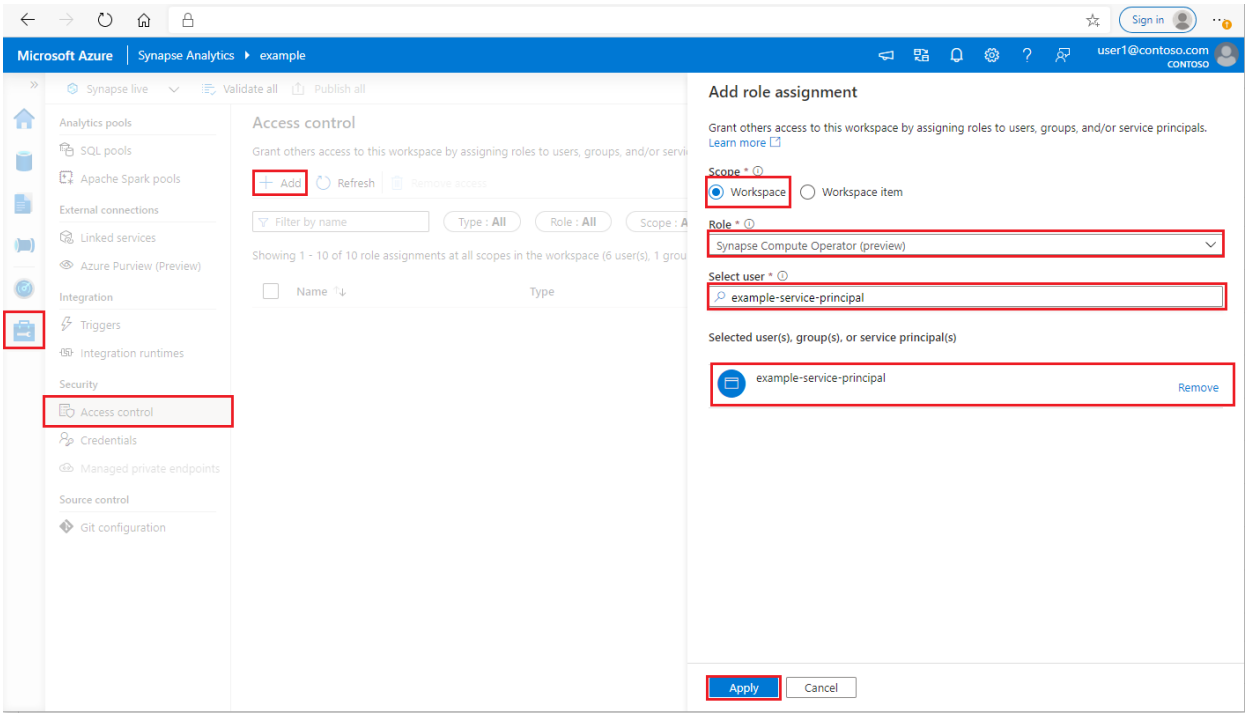
Create a service principal and grant permission to Synapse workspace

```
az ad sp create-for-rbac --name <service_principal_name> --role Contributor --scopes
/subscriptions/<subscription_id>
```

The result should look like:

```
{
  "appId": "abcdef...",
  "displayName": "<service_principal_name>",
  "name": "http://<service_principal_name>",
  "password": "abc....",
  "tenant": "<tenant_id>"
}
```

Note down the appId, password, and tenantID.



1. Log in to your Azure Synapse Analytics workspace as Synapse Administrator
2. In Synapse Studio, on the left-side pane, select **Manage > Access control**
3. Click the Add button on the upper left to **add a role assignment**
4. For Scope, choose **Workspace**
5. For Role, choose **Synapse Compute Operator**
6. For Select user, input your **<service_principal_name>** and click your service principal
7. Click **Apply** (Wait 3 minutes for permission to take effect.)

> **NOTE**
>
> Please make sure your service principal is at least "Reader" role in your Synapse workspace. Go to Access Control (IAM) tab of the Azure portal and check the permission settings.

Install connector, Prometheus server, Grafana dashboard

1. Add synapse-charts repo to Helm client.

```
helm repo add synapse-charts https://github.com/microsoft/azure-synapse-spark-metrics/releases/download/helm-chart
```

2. Install components through Helm client:

```
helm install spo synapse-charts/synapse-prometheus-operator --create-namespace --namespace spo \
    --set synapse.workspaces[0].workspace_name="<workspace_name>" \
    --set synapse.workspaces[0].tenant_id="<tenant_id>" \
    --set synapse.workspaces[0].service_principal_name="<service_principal_app_id>" \
    --set synapse.workspaces[0].service_principal_password="<service_principal_password>" \
    --set synapse.workspaces[0].subscription_id="<subscription_id>" \
    --set synapse.workspaces[0].resource_group="<workspace_resource_group_name>"
```

- workspace_name: Synapse workspace name.
- subscription_id: Synapse workspace subscription ID.
- workspace_resource_group_name: Synapse workspace resource group name.
- tenant_id: Synapse workspace tenant ID.
- service_principal_app_id: The service principal "appId"
- service_principal_password: The service principal password you created.

Log in to Grafana

Get the default password and address of Grafana. You may change the password in the Grafana settings.

```
kubectl get secret --namespace spo spo-grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
kubectl -n spo get svc spo-grafana
```

Get service ip, copy & paste the external ip to browser, and login with username "admin" and the password.

Use Grafana Dashboards

Find Synapse Dashboard on the upper left corner of the Grafana page (Home -> Synapse Workspace / Synapse Application), try to run an example code in Synapse Studio and wait a few seconds for the metrics pulling.

Also, you can use the "Synapse Workspace / Workspace" and "Synapse Workspace / Apache Spark pools" dashboards to get an overview of your workspace and your Apache Spark pools.

Uninstall

Remove the components by Helm command as follows.

```
helm delete <release_name> -n <namespace>
```

Delete the AKS cluster.

```
az aks delete --name <kubernetes_cluster_name> --resource-group <kubernetes_cluster_rg>
```

Components introduction

Azure Synapse Analytics provides a Helm chart based on Prometheus Operator and Synapse Prometheus Connector. The Helm chart includes Prometheus server, Grafana server, and Grafana dashboards for Apache Spark application-level metrics. You can use Prometheus, a popular open-source monitoring system, to collect these metrics in near real-time and use Grafana for visualization.

Synapse Prometheus Connector

Synapse Prometheus Connector helps to connect Azure Synapse Apache Spark pool and your Prometheus server. It implements:

1. Authentication: It is AAD-based authentication and can automatically refresh the AAD token of the service principal for application discovery, metrics ingestion and other functions.
2. Apache Spark application discovery: When you submit applications in the target workspace, Synapse Prometheus Connector can automatically discover these applications.
3. Apache Spark application metadata: It collects basic application information and exports the data to Prometheus.

Synapse Prometheus Connector is released as a docker image hosted on Microsoft Container Registry. It is open-source and is located in Azure Synapse Apache Spark application metrics.

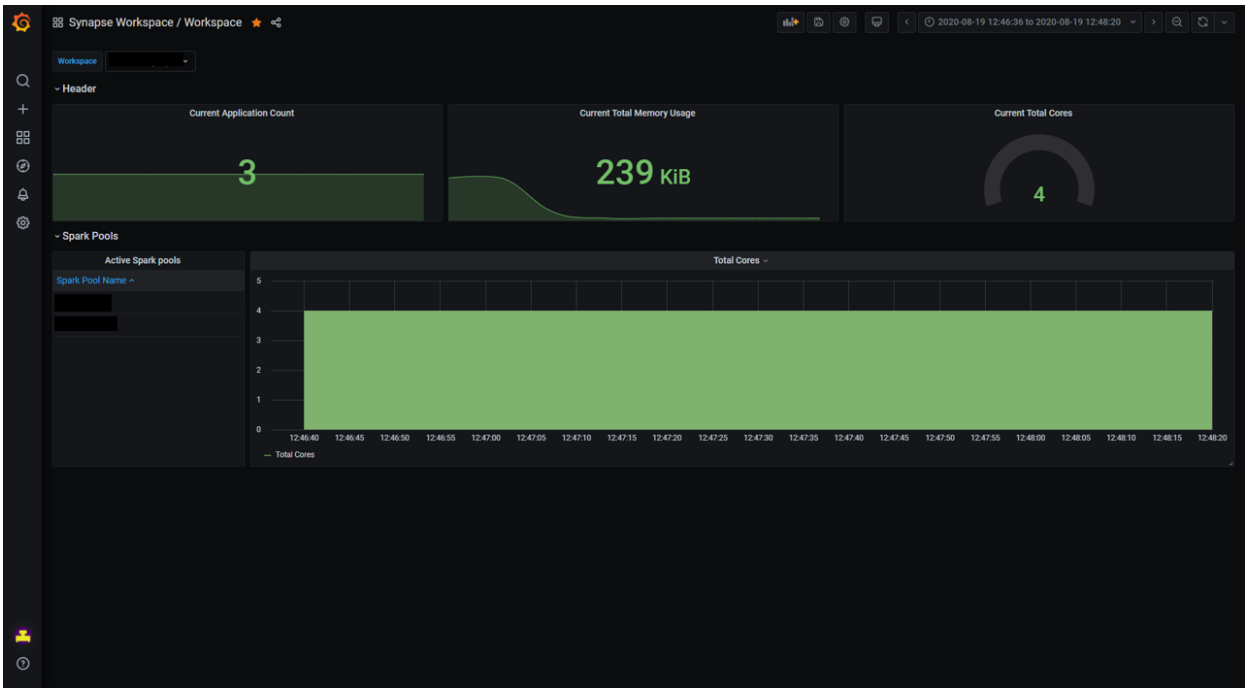
Prometheus server

Prometheus is an open-source monitoring and alerting toolkit. Prometheus graduated from the Cloud Native Computing Foundation (CNCF) and became the de facto standard for cloud-native monitoring. Prometheus can help us collect, query, and store massive amounts of time series data, and it can be easily integrated with Grafana. In this solution, we deploy the Prometheus component based on the helm chart.

Grafana and dashboards

Grafana is open-source visualization and analytics software. It allows you to query, visualize, alert on, and explore your metrics. Azure Synapse Analytics provides a set of default Grafana dashboards to visualize Apache Spark application-level metrics.

The "Synapse Workspace / Workspace" dashboard provides a workspace level view of all the Apache Spark pools, application counts, cpu cores, etc.



The "Synapse Workspace / Apache Spark pools" dashboard contains the metrics of Apache Spark applications running in the selected Apache Spark pool during the time period.

The "Synapse Workspace / Apache Spark Application" dashboard contains the selected Apache Spark application.

The above dashboard templates have been open-sourced in Azure Synapse Apache Spark application metrics.

Monitor Apache Spark applications with Azure Log Analytics

8/19/2022 • 8 minutes to read • Edit Online

In this tutorial, you learn how to enable the Synapse Studio connector that's built in to Log Analytics. You can then collect and send Apache Spark application metrics and logs to your Log Analytics workspace. Finally, you can use an Azure Monitor workbook to visualize the metrics and logs.

Configure workspace information

Follow these steps to configure the necessary information in Synapse Studio.

Step 1: Create a Log Analytics workspace

Consult one of the following resources to create this workspace:

- Create a workspace in the Azure portal.
- Create a workspace with Azure CLI.
- Create and configure a workspace in Azure Monitor by using PowerShell.

Step 2: Prepare an Apache Spark configuration file

Use any of the following options to prepare the file.

Option 1: Configure with Log Analytics workspace ID and key

Copy the following Apache Spark configuration, save it as *spark_loganalytics_conf.txt*, and fill in the following parameters:

- `<LOG_ANALYTICS_WORKSPACE_ID>`: Log Analytics workspace ID.
- `<LOG_ANALYTICS_WORKSPACE_KEY>`: Log Analytics key. To find this, in the Azure portal, go to **Azure Log Analytics workspace** > **Agents management** > **Primary key**.

```
spark.synapse.logAnalytics.enabled true
spark.synapse.logAnalytics.workspaceId <LOG_ANALYTICS_WORKSPACE_ID>
spark.synapse.logAnalytics.secret <LOG_ANALYTICS_WORKSPACE_KEY>
```

Option 2: Configure with Azure Key Vault

> **NOTE**
> You need to grant read secret permission to the users who will submit Apache Spark applications. For more information, see Provide access to Key Vault keys, certificates, and secrets with an Azure role-based access control. When you enable this feature in a Synapse pipeline, you need to use **Option 3**. This is necessary to obtain the secret from Azure Key Vault with workspace managed identity.

To configure Azure Key Vault to store the workspace key, follow these steps:

1. Create and go to your key vault in the Azure portal.
2. On the settings page for the key vault, select **Secrets**.
3. Select **Generate/Import**.
4. On the **Create a secret** screen, choose the following values:
   - **Name**: Enter a name for the secret. For the default, enter `SparkLogAnalyticsSecret`.
   - **Value**: Enter the `<LOG_ANALYTICS_WORKSPACE_KEY>` for the secret.

- Leave the other values to their defaults. Then select **Create**.

5. Copy the following Apache Spark configuration, save it as *spark_loganalytics_conf.txt*, and fill in the following parameters:

   - `<LOG_ANALYTICS_WORKSPACE_ID>`: The Log Analytics workspace ID.
   - `<AZURE_KEY_VAULT_NAME>`: The key vault name that you configured.
   - `<AZURE_KEY_VAULT_SECRET_KEY_NAME>` (optional): The secret name in the key vault for the workspace key. The default is `SparkLogAnalyticsSecret`.

```
spark.synapse.logAnalytics.enabled true
spark.synapse.logAnalytics.workspaceId <LOG_ANALYTICS_WORKSPACE_ID>
spark.synapse.logAnalytics.keyVault.name <AZURE_KEY_VAULT_NAME>
spark.synapse.logAnalytics.keyVault.key.secret <AZURE_KEY_VAULT_SECRET_KEY_NAME>
```

> **NOTE**
>
> You can also store the workspace ID in Key Vault. Refer to the preceding steps, and store the workspace ID with the secret name `SparkLogAnalyticsWorkspaceId`. Alternatively, you can use the configuration `spark.synapse.logAnalytics.keyVault.key.workspaceId` to specify the workspace ID secret name in Key Vault.

Option 3. Configure with a linked service

> **NOTE**
>
> In this option, you need to grant read secret permission to workspace managed identity. For more information, see Provide access to Key Vault keys, certificates, and secrets with an Azure role-based access control.

To configure a Key Vault linked service in Synapse Studio to store the workspace key, follow these steps:

1. Follow all the steps in the preceding section, "Option 2."
2. Create a Key Vault linked service in Synapse Studio:

   a. Go to **Synapse Studio** > **Manage** > **Linked services**, and then select **New**.

   b. In the search box, search for **Azure Key Vault**.

   c. Enter a name for the linked service.

   d. Choose your key vault, and select **Create**.

3. Add a `spark.synapse.logAnalytics.keyVault.linkedServiceName` item to the Apache Spark configuration.

```
spark.synapse.logAnalytics.enabled true
spark.synapse.logAnalytics.workspaceId <LOG_ANALYTICS_WORKSPACE_ID>
spark.synapse.logAnalytics.keyVault.name <AZURE_KEY_VAULT_NAME>
spark.synapse.logAnalytics.keyVault.key.secret <AZURE_KEY_VAULT_SECRET_KEY_NAME>
spark.synapse.logAnalytics.keyVault.linkedServiceName <LINKED_SERVICE_NAME>
```

Available Apache Spark configuration

CONFIGURATION NAME	DEFAULT VALUE	DESCRIPTION
spark.synapse.logAnalytics.enabled	false	To enable the Log Analytics sink for the Spark applications, true. Otherwise, false.
spark.synapse.logAnalytics.workspaceId	-	The destination Log Analytics workspace ID.

CONFIGURATION NAME	DEFAULT VALUE	DESCRIPTION
spark.synapse.logAnalytics.secret	-	The destination Log Analytics workspace secret.
spark.synapse.logAnalytics.keyVault.linkedServiceName	-	The Key Vault linked service name for the Log Analytics workspace ID and key.
spark.synapse.logAnalytics.keyVault.name	-	The Key Vault name for the Log Analytics ID and key.
spark.synapse.logAnalytics.keyVault.key.workspaceId	SparkLogAnalyticsWorkspaceId	The Key Vault secret name for the Log Analytics workspace ID.
spark.synapse.logAnalytics.keyVault.key.secret	SparkLogAnalyticsSecret	The Key Vault secret name for the Log Analytics workspace
spark.synapse.logAnalytics.uriSuffix	ods.opinsights.azure.com	The destination Log Analytics workspace URI suffix. If your workspace isn't in Azure global, you need to update the URI suffix according to the respective cloud.
spark.synapse.logAnalytics.filter.eventName.match	-	Optional. The comma-separated spark event names, you can specify which events to collect. For example: `SparkListenerJobStart,SparkListenerJobEnd`
spark.synapse.logAnalytics.filter.loggerName.match	-	Optional. The comma-separated log4j logger names, you can specify which logs to collect. For example: `org.apache.spark.SparkContext,org.example.Logg`
spark.synapse.logAnalytics.filter.metricName.match	-	Optional. The comma-separated spark metric name suffixes, you can specify which metrics to collect. For example: `jvm.heap.used`

NOTE

- For Azure China, the `spark.synapse.logAnalytics.uriSuffix` parameter should be `ods.opinsights.azure.cn`.
- For Azure Government, the `spark.synapse.logAnalytics.uriSuffix` parameter should be `ods.opinsights.azure.us`.

Step 3: Upload your Apache Spark configuration to an Apache Spark pool

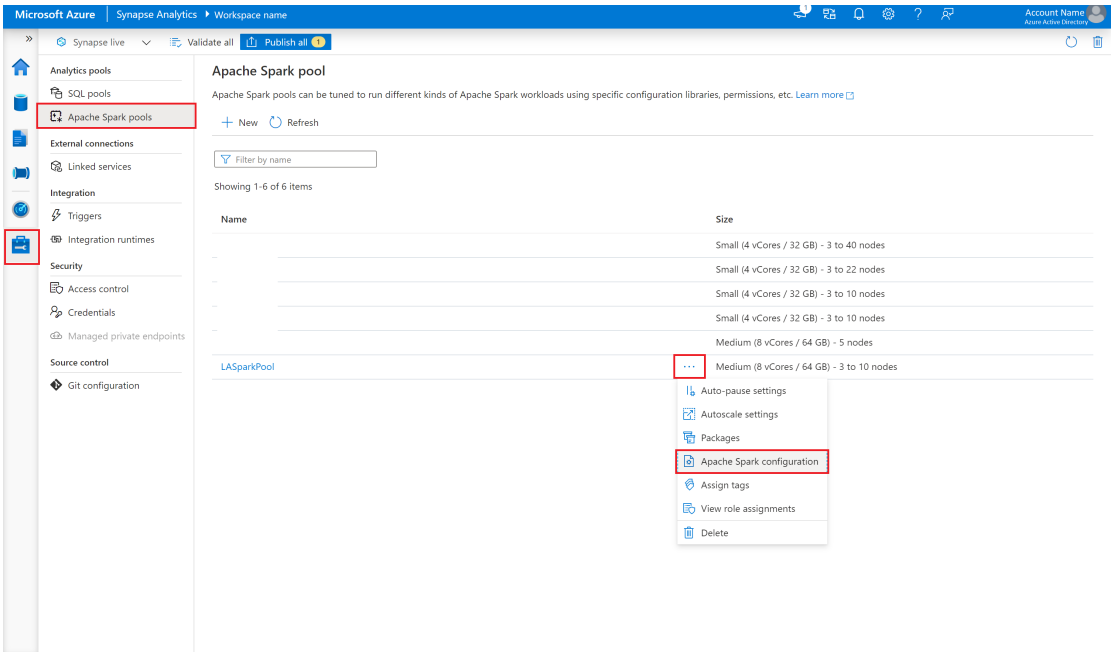
NOTE

This step will be replaced by step 4.

You can upload the configuration file to your Azure Synapse Analytics Apache Spark pool. In Synapse Studio:

1. Select **Manage** > **Apache Spark pools**.
2. Next to your Apache Spark pool, select the **...** button.
3. Select **Apache Spark configuration**.
4. Select **Upload**, and choose the *spark_loganalytics_conf.txt* file.

5. Select **Upload**, and then select **Apply**.

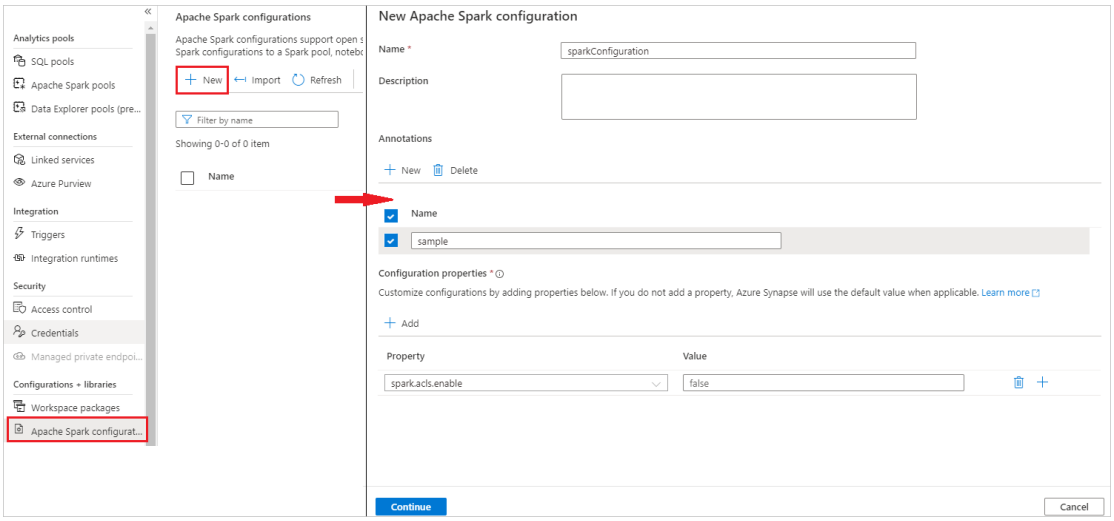


> **NOTE**
>
> All the Apache Spark applications submitted to the Apache Spark pool will use the configuration setting to push the Apache Spark application metrics and logs to your specified workspace.

Step 4: Create an Apache Spark Configuration

You can create an Apache Spark Configuration to your workspace, and when you create Notebook or Apache spark job definition can select the Apache Spark configuration that you want to use with your Apache Spark pool. When you select it, the details of the configuration are displayed.

1. Select **Manage** > **Apache Spark configurations**.
2. Click on **New** button to create a new Apache Spark configuration, or click on **Import** a local .json file to your workspace.
3. **New Apache Spark configuration** page will be opened after you click on **New** button.
4. For **Name**, you can enter your preferred and valid name.
5. For **Description**, you can input some description in it.
6. For **Annotations**, you can add annotations by clicking the **New** button, and also you can delete existing annotations by selecting and clicking **Delete** button.
7. For **Configuration properties**, customize the configuration by clicking **Add** button to add properties. If you do not add a property, Azure Synapse will use the default value when applicable.



Submit an Apache Spark application and view the logs and metrics

Here's how:

1. Submit an Apache Spark application to the Apache Spark pool configured in the previous step. You can use any of the following ways to do so:
   - Run a notebook in Synapse Studio.
   - In Synapse Studio, submit an Apache Spark batch job through an Apache Spark job definition.
   - Run a pipeline that contains Apache Spark activity.
2. Go to the specified Log Analytics workspace, and then view the application metrics and logs when the Apache Spark application starts to run.

Write custom application logs

You can use the Apache Log4j library to write custom logs.

Example for Scala:

```
%%spark
val logger = org.apache.log4j.LogManager.getLogger("com.contoso.LoggerExample")
logger.info("info message")
logger.warn("warn message")
logger.error("error message")
```

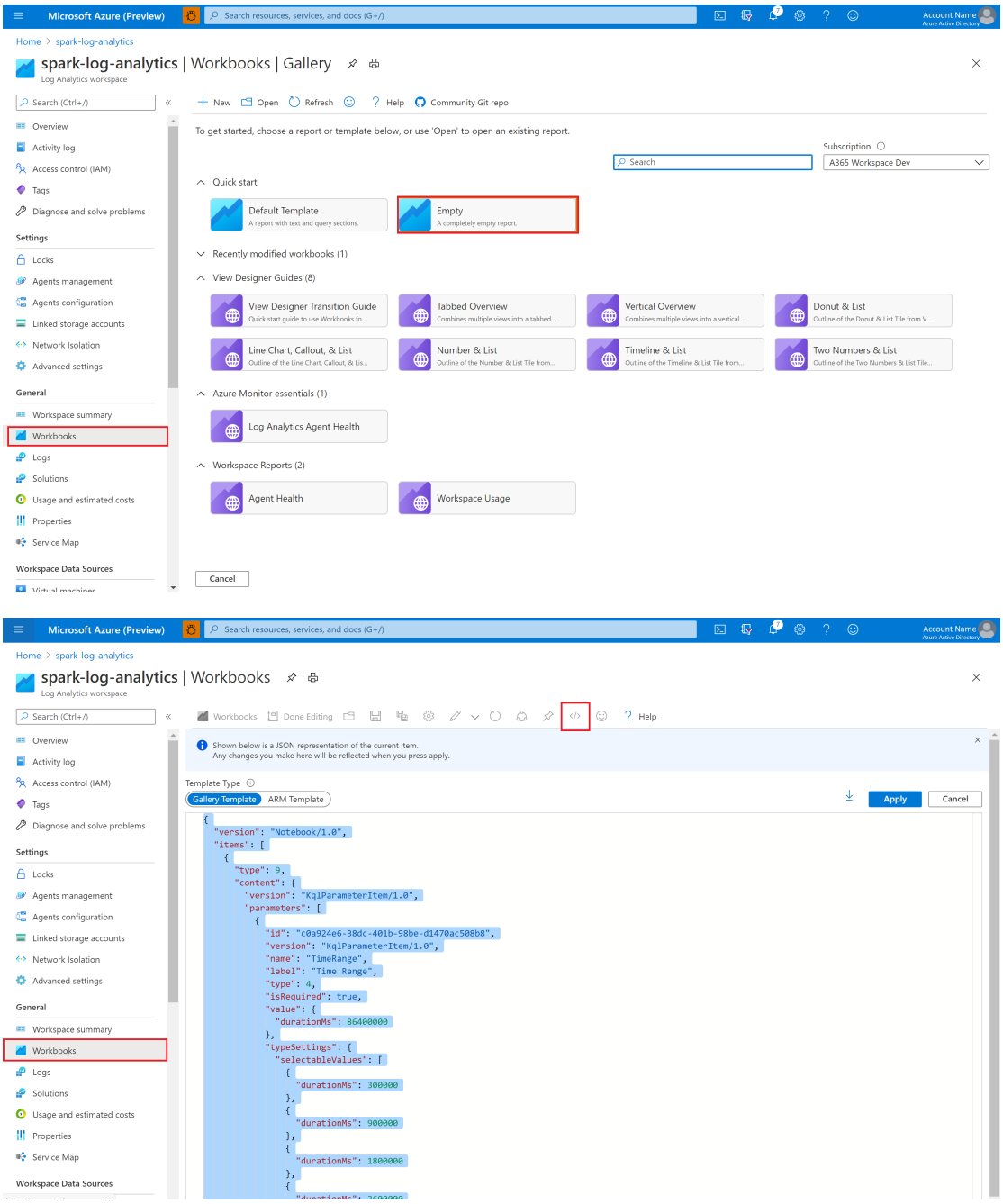
Example for PySpark:

```
%%pyspark
logger = sc._jvm.org.apache.log4j.LogManager.getLogger("com.contoso.PythonLoggerExample")
logger.info("info message")
logger.warn("warn message")
logger.error("error message")
```

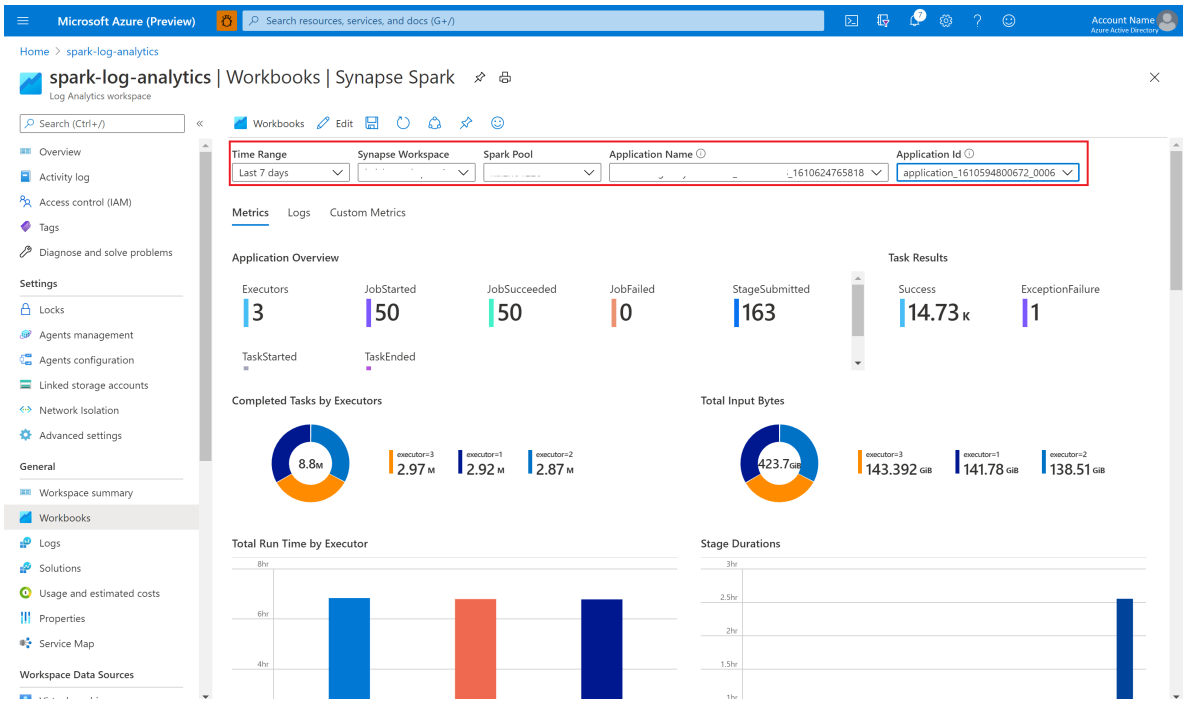
Use the sample workbook to visualize the metrics and logs

1. Download the workbook.
2. Open and copy the workbook file content.
3. In the Azure portal, select **Log Analytics workspace** > **Workbooks**.
4. Open the **Empty** workbook. Use the **Advanced Editor** mode by selecting the **</>** icon.
5. Paste over any JSON code that exists.

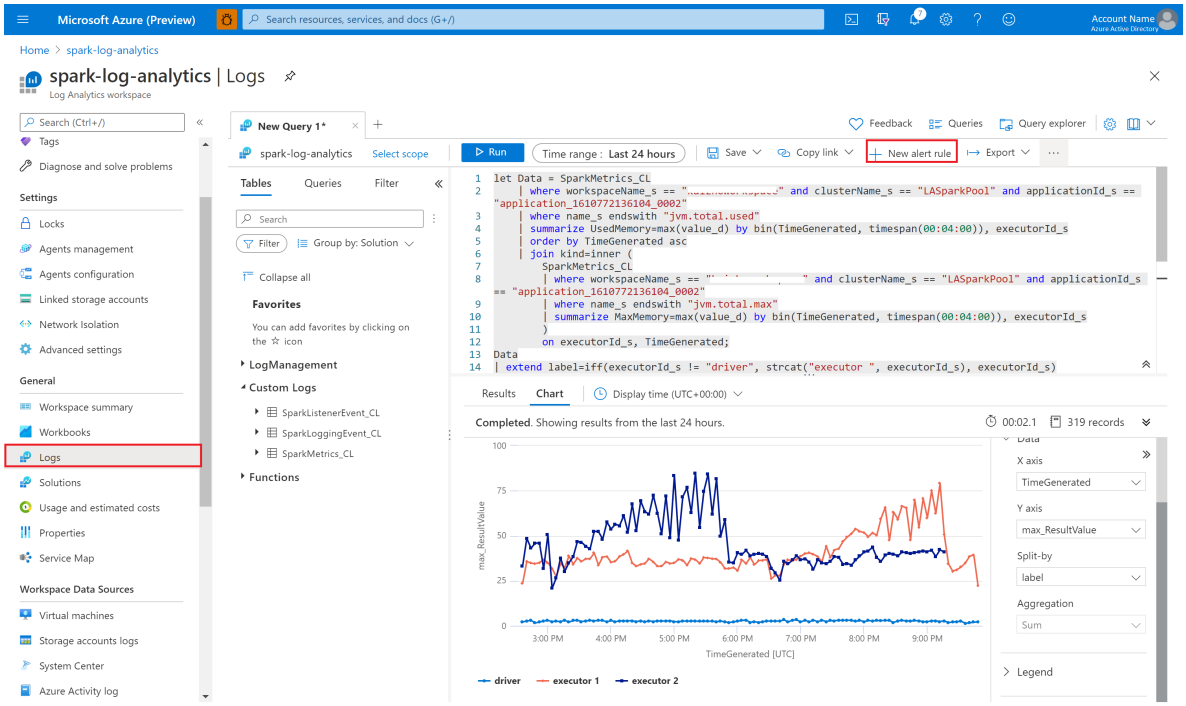
6. Select **Apply**, and then select **Done Editing**.



Then, submit your Apache Spark application to the configured Apache Spark pool. After the application goes to a running state, choose the running application in the workbook dropdown list.



You can customize the workbook. For example, you can use Kusto queries and configure alerts.



Query data with Kusto

The following is an example of querying Apache Spark events:

```
SparkListenerEvent_CL
| where workspaceName_s == "{SynapseWorkspace}" and clusterName_s == "{SparkPool}" and livyId_s == "
{LivyId}"
| order by TimeGenerated desc
| limit 100
```

Here's an example of querying the Apache Spark application driver and executors logs:

```
SparkLoggingEvent_CL
| where workspaceName_s == "{SynapseWorkspace}" and clusterName_s == "{SparkPool}" and livyId_s == "
{LivyId}"
| order by TimeGenerated desc
| limit 100
```

And here's an example of querying Apache Spark metrics:

```
SparkMetrics_CL
| where workspaceName_s == "{SynapseWorkspace}" and clusterName_s == "{SparkPool}" and livyId_s == "
{LivyId}"
| where name_s endswith "jvm.total.used"
| summarize max(value_d) by bin(TimeGenerated, 30s), executorId_s
| order by TimeGenerated asc
```

Create and manage alerts

Users can query to evaluate metrics and logs at a set frequency, and fire an alert based on the results. For more information, see Create, view, and manage log alerts by using Azure Monitor.

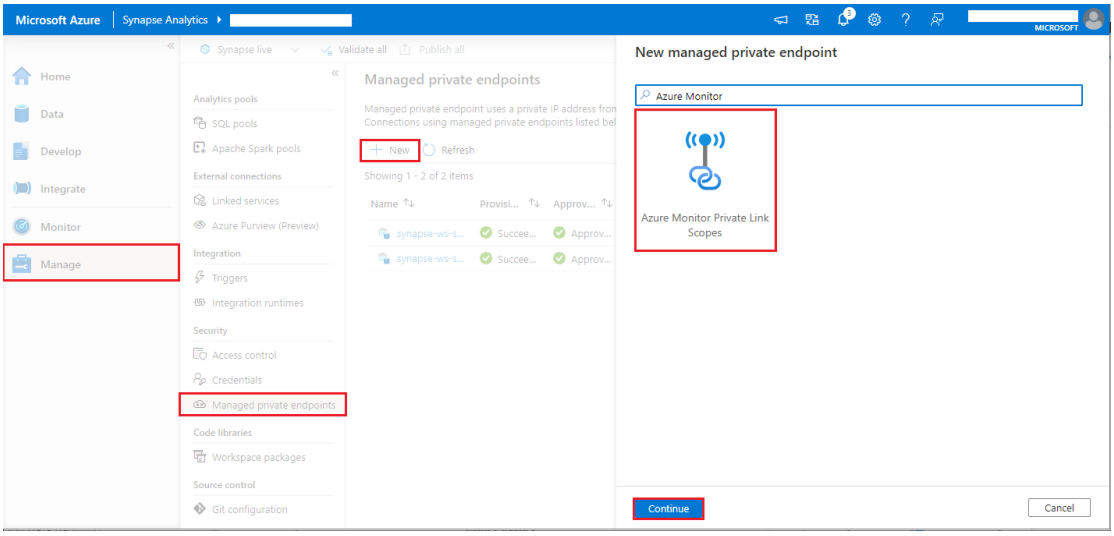
Synapse workspace with data exfiltration protection enabled

After the Synapse workspace is created with data exfiltration protection enabled.

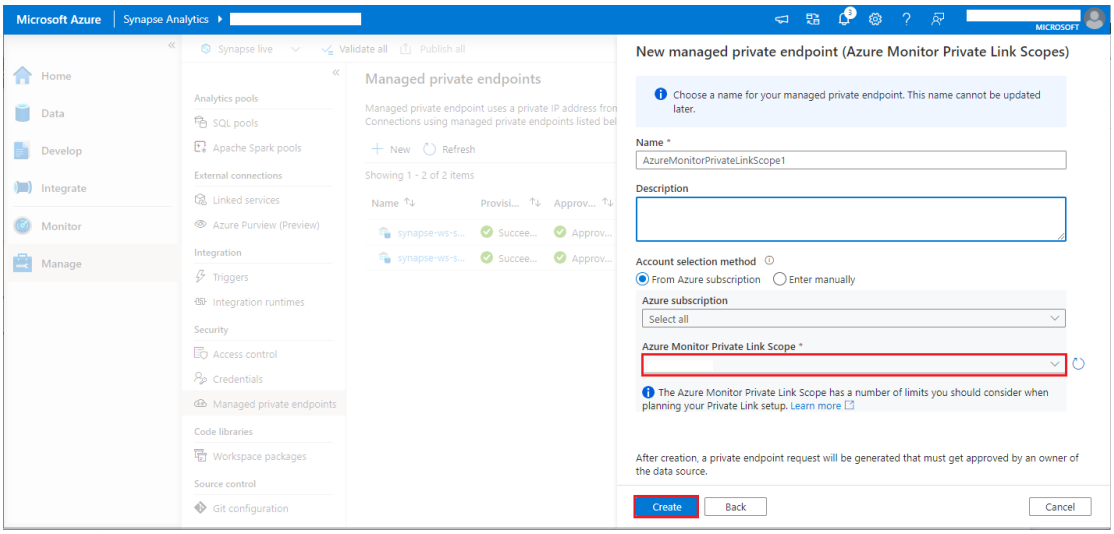
When you want to enable this feature, you need to create managed private endpoint connection requests to Azure Monitor private link scopes (A M P L S) in the workspace's approved Azure AD tenants.

You can follow below steps to create a managed private endpoint connection to Azure Monitor private link scopes (A M P L S):

1. If there is no existing A M P L S, you can follow Azure Monitor Private Link connection setup to create one.
2. Navigate to your A M P L S in Azure portal, on the **Azure Monitor Resources** page, click **Add** to add connection to your Azure Log Analytics workspace.
3. Navigate to **Synapse Studio > Manage > Managed private endpoints**, click **New** button, select **Azure Monitor Private Link Scopes**, and **continue**.



4. Choose your Azure Monitor Private Link Scope you created, and click **Create** button.



5. Wait a few minutes for private endpoint provisioning.
6. Navigate to your A M P L S in Azure portal again, on the **Private Endpoint connections** page, select the connection provisioned and **Approve**.

> **NOTE**
>
> - The A M P L S object has a number of limits you should consider when planning your Private Link setup. See A M P L S limits for a deeper review of these limits.
> - Check if you have right permission to create managed private endpoint.

Next steps

- Use serverless Apache Spark pool in Synapse Studio.
- Run a Spark application in notebook.
- Create Apache Spark job definition in Azure Studio.
- Collect Apache Spark applications logs and metrics with Azure Storage account.
- Collect Apache Spark applications logs and metrics with Azure Event Hubs.

Collect your Apache Spark applications logs and metrics using Azure Storage account

8/19/2022 • 3 minutes to read • Edit Online

The Synapse Apache Spark diagnostic emitter extension is a library that enables the Apache Spark application to emit the logs, event logs, and metrics to one or more destinations, including Azure Log Analytics, Azure Storage, and Azure Event Hubs.

In this tutorial, you learn how to use the Synapse Apache Spark diagnostic emitter extension to emit Apache Spark applications' logs, event logs, and metrics to your Azure storage account.

Collect logs and metrics to storage account

Step 1: Create a storage account

To collect diagnostic logs and metrics to storage account, you can use existing Azure Storage accounts. Or if you don't have one, you can create an Azure blob storage account or create a storage account to use with Azure Data Lake Storage Gen2.

Step 2: Create an Apache Spark configuration file

Create a `diagnostic-emitter-azure-storage-conf.txt` and copy following contents to the file. Or download a sample template file for Apache Spark pool configuration.

```
spark.synapse.diagnostic.emitters MyDestination1
spark.synapse.diagnostic.emitter.MyDestination1.type AzureStorage
spark.synapse.diagnostic.emitter.MyDestination1.categories Log,EventLog,Metrics
spark.synapse.diagnostic.emitter.MyDestination1.uri https://<my-blob-
storage>.blob.core.windows.net/<container-name>/<folder-name>
spark.synapse.diagnostic.emitter.MyDestination1.auth AccessKey
spark.synapse.diagnostic.emitter.MyDestination1.secret <storage-access-key>
```

Fill in the following parameters in the configuration file: `<my-blob-storage>`, `<container-name>`, `<folder-name>`, `<storage-access-key>`. For more description of the parameters, you can refer to Azure Storage configurations

Step 3: Upload the Apache Spark configuration file to Spark pool

1. Navigate to your Apache Spark pool in Synapse Studio **(Manage -> Apache Spark pools)**
2. Click the **"..."** button on the right of your Apache Spark pool and select **Apache Spark configuration**
3. Click **Upload** and choose the ".txt" configuration file. And click **Apply**.

Step 4: View the logs files in Azure storage account

After you submit a job to the configured Apache Spark pool, you should be able to see the logs and metrics files in destination storage account. The logs will be placed in corresponding paths according to different applications by `<workspaceName>.<sparkPoolName>.<livySessionId>`. All the logs files will be in JSON lines format (also called newline-delimited JSON, ndjson), which is convenient for data processing.

Available configurations

CONFIGURATION	DESCRIPTION

CONFIGURATION	DESCRIPTION
`spark.synapse.diagnostic.emitters`	Required. The comma-separated destination names of diagnostic emitters. For example, `MyDest1,MyDest2`
`spark.synapse.diagnostic.emitter.<destination>.type`	Required. Built-in destination type. To enable Azure storage destination, `AzureStorage` needs to be included in this field.
`spark.synapse.diagnostic.emitter.<destination>.categories`	Optional. The comma-separated selected log categories. Available values include `DriverLog`, `ExecutorLog`, `EventLog`, `Metrics`. If not set, the default value is **all** categories.
`spark.synapse.diagnostic.emitter.<destination>.auth`	Required. `AccessKey` for using storage account access key authorization. `SAS` for shared access signatures authorization.
`spark.synapse.diagnostic.emitter.<destination>.uri`	Required. The destination blob container folder uri. Should match pattern `https://<my-blob-storage>.blob.core.windows.net/<container-name>/<folder-name>` .
`spark.synapse.diagnostic.emitter.<destination>.secret`	Optional. The secret (AccessKey or SAS) content.
`spark.synapse.diagnostic.emitter.<destination>.secret.keyVault`	Required if `.secret` is not specified. The Azure Key vault name where the secret (AccessKey or SAS) is stored.
`spark.synapse.diagnostic.emitter.<destination>.secret.keyVault.secretName`	Required if `.secret.keyVault` is specified. The Azure Key vault secret name where the secret (AccessKey or SAS) is stored.
`spark.synapse.diagnostic.emitter.<destination>.secret.keyVault.linkedService`	Optional. The Azure Key vault linked service name. When enabled in Synapse pipeline, this is necessary to obtain the secret from AKV. (Please make sure MSI has read permission on the AKV).
`spark.synapse.diagnostic.emitter.<destination>.filter.eventName.match`	Optional. The comma-separated spark event names, you can specify which events to collect. For example: `SparkListenerApplicationStart,SparkListenerApplicationEnd`
`spark.synapse.diagnostic.emitter.<destination>.filter.loggerName.match`	Optional. The comma-separated log4j logger names, you can specify which logs to collect. For example: `org.apache.spark.SparkContext,org.example.Logger`
`spark.synapse.diagnostic.emitter.<destination>.filter.metricName.match`	Optional. The comma-separated spark metric name suffixes, you can specify which metrics to collect. For example: `jvm.heap.used`

Log data sample

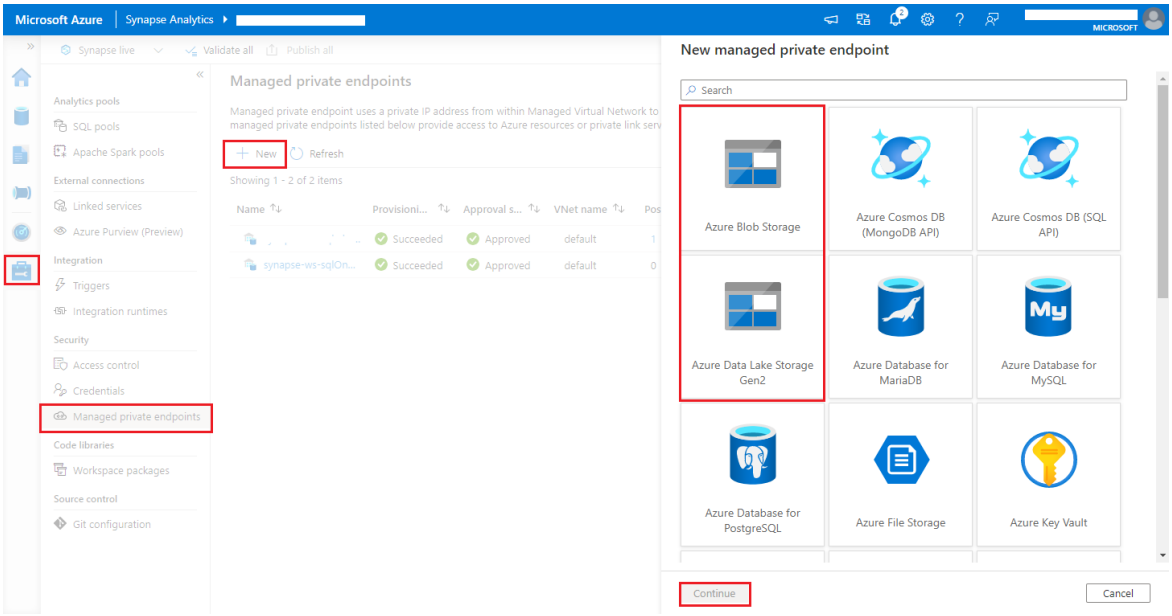
Here is a sample log record in JSON format:

```
{
    "timestamp": "2021-01-02T12:34:56.789Z",
    "category": "Log|EventLog|Metrics",
    "workspaceName": "<my-workspace-name>",
    "sparkPool": "<spark-pool-name>",
    "livyId": "<livy-session-id>",
    "applicationId": "<application-id>",
    "applicationName": "<application-name>",
    "executorId": "<driver-or-executor-id>",
    "properties": {
        // The message properties of logs, events and metrics.
        "timestamp": "2021-01-02T12:34:56.789Z",
        "message": "Registering signal handler for TERM",
        "logger_name": "org.apache.spark.util.SignalUtils",
        "level": "INFO",
        "thread_name": "main"
        // ...
    }
}
```

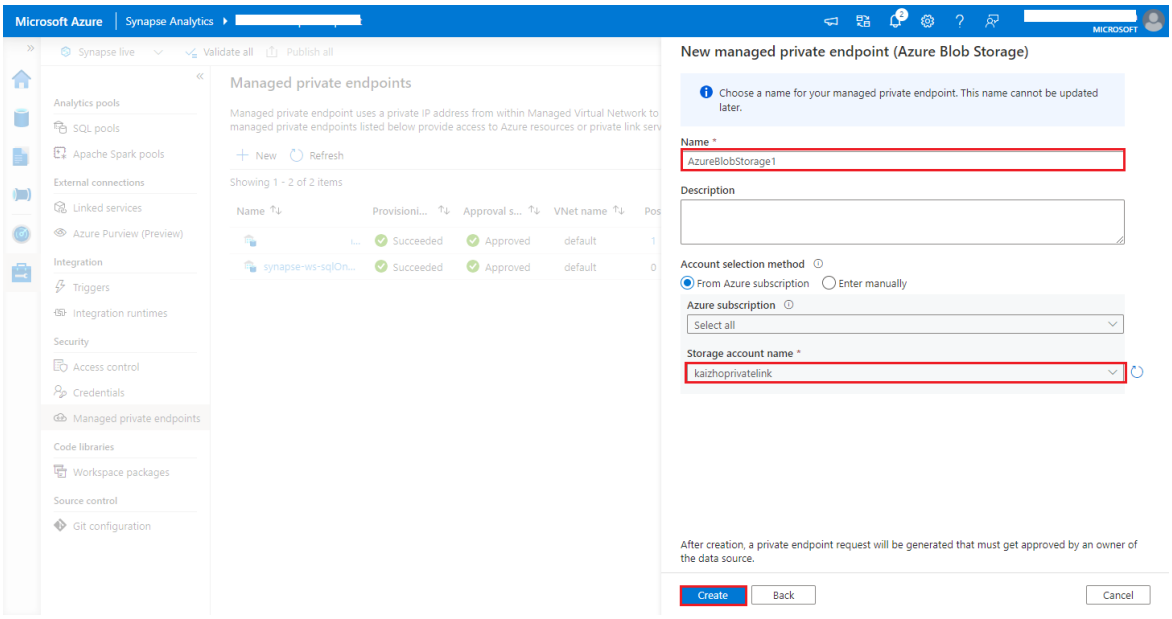
Synapse workspace with data exfiltration protection enabled

Azure Synapse Analytics workspaces support enabling data exfiltration protection for workspaces. With exfiltration protection, the logs and metrics cannot be sent out to the destination endpoints directly. You can create corresponding managed private endpoints for different destination endpoints or create IP firewall rules in this scenario.

1. Navigate to **Synapse Studio** > **Manage** > **Managed private endpoints**, click **New** button, select **Azure Blob Storage** or **Azure Data Lake Storage Gen2** and **continue**.



2. Choose your Azure Storage account in the **Storage account name**, and click **Create** button.



3. Wait a few minutes for private endpoint provisioning.
4. Navigate to your storage account in Azure portal, on the **Networking** > **Private Endpoint connections** page, select the connection provisioned and **Approve**.

Collect your Apache Spark applications logs and metrics using Azure Event Hubs

8/19/2022 • 2 minutes to read • Edit Online

The Synapse Apache Spark diagnostic emitter extension is a library that enables the Apache Spark application to emit the logs, event logs, and metrics to one or more destinations, including Azure Log Analytics, Azure Storage, and Azure Event Hubs.

In this tutorial, you learn how to use the Synapse Apache Spark diagnostic emitter extension to emit Apache Spark applications' logs, event logs, and metrics to your Azure Event Hubs.

Collect logs and metrics to Azure Event Hubs

Step 1: Create an Azure Event Hub instance

To collect diagnostic logs and metrics to Azure Event Hubs, you can use existing Azure Event Hubs instance. Or if you don't have one, you can create an event hub.

Step 2: Create an Apache Spark configuration file

Create a `diagnostic-emitter-azure-event-hub-conf.txt` and copy following contents to the file. Or download a sample template file for Apache Spark pool configuration.

```
spark.synapse.diagnostic.emitters MyDestination1
spark.synapse.diagnostic.emitter.MyDestination1.type AzureEventHub
spark.synapse.diagnostic.emitter.MyDestination1.categories Log,EventLog,Metrics
spark.synapse.diagnostic.emitter.MyDestination1.secret <connection-string>
```

Fill in the following parameters in the configuration file: `<connection-string>`. For more description of the parameters, you can refer to Azure EventHub configurations

Step 3: Upload the Apache Spark configuration file to Apache Spark pool

1. Navigate to your Apache Spark pool in Synapse Studio **(Manage -> Apache Spark pools)**
2. Click the **"..."** button on the right of your Apache Spark pool and select **Apache Spark configuration**
3. Click **Upload** and choose the ".txt" configuration file, and click **Apply**.

Available configurations

CONFIGURATION	DESCRIPTION
`spark.synapse.diagnostic.emitters`	Required. The comma-separated destination names of diagnostic emitters.
`spark.synapse.diagnostic.emitter.<destination>.type`	Required. Built-in destination type. To enable Azure Event Hubs destination, the value should be `AzureEventHub`.
`spark.synapse.diagnostic.emitter.<destination>.categories`	Optional. The comma-separated selected log categories. Available values include `DriverLog`, `ExecutorLog`, `EventLog`, `Metrics`. If not set, the default value is **all** categories.

CONFIGURATION	DESCRIPTION
`spark.synapse.diagnostic.emitter.<destination>.secret`	Optional. The Azure Eventhub instance connection string. This field should match this pattern `Endpoint=sb://<FQDN>/;SharedAccessKeyName=<KeyName>;SharedAccessKey=<KeyValue>;EntityPath=<PathName>`
`spark.synapse.diagnostic.emitter.<destination>.secret.keyVault`	Required if `.secret` is not specified. The Azure Key vault name where the secret (connection string) is stored.
`spark.synapse.diagnostic.emitter.<destination>.secret.keyVault.secretName`	Required if `.secret.keyVault` is specified. The Azure Key vault secret name where the secret (connection string) is stored.
`spark.synapse.diagnostic.emitter.<destination>.secret.keyVault.linkedService`	Optional. The Azure Key vault linked service name. When enabled in Synapse pipeline, this is necessary to obtain the secret from AKV. (Please make sure MSI has read permission on the AKV).
`spark.synapse.diagnostic.emitter.<destination>.filter.eventName.match`	Optional. The comma-separated spark event names, you can specify which events to collect. For example: `SparkListenerApplicationStart,SparkListenerApplicationEnd`
`spark.synapse.diagnostic.emitter.<destination>.filter.loggerName.match`	Optional. The comma-separated log4j logger names, you can specify which logs to collect. For example: `org.apache.spark.SparkContext,org.example.Logger`
`spark.synapse.diagnostic.emitter.<destination>.filter.metricName.match`	Optional. The comma-separated spark metric name suffixes, you can specify which metrics to collect. For example: `jvm.heap.used`

NOTE

The Azure Eventhub instance connection string should always contains the `EntityPath`, which is the name of the Azure Event Hubs instance.

Log data sample

Here is a sample log record in JSON format:

```
{
    "timestamp": "2021-01-02T12:34:56.789Z",
    "category": "Log|EventLog|Metrics",
    "workspaceName": "<my-workspace-name>",
    "sparkPool": "<spark-pool-name>",
    "livyId": "<livy-session-id>",
    "applicationId": "<application-id>",
    "applicationName": "<application-name>",
    "executorId": "<driver-or-executor-id>",
    "properties": {
        // The message properties of logs, events and metrics.
        "timestamp": "2021-01-02T12:34:56.789Z",
        "message": "Registering signal handler for TERM",
        "logger_name": "org.apache.spark.util.SignalUtils",
        "level": "INFO",
        "thread_name": "main"
        // ...
    }
}
```

Synapse workspace with data exfiltration protection enabled

Azure Synapse Analytics workspaces support enabling data exfiltration protection for workspaces. With exfiltration protection, the logs and metrics cannot be sent out to the destination endpoints directly. You can create corresponding managed private endpoints for different destination endpoints or create IP firewall rules in this scenario.

Manage Apache Spark configuration

8/19/2022 • 4 minutes to read • Edit Online

In this tutorial, you will learn how to create an Apache Spark configuration for your synapse studio. The created Apache Spark configuration can be managed in a standardized manner and when you create Notebook or Apache spark job definition can select the Apache Spark configuration that you want to use with your Apache Spark pool. When you select it, the details of the configuration are displayed.

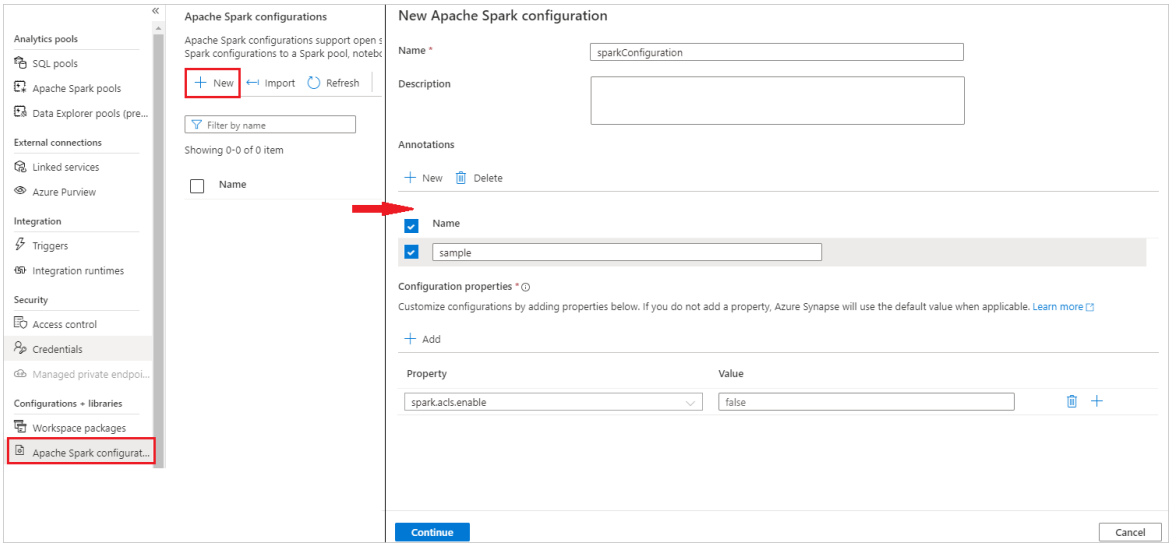
Create an Apache Spark Configuration

You can create custom configurations from different entry points, such as from the Apache Spark configurations page, from the Apache Spark configuration page of an existing spark pool.

Create custom configurations in Apache Spark configurations

Follow the steps below to create an Apache Spark Configuration in Synapse Studio.

1. Select **Manage** > **Apache Spark configurations**.
2. Click on **New** button to create a new Apache Spark configuration, or click on **Import** a local .json file to your workspace.
3. **New Apache Spark configuration** page will be opened after you click on **New** button.
4. For **Name**, you can enter your preferred and valid name.
5. For **Description**, you can input some description in it.
6. For **Annotations**, you can add annotations by clicking the **New** button, and also you can delete existing annotations by selecting and clicking **Delete** button.
7. For **Configuration properties**, customize the configuration by clicking **Add** button to add properties. If you do not add a property, Azure Synapse will use the default value when applicable.



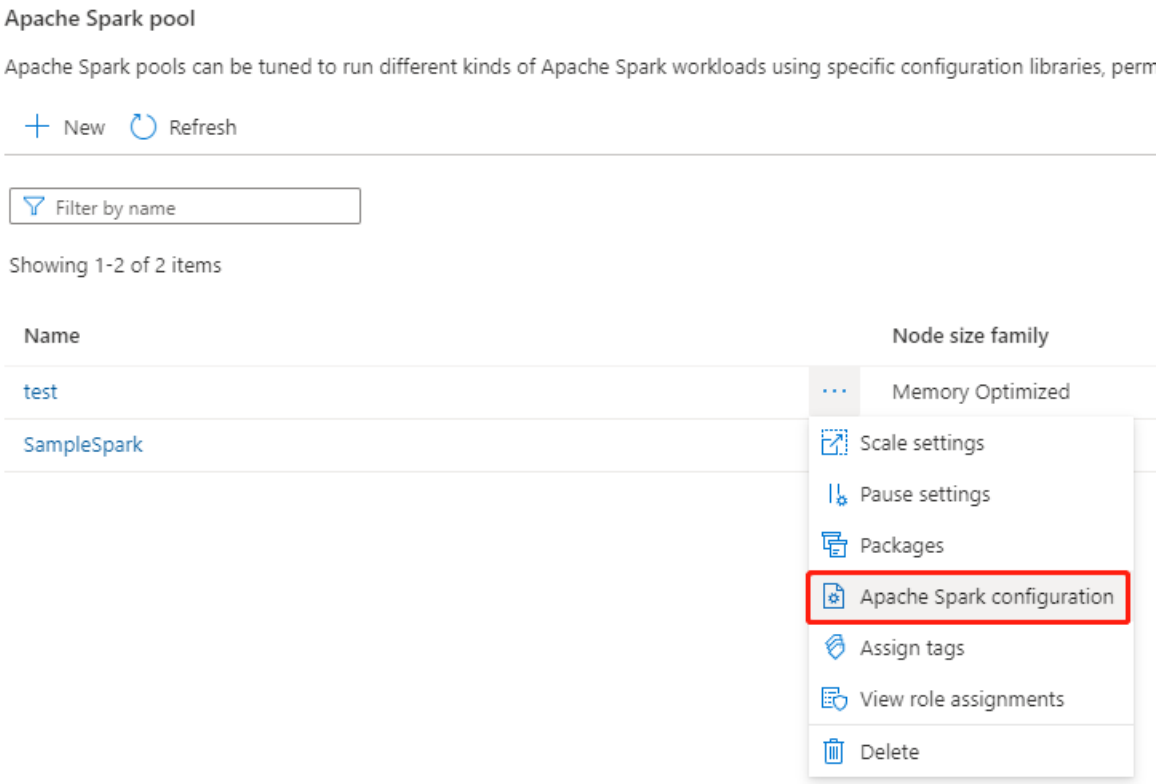
8. Click on **Continue** button.
9. Click on **Create** button when the validation succeeded.
10. Publish all

> **NOTE**
>
> **Upload Apache Spark configuration** feature has been removed, but Synapse Studio will keep your previously uploaded configuration.

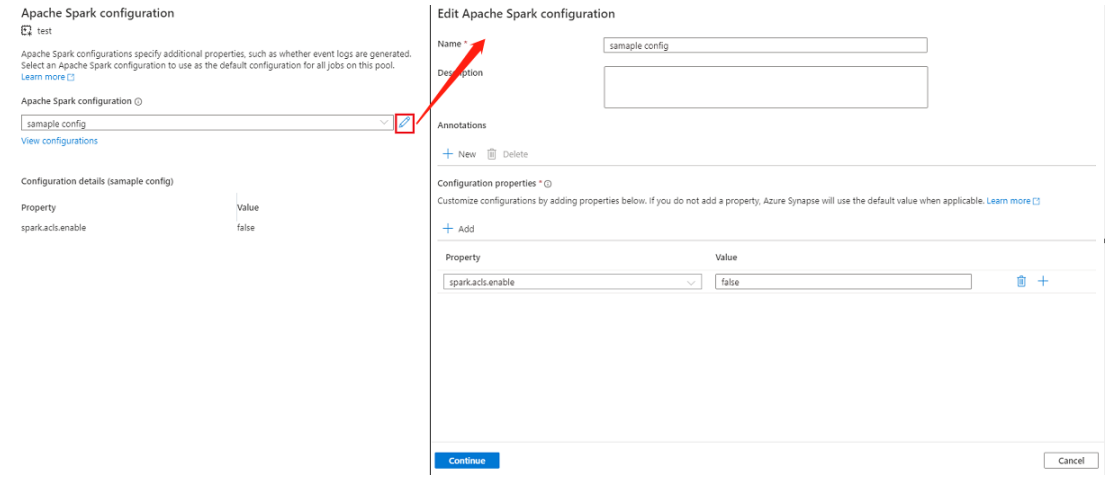
Create an Apache Spark Configuration in already existing Apache Spark pool

Follow the steps below to create an Apache Spark configuration in an existing Apache Spark pool.

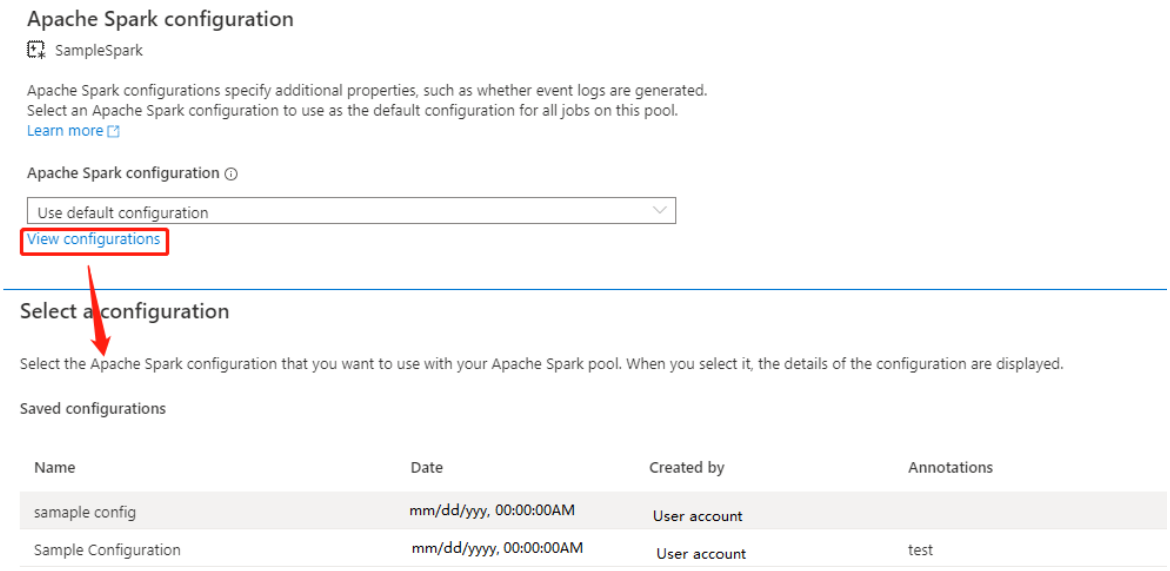
1. Select an existing Apache Spark pool, and click on action "..." button.
2. Select the **Apache Spark configuration** in the content list.



3. For Apache Spark configuration, you can select an already created configuration from the drop-down list, or click on **+New** to create a new configuration.
   - If you click **+New**, the Apache Spark Configuration page will open, and you can create a new configuration by following the steps in Create custom configurations in Apache Spark configurations.
   - If you select an existing configuration, the configuration details will be displayed at the bottom of the page, you can also click the **Edit** button to edit the existing configuration.



4. Click **View Configurations** to open the **Select a Configuration** page. All configurations will be displayed on this page. You can select a configuration that you want to use on this Apache Spark pool.

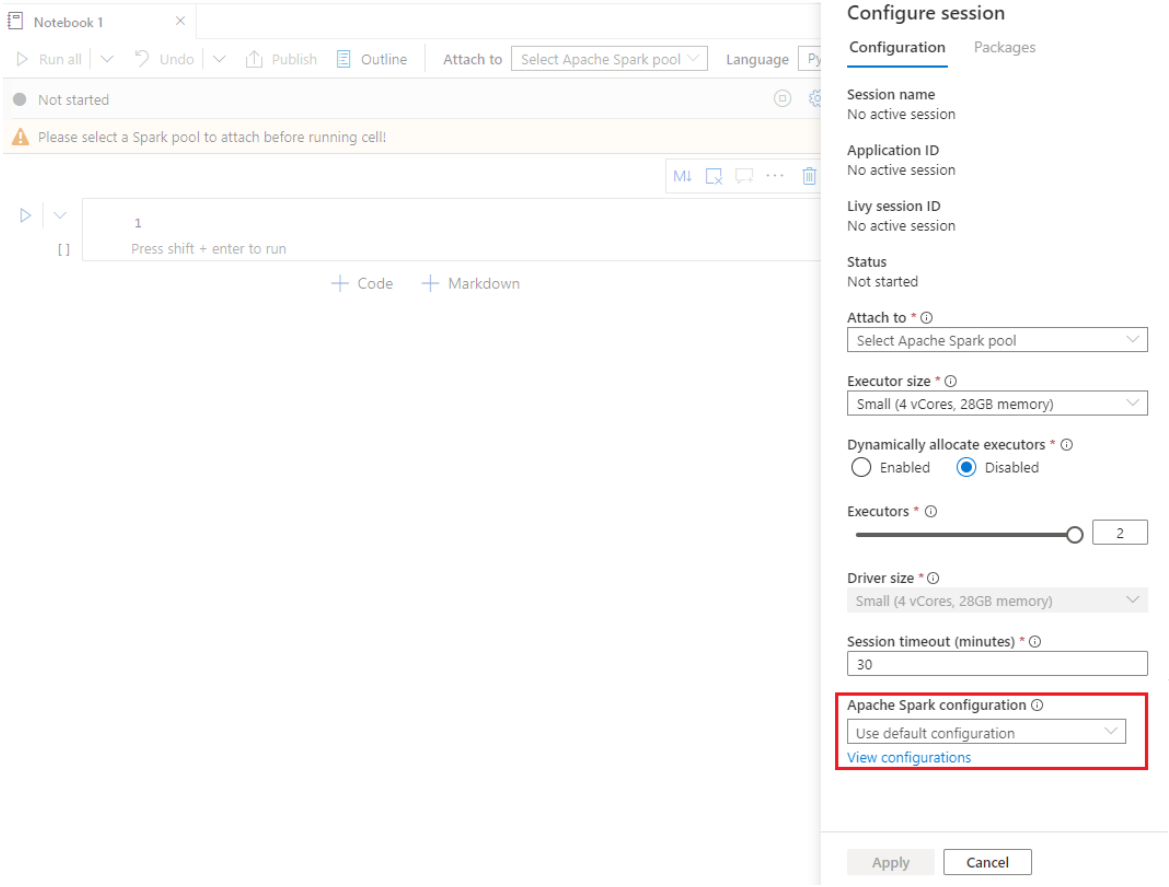


5. Click on **Apply** button to save your action.

Create an Apache Spark Configuration in the Notebook's configure session

If you need to use a custom Apache Spark Configuration when creating a Notebook, you can create and configure it in the **configure session** by following the steps below.

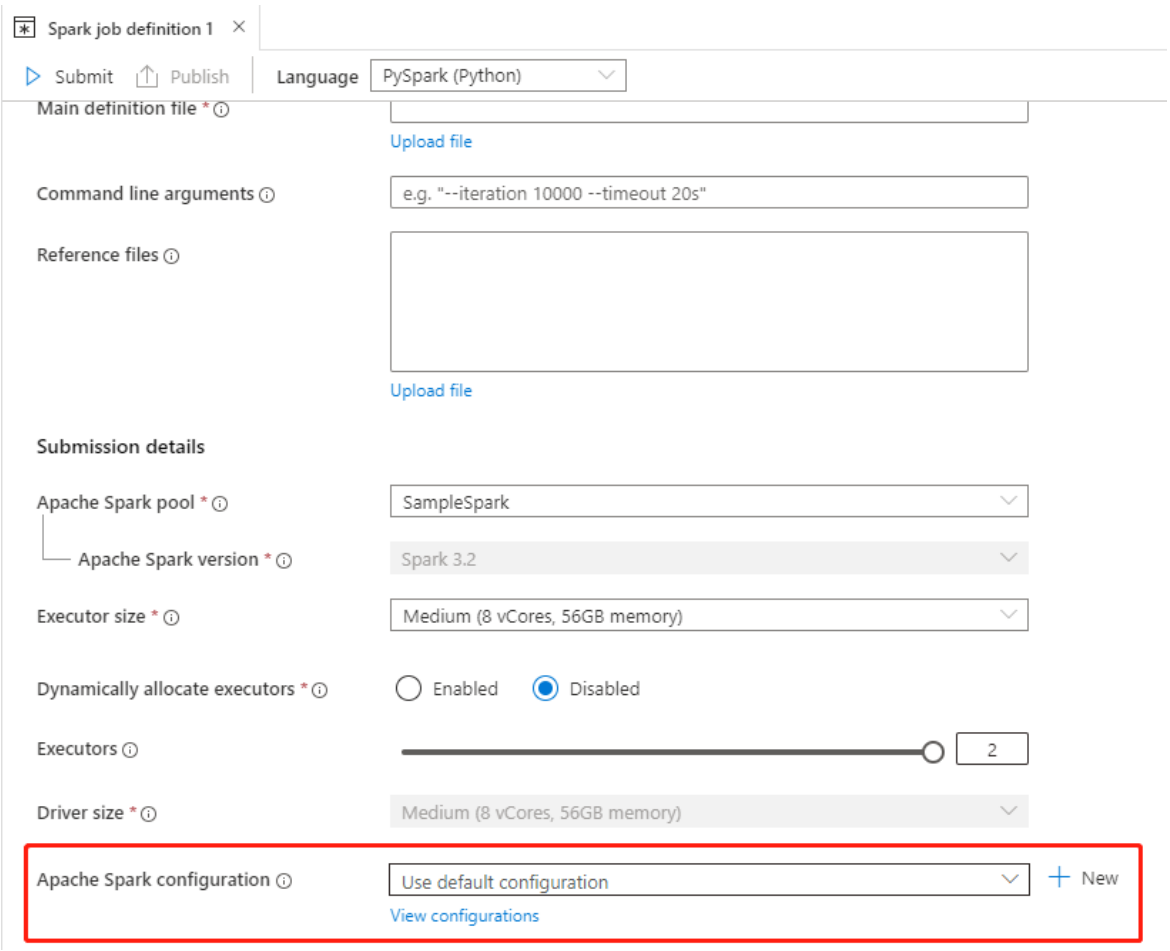
1. Create a new/Open an existing Notebook.
2. Open the **Properties** of this notebook.
3. Click on **Configure session** to open the Configure session page.
4. Scroll down the configure session page, for Apache Spark configuration, expand the drop-down menu, you can click on New button to create a new configuration. Or select an existing configuration, if you select an existing configuration, click the **Edit** icon to go to the Edit Apache Spark configuration page to edit the configuration.
5. Click **View Configurations** to open the **Select a Configuration** page. All configurations will be displayed on this page. You can select a configuration that you want to use.



Create an Apache Spark Configuration in Apache Spark job definitions

When you are creating a spark job definition, you need to use Apache Spark configuration, which can be created by following the steps below:

1. Create a new/Open an existing Apache Spark job definitions.
2. For **Apache Spark configuration**, you can click on New button to create a new configuration. Or select an existing configuration in the drop-down menu, if you select an existing configuration, click the **Edit** icon to go to the Edit Apache Spark configuration page to edit the configuration.
3. Click **View Configurations** to open the **Select a Configuration** page. All configurations will be displayed on this page. You can select a configuration that you want to use.



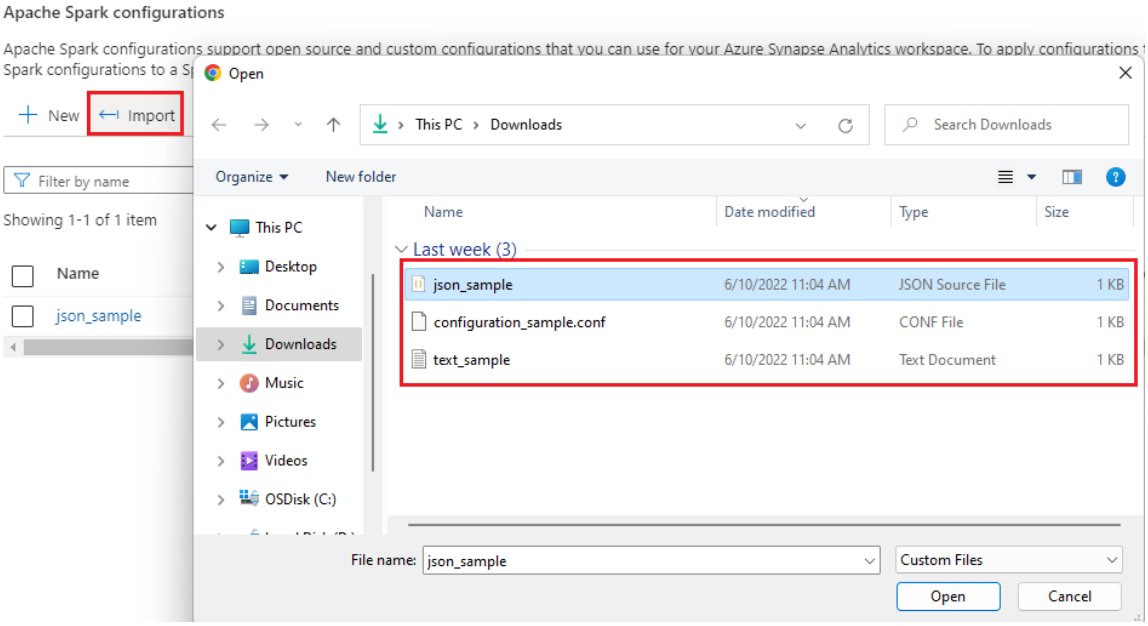
NOTE

If the Apache Spark configuration in the Notebook and Apache Spark job definition does not do anything special, the default configuration will be used when running the job.

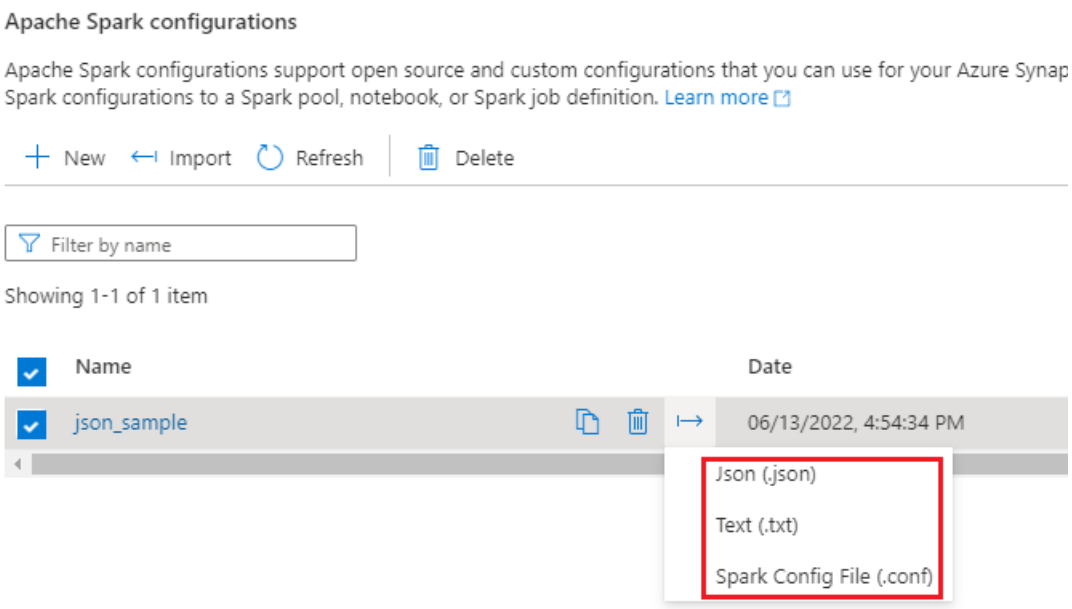
Import and Export an Apache Spark configuration

You can import .txt/.conf/.json config in three formats and then convert it to artifact and publish it. And can also export to one of these three formats.

- Import .txt/.conf/.json configuration from local.



- Export .txt/.conf/.json configuration to local.



For .txt config file and .conf config file, you can refer to the following examples:

```
spark.synapse.key1 sample
spark.synapse.key2 true
# spark.synapse.key3 sample2
```

For .json config file, you can refer to the following examples:

```
{
 "configs": {
   "spark.synapse.key1": "hello world",
  "spark.synapse.key2": "true"
 },
 "annotations": [
   "Sample"
 ]
}
```

Next steps

- Use serverless Apache Spark pool in Synapse Studio.
- Run a Spark application in notebook.
- Create Apache Spark job definition in Azure Studio.
- Collect Apache Spark applications logs and metrics with Azure Storage account.
- Collect Apache Spark applications logs and metrics with Azure Event Hubs.

Spark Advisor

8/19/2022 • 2 minutes to read • Edit Online

Spark Advisor is a system to automatically analyze commands/queries, and show the appropriate advice when customer executes code or query. After applying the advice, you would have chance to improve your execution performance, decrease cost and fix the execution failures.

Advice provided

May return inconsistent results when using 'randomSplit'

Inconsistent or inaccurate results may be returned when working with the results of the 'randomSplit' method. Use Apache Spark (RDD) caching before using the 'randomSplit' method.

Method randomSplit() is equivalent to performing sample() on your data frame multiple times, with each sample refetching, partitioning, and sorting your data frame within partitions. The data distribution across partitions and sorting order is important for both randomSplit() and sample(). If either changes upon data refetch, there may be duplicates, or missing values across splits and the same sample using the same seed may produce different results.

These inconsistencies may not happen on every run, but to eliminate them completely, cache your data frame, repartition on a column(s), or apply aggregate functions such as groupBy.

Table/view name is already in use

A view already exists with the same name as the created table, or a table already exists with the same name as the created view. When this name is used in queries or applications, only the view will be returned no matter, which one created first. To avoid conflicts, rename either the table or the view.

Hints related advise

Unable to recognize a hint

The selected query contains a hint that isn't recognized. Verify that the hint is spelled correctly.

```
spark.sql("SELECT /*+ unknownHint */ * FROM t1")
```

Unable to find a specified relation name(s)

Unable to find the relation(s) specified in the hint. Verify that the relation(s) are spelled correctly and accessible within the scope of the hint.

```
spark.sql("SELECT /*+ BROADCAST(unknownTable) */ * FROM t1 INNER JOIN t2 ON t1.str = t2.str")
```

A hint in the query prevents another hint from being applied

The selected query contains a hint that prevents another hint from being applied.

```
spark.sql("SELECT /*+ BROADCAST(t1), MERGE(t1, t2) */ * FROM t1 INNER JOIN t2 ON t1.str = t2.str")
```

Enable 'spark.advise.divisionExprConvertRule.enable' to reduce

rounding error propagation

This query contains the expression with Double type. We recommend that you enable the configuration 'spark.advise.divisionExprConvertRule.enable', which can help reduce the division expressions and to reduce the rounding error propagation.

```
"t.a/t.b/t.c" convert into "t.a/(t.b * t.c)"
```

Enable 'spark.advise.nonEqJoinConvertRule.enable' to improve query performance

This query contains time consuming join due to "Or" condition within query. We recommend that you enable the configuration 'spark.advise.nonEqJoinConvertRule.enable', which can help to convert the join triggered by "Or" condition to SMJ or BHJ to accelerate this query.

Next steps

For more information on monitoring pipeline runs, see the Monitor pipeline runs using Synapse Studio article.

Interact with Azure Cosmos DB using Apache Spark 3 in Azure Synapse Link

8/19/2022 • 7 minutes to read • Edit Online

In this article, you'll learn how to interact with Azure Cosmos DB using Synapse Apache Spark 3. With its full support for Scala, Python, SparkSQL, and C#, Synapse Apache Spark 3 is central to analytics, data engineering, data science, and data exploration scenarios in Azure Synapse Link for Azure Cosmos DB.

The following capabilities are supported while interacting with Azure Cosmos DB:

- Synapse Apache Spark 3 allows you to analyze data in your Azure Cosmos DB containers that are enabled with Azure Synapse Link in near real-time without impacting the performance of your transactional workloads. The following two options are available to query the Azure Cosmos DB analytical store from Spark:
  - Load to Spark DataFrame
  - Create Spark table
- Synapse Apache Spark also allows you to ingest data into Azure Cosmos DB. It is important to note that data is always ingested into Azure Cosmos DB containers through the transactional store. When Synapse Link is enabled, any new inserts, updates, and deletes are then automatically synced to the analytical store.
- Synapse Apache Spark also supports Spark structured streaming with Azure Cosmos DB as a source as well as a sink.

The following sections walk you through the syntax of above capabilities. You can also checkout the learn module on how to Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics. Gestures in Azure Synapse Analytics workspace are designed to provide an easy out-of-the-box experience to get started. Gestures are visible when you right-click on an Azure Cosmos DB container in the **Data** tab of the Synapse workspace. With gestures, you can quickly generate code and tailor it to your needs. Gestures are also perfect for discovering data with a single click.

> **IMPORTANT**
>
> You should be aware of some constraints in the analytical schema that could lead to the unexpected behavior in data loading operations. As an example, only first 1000 properties from transactional schema are available in the analytical schema, properties with spaces are not available, etc. If you are experiencing some unexpected results, check the analytical store schema constraints for more details.

Query Azure Cosmos DB analytical store

Before you learn about the two possible options to query Azure Cosmos DB analytical store, loading to Spark DataFrame and creating Spark table, it is worth exploring the differences in experience so you can choose the option that works for your needs.

The difference in experience is around whether underlying data changes in the Azure Cosmos DB container should be automatically reflected in the analysis performed in Spark. When either a Spark DataFrame is registered or a Spark table is created against a container's analytical store, metadata around the current snapshot of data in the analytical store is fetched to Spark for efficient pushdown of subsequent analysis. It is important to note that since Spark follows a lazy evaluation policy, unless an action is invoked on the Spark DataFrame or a SparkSQL query is executed against the Spark table, actual data is not fetched from the underlying container's analytical store.

In the case of **loading to Spark DataFrame**, the fetched metadata is cached through the lifetime of the Spark session and hence subsequent actions invoked on the DataFrame are evaluated against the snapshot of the analytical store at the time of DataFrame creation.

On the other hand, in the case of **creating a Spark table**, the metadata of the analytical store state is not cached in Spark and is reloaded on every SparkSQL query execution against the Spark table.

Thus, you can choose between loading to Spark DataFrame and creating a Spark table based on whether you want your Spark analysis to be evaluated against a fixed snapshot of the analytical store or against the latest snapshot of the analytical store respectively.

> **NOTE**
>
> To query the Azure Cosmos DB API of Mongo DB accounts, learn more about the full fidelity schema representation in the analytical store and the extended property names to be used.

> **NOTE**
>
> Please note that all `options` in the commands below are case sensitive.

Load to Spark DataFrame

In this example, you'll create a Spark DataFrame that points to the Azure Cosmos DB analytical store. You can then perform additional analysis by invoking Spark actions against the DataFrame. This operation doesn't impact the transactional store.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

df = spark.read.format("cosmos.olap")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .load()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

val df_olap = spark.read.format("cosmos.olap").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    load()
```

Create Spark table

In this example, you'll create a Spark table that points the Azure Cosmos DB analytical store. You can then perform additional analysis by invoking SparkSQL queries against the table. This operation neither impacts the transactional store nor does it incur any data movement. If you decide to delete this Spark table, the underlying Azure Cosmos DB container and the corresponding analytical store will not be affected.

This scenario is convenient to reuse Spark tables through third-party tools and provide accessibility to the underlying data for the run-time.

The syntax to create a Spark table is as follows:

```
%%sql
-- To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
spark.cosmos.preferredRegions '<Region1>,<Region2>' in the config options

create table call_center using cosmos.olap options (
    spark.synapse.linkedService '<enter linked service name>',
    spark.cosmos.container '<enter container name>'
)
```

NOTE

If you have scenarios where the schema of the underlying Azure Cosmos DB container changes over time; and if you want the updated schema to automatically reflect in the queries against the Spark table, you can achieve this by setting the `spark.cosmos.autoSchemaMerge` option to `true` in the Spark table options.

Write Spark DataFrame to Azure Cosmos DB container

In this example, you'll write a Spark DataFrame into an Azure Cosmos DB container. This operation will impact the performance of transactional workloads and consume request units provisioned on the Azure Cosmos DB container or the shared database.

The syntax in **Python** would be the following:

```
# Write a Spark DataFrame into an Azure Cosmos DB container
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

YOURDATAFRAME.write.format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .mode('append')\
    .save()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

import org.apache.spark.sql.SaveMode

df.write.format("cosmos.oltp").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    mode(SaveMode.Append).
    save()
```

Load streaming DataFrame from container

In this gesture, you'll use Spark Streaming capability to load data from a container into a dataframe. The data will be stored in the primary data lake account (and file system) you connected to the workspace.

> **NOTE**
>
> If you are looking to reference external libraries in Synapse Apache Spark, learn more here. For instance, if you are looking to ingest a Spark DataFrame to a container of Cosmos DB API for Mongo DB, you can leverage the Mongo DB connector for Spark here.

Load streaming DataFrame from Azure Cosmos DB container

In this example, you'll use Spark's structured streaming capability to load data from an Azure Cosmos DB container into a Spark streaming DataFrame using the change feed functionality in Azure Cosmos DB. The checkpoint data used by Spark will be stored in the primary data lake account (and file system) that you connected to the workspace.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

dfStream = spark.readStream\
    .format("cosmos.oltp.changeFeed")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .option("spark.cosmos.changeFeed.startFrom", "Beginning")\
    .option("spark.cosmos.changeFeed.mode", "Incremental")\
    .load()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

val dfStream = spark.readStream.
    format("cosmos.oltp.changeFeed").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    option("spark.cosmos.changeFeed.startFrom", "Beginning").
    option("spark.cosmos.changeFeed.mode", "Incremental").
    load()
```

Write streaming DataFrame to Azure Cosmos DB container

In this example, you'll write a streaming DataFrame into an Azure Cosmos DB container. This operation will impact the performance of transactional workloads and consume Request Units provisioned on the Azure Cosmos DB container or shared database. If the folder */localWriteCheckpointFolder* isn't created (in the example below), it will be automatically created.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

streamQuery = dfStream\
    .writeStream\
    .format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .option("checkpointLocation", "/tmp/myRunId/")\
    .outputMode("append")\
    .start()

streamQuery.awaitTermination()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

val query = dfStream.
            writeStream.
            format("cosmos.oltp").
            outputMode("append").
            option("spark.synapse.linkedService", "<enter linked service name>").
            option("spark.cosmos.container", "<enter container name>").
            option("checkpointLocation", "/tmp/myRunId/").
            start()

query.awaitTermination()
```

Next steps

- Samples to get started with Azure Synapse Link on GitHub
- Learn what is supported in Azure Synapse Link for Azure Cosmos DB
- Connect to Synapse Link for Azure Cosmos DB
- Checkout the learn module on how to Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics.

Interact with Azure Cosmos DB using Apache Spark 2 in Azure Synapse Link

8/19/2022 • 9 minutes to read • Edit Online

> **NOTE**
> For Synapse Link for Cosmos DB using Spark 3, refer to this article Azure Synapse Link for Azure Cosmos DB on Spark 3

In this article, you'll learn how to interact with Azure Cosmos DB using Synapse Apache Spark 2. With its full support for Scala, Python, SparkSQL, and C#, Synapse Apache Spark is central to analytics, data engineering, data science, and data exploration scenarios in Azure Synapse Link for Azure Cosmos DB.

The following capabilities are supported while interacting with Azure Cosmos DB:

- Synapse Apache Spark allows you to analyze data in your Azure Cosmos DB containers that are enabled with Azure Synapse Link in near real-time without impacting the performance of your transactional workloads. The following two options are available to query the Azure Cosmos DB analytical store from Spark:
  - Load to Spark DataFrame
  - Create Spark table
- Synapse Apache Spark also allows you to ingest data into Azure Cosmos DB. It is important to note that data is always ingested into Azure Cosmos DB containers through the transactional store. When Synapse Link is enabled, any new inserts, updates, and deletes are then automatically synced to the analytical store.
- Synapse Apache Spark also supports Spark structured streaming with Azure Cosmos DB as a source as well as a sink.

The following sections walk you through the syntax of above capabilities. You can also checkout the learn module on how to Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics. Gestures in Azure Synapse Analytics workspace are designed to provide an easy out-of-the-box experience to get started. Gestures are visible when you right-click on an Azure Cosmos DB container in the **Data** tab of the Synapse workspace. With gestures, you can quickly generate code and tailor it to your needs. Gestures are also perfect for discovering data with a single click.

> **IMPORTANT**
> You should be aware of some constraints in the analytical schema that could lead to the unexpected behavior in data loading operations. As an example, only first 1000 properties from transactional schema are available in the analytical schema, properties with spaces are not available, etc. If you are experiencing some unexpected results, check the analytical store schema constraints for more details.

Query Azure Cosmos DB analytical store

Before you learn about the two possible options to query Azure Cosmos DB analytical store, loading to Spark DataFrame and creating Spark table, it is worth exploring the differences in experience so you can choose the option that works for your needs.

The difference in experience is around whether underlying data changes in the Azure Cosmos DB container should be automatically reflected in the analysis performed in Spark. When either a Spark DataFrame is registered or a Spark table is created against a container's analytical store, metadata around the current

snapshot of data in the analytical store is fetched to Spark for efficient pushdown of subsequent analysis. It is important to note that since Spark follows a lazy evaluation policy, unless an action is invoked on the Spark DataFrame or a SparkSQL query is executed against the Spark table, actual data is not fetched from the underlying container's analytical store.

In the case of **loading to Spark DataFrame**, the fetched metadata is cached through the lifetime of the Spark session and hence subsequent actions invoked on the DataFrame are evaluated against the snapshot of the analytical store at the time of DataFrame creation.

On the other hand, in the case of **creating a Spark table**, the metadata of the analytical store state is not cached in Spark and is reloaded on every SparkSQL query execution against the Spark table.

Thus, you can choose between loading to Spark DataFrame and creating a Spark table based on whether you want your Spark analysis to be evaluated against a fixed snapshot of the analytical store or against the latest snapshot of the analytical store respectively.

If your analytical queries have frequently used filters, you have the option to partition based on these fields for better query performance. You can periodically execute partitioning job from an Azure Synapse Spark notebook, to trigger partitioning on analytical store. This partitioned store points to the ADLS Gen2 primary storage account that is linked to your Azure Synapse workspace. To learn more, see the introduction to custom partitioning and how to configure custom partitioning articles.

> **NOTE**
> To query the Azure Cosmos DB API of Mongo DB accounts, learn more about the full fidelity schema representation in the analytical store and the extended property names to be used.

> **NOTE**
> Please note that all `options` in the commands below are case sensitive. For example, you must use `Gateway` while `gateway` will return an error.

Load to Spark DataFrame

In this example, you'll create a Spark DataFrame that points to the Azure Cosmos DB analytical store. You can then perform additional analysis by invoking Spark actions against the DataFrame. This operation doesn't impact the transactional store.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

df = spark.read.format("cosmos.olap")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .load()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

val df_olap = spark.read.format("cosmos.olap").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    load()
```

Create Spark table

In this example, you'll create a Spark table that points the Azure Cosmos DB analytical store. You can then perform additional analysis by invoking SparkSQL queries against the table. This operation neither impacts the transactional store nor does it incur any data movement. If you decide to delete this Spark table, the underlying Azure Cosmos DB container and the corresponding analytical store will not be affected.

This scenario is convenient to reuse Spark tables through third-party tools and provide accessibility to the underlying data for the run-time.

The syntax to create a Spark table is as follows:

```
%%sql
-- To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
spark.cosmos.preferredRegions '<Region1>,<Region2>' in the config options

create table call_center using cosmos.olap options (
    spark.synapse.linkedService '<enter linked service name>',
    spark.cosmos.container '<enter container name>'
)
```

> **NOTE**
>
> If you have scenarios where the schema of the underlying Azure Cosmos DB container changes over time; and if you want the updated schema to automatically reflect in the queries against the Spark table, you can achieve this by setting the `spark.cosmos.autoSchemaMerge` option to `true` in the Spark table options.

Write Spark DataFrame to Azure Cosmos DB container

In this example, you'll write a Spark DataFrame into an Azure Cosmos DB container. This operation will impact the performance of transactional workloads and consume request units provisioned on the Azure Cosmos DB container or the shared database.

The syntax in **Python** would be the following:

```
# Write a Spark DataFrame into an Azure Cosmos DB container
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

YOURDATAFRAME.write.format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .option("spark.cosmos.write.upsertEnabled", "true")\
    .mode('append')\
    .save()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

import org.apache.spark.sql.SaveMode

df.write.format("cosmos.oltp").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    option("spark.cosmos.write.upsertEnabled", "true").
    mode(SaveMode.Overwrite).
    save()
```

Load streaming DataFrame from container

In this gesture, you'll use Spark Streaming capability to load data from a container into a dataframe. The data will be stored in the primary data lake account (and file system) you connected to the workspace.

> **NOTE**
> If you are looking to reference external libraries in Synapse Apache Spark, learn more here. For instance, if you are looking to ingest a Spark DataFrame to a container of Cosmos DB API for Mongo DB, you can leverage the Mongo DB connector for Spark here.

Load streaming DataFrame from Azure Cosmos DB container

In this example, you'll use Spark's structured streaming capability to load data from an Azure Cosmos DB container into a Spark streaming DataFrame using the change feed functionality in Azure Cosmos DB. The checkpoint data used by Spark will be stored in the primary data lake account (and file system) that you connected to the workspace.

If the folder */localReadCheckpointFolder* isn't created (in the example below), it will be automatically created. This operation will impact the performance of transactional workloads and consume Request Units provisioned on the Azure Cosmos DB container or shared database.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

dfStream = spark.readStream\
    .format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .option("spark.cosmos.changeFeed.readEnabled", "true")\
    .option("spark.cosmos.changeFeed.startFromTheBeginning", "true")\
    .option("spark.cosmos.changeFeed.checkpointLocation", "/localReadCheckpointFolder")\
    .option("spark.cosmos.changeFeed.queryName", "streamQuery")\
    .load()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

val dfStream = spark.readStream.
    format("cosmos.oltp").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    option("spark.cosmos.changeFeed.readEnabled", "true").
    option("spark.cosmos.changeFeed.startFromTheBeginning", "true").
    option("spark.cosmos.changeFeed.checkpointLocation", "/localReadCheckpointFolder").
    option("spark.cosmos.changeFeed.queryName", "streamQuery").
    load()
```

Write streaming DataFrame to Azure Cosmos DB container

In this example, you'll write a streaming DataFrame into an Azure Cosmos DB container. This operation will impact the performance of transactional workloads and consume Request Units provisioned on the Azure Cosmos DB container or shared database. If the folder */localWriteCheckpointFolder* isn't created (in the example below), it will be automatically created.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

# If you are using managed private endpoints for Azure Cosmos DB analytical store and using batch
writes/reads and/or streaming writes/reads to transactional store you should set connectionMode to Gateway.

def writeBatchToCosmos(batchDF, batchId):
  batchDF.persist()
  print("--> BatchId: {}, Document count: {} : {}".format(batchId, batchDF.count(),
datetime.datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S.%f")))
  batchDF.write.format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .option("spark.cosmos.write.upsertEnabled", "true")\
    .mode('append')\
    .save()
  print("<-- BatchId: {}, Document count: {} : {}".format(batchId, batchDF.count(),
datetime.datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S.%f")))
  batchDF.unpersist()

streamQuery = dfStream\
        .writeStream\
        .foreachBatch(writeBatchToCosmos) \
        .option("checkpointLocation", "/localWriteCheckpointFolder")\
        .start()

streamQuery.awaitTermination()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

// If you are using managed private endpoints for Azure Cosmos DB analytical store and using batch
writes/reads and/or streaming writes/reads to transactional store you should set connectionMode to Gateway.

val query = dfStream.
            writeStream.
            foreachBatch { (batchDF: DataFrame, batchId: Long) =>
              batchDF.persist()
              batchDF.write.format("cosmos.oltp").
                option("spark.synapse.linkedService", "<enter linked service name>").
                option("spark.cosmos.container", "<enter container name>").
                option("spark.cosmos.write.upsertEnabled", "true").
                mode(SaveMode.Overwrite).
                save()
              println(s"BatchId: $batchId, Document count: ${batchDF.count()}")
              batchDF.unpersist()
              ()
            }.
            option("checkpointLocation", "/localWriteCheckpointFolder").
            start()

query.awaitTermination()
```

Next steps

- Samples to get started with Azure Synapse Link on GitHub
- Learn what is supported in Azure Synapse Link for Azure Cosmos DB
- Connect to Synapse Link for Azure Cosmos DB
- Checkout the learn module on how to Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics.

Azure Data Explorer (Kusto) connector for Apache Spark

8/19/2022 • 2 minutes to read • Edit Online

The Azure Data Explorer (Kusto) connector for Apache Spark is designed to efficiently transfer data between Kusto clusters and Spark. This connector is available in Python, Java, and .NET. It is built in to the Azure Synapse Apache Spark 2.4 runtime.

Authentication

When using Azure Synapse Notebooks or Apache Spark job definitions, the authentication between systems is made seamless with the linked service. The Token Service connects with Azure Active Directory to obtain security tokens for use when accessing the Kusto cluster.

For Azure Synapse Pipelines, the authentication will use the service principal name. Currently, managed identities are not supported with the Azure Data Explorer connector.

Prerequisites

- Connect to Azure Data Explorer: You will need to set up a Linked Service to connect to an existing Kusto cluster.

Limitations

- The Azure Data Explorer (Kusto) connector is currently only supported on the Azure Synapse Apache Spark 2.4 runtime.
- The Azure Data Explorer linked service can only be configured with the Service Principal Name.
- Within Azure Synapse Notebooks or Apache Spark Job Definitions, the Azure Data Explorer connector will use AAD pass-through to connect to the Kusto Cluster.

Use the Azure Data Explorer (Kusto) connector

The following section provides a simple example of how to write data to a Kusto table and read data from a Kusto table. See the Azure Data Explorer (Kusto) connector project for detailed documentation.

Read data

```
kustoDf  = spark.read \
            .format("com.microsoft.kusto.spark.synapse.datasource") \
            .option("spark.synapse.linkedService", "<link service name>") \
            .option("kustoDatabase", "<Database name>") \
            .option("kustoQuery", "<KQL Query>") \
            .load()

display(kustoDf)
```

You can also batch read with forced distribution mode and other advanced options. For additional information, you can refer to Kusto source options reference.

```
crp = sc._jvm.com.microsoft.azure.kusto.data.ClientRequestProperties()
crp.setOption("norequesttimeout",True)
crp.toString()

kustoDf  = spark.read \
            .format("com.microsoft.kusto.spark.synapse.datasource") \
            .option("spark.synapse.linkedService", "<link service name>") \
            .option("kustoDatabase", "<Database name>") \
            .option("kustoQuery", "<KQL Query>") \
            .option("clientRequestPropertiesJson", crp.toString()) \
            .option("readMode", 'ForceDistributedMode') \
            .load()

display(kustoDf)
```

Write data

```
df.write \
    .format("com.microsoft.kusto.spark.synapse.datasource") \
    .option("spark.synapse.linkedService", "<link service name>") \
    .option("kustoDatabase", "<Database name>") \
    .option("kustoTable", "<Table name>") \
    .mode("Append") \
    .save()
```

In addition, you can also batch write data by providing additional ingestion properties. For more info on the supported ingestion properties, you can visit the Kusto ingestion properties reference material.

```
extentsCreationTime = sc._jvm.org.joda.time.DateTime.now().plusDays(1)
csvMap = "[{\"Name\":\"ColA\",\"Ordinal\":0},{\"Name\":\"ColB\",\"Ordinal\":1}]"
# Alternatively use an existing csv mapping configured on the table and pass it as the last parameter of
SparkIngestionProperties or use none


sp = sc._jvm.com.microsoft.kusto.spark.datasink.SparkIngestionProperties(
       False, ["dropByTags"], ["ingestByTags"], ["tags"], ["ingestIfNotExistsTags"], extentsCreationTime,
csvMap, None)

df.write \
   .format("com.microsoft.kusto.spark.synapse.datasource") \
   .option("spark.synapse.linkedService", "<link service name>") \
   .option("kustoDatabase", "<Database name>") \
   .option("kustoTable", "<Table name>") \
   .option("sparkIngestionPropertiesJson", sp.toString()) \
   .option("tableCreateOptions","CreateIfNotExist") \
   .mode("Append") \
   .save()
```

Next steps

- Connect to Azure Data Explorer
- Azure Data Explorer (Kusto) Apache Spark connector

Common Data Model (CDM) Connector for Azure Synapse Spark

8/19/2022 • 22 minutes to read • Edit Online

The Synapse Spark Common Data Model (CDM) format reader/writer enables a Spark program to read and write CDM entities in a CDM folder via Spark dataframes.

For information on defining CDM documents using CDM 1.0 see. What is CDM and how to use it.

High level functionality

The following capabilities are supported:

- Reading data from an entity in a CDM folder into a Spark dataframe
- Writing from a Spark dataframe to an entity in a CDM folder based on a CDM entity definition
- Writing from a Spark dataframe to an entity in a CDM folder based on the dataframe schema

Capabilities

- Supports reading and writing to CDM folders in ADLS gen2 with HNS enabled.
- Supports reading from CDM folders described by either manifest or model.json files.
- Supports writing to CDM folders described by a manifest file.
- Supports data in CSV format with/without column headers and with user selectable delimiter character.
- Supports data in Apache Parquet format, including nested Parquet.
- Supports submanifests on read, optional use of entity-scoped submanifests on write.
- Supports writing data using user modifiable partition patterns.
- Supports use of managed identity Synapse and credentials.
- Supports resolving CDM aliases locations used in imports using CDM adapter definitions described in a config.json.

Limitations

The following scenarios aren't supported:

- Programmatic access to entity metadata after reading an entity.
- Programmatic access to set or override metadata when writing an entity.
- Schema drift - where data in a dataframe being written includes extra attributes not included in the entity definition.
- Schema evolution - where entity partitions reference different versions of the entity definition
- Write support for model.json isn't supported.
- Executing `com.microsoft.cdm.BuildInfo.version` will verify the version

Spark 2.4 and Spark 3.1 are supported.

Reading data

When reading data, the connector uses metadata in the CDM folder to create the dataframe based on the resolved entity definition for the specified entity, as referenced in the manifest. Entity attribute names are used

as dataframe column names. Attribute datatypes are mapped to the column datatype. When the dataframe is loaded, it's populated from the entity partitions identified in the manifest.

The connector looks in the specified manifest and any first-level submanifests for the specified entity. If the required entity is in a second-level or lower submanifest, or if there are multiple entities of the same name in different submanifests, then the user should specify the submanifest that contains the required entity rather than the root manifest. Entity partitions can be in a mix of formats (CSV, Parquet, etc.). All the entity data files identified in the manifest are combined into one dataset regardless of format and loaded to the dataframe.

When reading CSV data, the connector uses the Spark FAILFAST option by default. It will return an error if the number of columns isn't equal to the number of attributes in the entity. Alternatively, as of 0.19, permissive mode is now supported. This mode is only supported for CSV files. With the permissive mode, when a CSV row has fewer number of columns than the entity schema, null values will be assigned for the missing columns. When a CSV row has more columns than the entity schema, the columns greater than the entity schema column count will be truncated to the schema column count. Usage is as follows:

```
.option("entity", "permissive") or .option("mode", "failfast")
```

For example, here's an example python sample.

Writing data

When writing to a CDM folder, if the entity doesn't already exist in the CDM folder, a new entity and definition is created and added to the CDM folder and referenced in the manifest. Two writing modes are supported:

Explicit write: the physical entity definition is based on a logical CDM entity definition that the user specifies.

- The specified logical entity definition is read and resolved to create the physical entity definition used in the CDM folder. If import statements in any directly or indirectly referenced CDM definition file include aliases, then a config.json file that maps these aliases to CDM adapters and storage locations must be provided. For more on the use of aliases, see *Aliases and adapter configuration* below.
- If the dataframe schema doesn't match the referenced entity definition, an error is returned. Ensure that the column datatypes in the dataframe match the attribute datatypes in the entity, including for decimal data, precision and scale set via traits in CDM.
- If the dataframe is inconsistent with the entity definition an error is returned.
- If the dataframe is consistent:
  - If the entity already exists in the manifest, the provided entity definition is resolved and validated against the definition in the CDM folder. If the definitions don't match, an error is returned, otherwise data is written and the partition information in the manifest is updated
    - If the entity doesn't exist in the CDM folder, a resolved copy of the entity definition is written to the manifest in the CDM folder and data is written and the partition information in the manifest is updated.

Implicit write: the entity definition is derived from the dataframe structure.

- If the entity doesn't exist in the CDM folder, the implicit definition is used to create the resolved entity definition in the target CDM folder.
- If the entity exists in the CDM folder, the implicit definition is validated against the existing entity definition. If the definitions don't match an error is returned, otherwise data is written and a derived logical entity definition(s) is written into a subfolder of the entity folder. Data is written to data folder(s) within an entity subfolder. A save mode determines whether the new data overwrites or is appended to existing data, or an error is returned if data exists. The default is to return an error if data already exists.

CDM alias integration

CDM definition files use aliases in import statements to simplify the import statement and allow the location of the imported content to be late bound at execution time. Using aliases:

- Facilitates easy organization of CDM files so that related CDM definitions can be grouped together at different locations.
- Allows CDM content to be accessed from different deployed locations at runtime.

The snippet below shows the use of aliases in import statements in a CDM definition file.

```
"imports": [
{
  "corpusPath": "cdm:/foundations.cdm.json"
},
{
  "corpusPath": "core:/TrackedEntity.cdm.json"
},
{
  "corpusPath": "Customer.cdm.json"
}
]
```

In the example above, 'cdm' is used as an alias for the location of the CDM foundations file, and 'core' is used as an alias for the location of the TrackedEntity definition file.

Aliases are text labels that are matched to a namespace value in an adapter entry in a CDM config.json file. An adapter entry specifies the adapter type (for example "adls", "CDN", "GitHub", "local", etc.) and a URL that defines a location. Some adapters support other configuration options, such as a connection timeout. While aliases are arbitrary text labels, the 'cdm' alias is treated in a special manner as described below.

The Spark CDM Connector will look in the entity definition model root location for the config.json file to load. If the config.json file is at some other location or the user seeks to override the config.json file in the model root, then the user can provide the location of a config.json file using the *configPath* option. The config.json file must contain adapter entries for all the aliases used in the CDM code being resolved or an error will be reported.

By being able to override the config.json, the user can provide runtime-accessible locations for CDM definitions. Ensure that the content referenced at runtime is consistent with the definitions used when the CDM was originally authored.

By convention, the *cdm* alias is used to refer to the location of the root-level standard CDM definitions, including the foundations.cdm.json file, which includes the CDM primitive datatypes and a core set of trait definitions required for most CDM entity definitions. The *cdm* alias can be resolved like any other alias using an adapter entry in the config.json file. Alternatively, if an adapter isn't specified or a null entry is provided, then the *cdm* alias will be resolved by default to the CDM public CDN at `https://cdm-schema.microsoft.com/logical/`. The user can also use the *cdmSource* option to override how the cdm alias is resolved (see the option details below). Using the *cdmsource* option is useful if the cdm alias is the only alias used in the CDM definitions being resolved as it can avoid needing to create or reference a config.json file.

Parameters, options and save mode

For both read and write, the Spark CDM Connector library name is provided as a parameter. A set of options are used to parameterize the behavior of the connector. When writing, a save mode is also supported.

The connector library name, options and save mode are formatted as follows:

- dataframe.read.format("com.microsoft.cdm") [.option("option", "value")]*

- dataframe.write.format("com.microsoft.cdm") [.option("option", "value")]* .mode(savemode.<saveMode>)

Here's an example of how the connector is used for read, showing some of the options. More examples are provided later.

```
val readDf = spark.read.format("com.microsoft.cdm")
  .option("storage", "mystorageaccount.dfs.core.windows.net")
  .option("manifestPath", "customerleads/default.manifest.cdm.json")
  .option("entity", "Customer")
  .load()
```

Common READ and WRITE options

The following options identify the entity in the CDM folder that is either being read or written to.

OPTION	DESCRIPTION	PATTERN AND EXAMPLE USAGE
storage	The endpoint URL for the ADLS gen2 storage account with HNS enabled in which the CDM folder is located. Use the *dfs*.core.windows.net URL	<accountName>.dfs.core.windows.net "myAccount.dfs.core.windows.net"
manifestPath	The relative path to the manifest or model.json file in the storage account. For read, can be a root manifest or a submanifest or a model.json. For write, must be a root manifest.	<container>/{<folderPath>/} <manifestFileName>, "mycontainer/default.manifest.cdm.json" "models/hr/employees.manifest.cdm.json" "models/hr/employees/model.json" (read only)
entity	The name of the source or target entity in the manifest. When writing an entity for the first time in a folder, the resolved entity definition will be given this name. Entity name is case sensitive.	<entityName> "customer"
maxCDMThreads	The maximum number of concurrent reads while resolving an entity definition.	Any valid integer. for example - 5

> **NOTE**
> You no longer need to specify a logical entity definition in addition to the physical entity definition in the CDM folder on read.

Explicit write options

The following options identify the logical entity definition that defines the entity being written. The logical entity definition will be resolved to a physical definition that defines how the entity will be written.

OPTION	DESCRIPTION	PATTERN / EXAMPLE USAGE
entityDefinitionStorage	The ADLS gen2 storage account containing the entity definition. Required if different to the storage account hosting the CDM folder.	<accountName>.dfs.core.windows.net "myAccount.dfs.core.windows.net"

OPTION	DESCRIPTION	PATTERN / EXAMPLE USAGE
entityDefinitionModelRoot	The location of the model root or corpus within the account.	<container>/<folderPath> "crm/core"
entityDefinitionPath	Location of the entity. File path to the CDM definition file relative to the model root, including the name of the entity in that file.	<folderPath>/<entityName>.cdm.json/<entityName> "sales/customer.cdm.json/customer"
configPath	The container and folder path to a config.json file that contains the adapter configurations for all aliases included in the entity definition file and any directly or indirectly referenced CDM files. **Not required if the config.json is in the model root folder.**	<container><folderPath>
useCdmStandardModelRoot	Indicates the model root is located at https://cdm-schema.microsoft.com/CDM/logical/ Used to reference entity types defined in the CDM Content Delivery Network (CDN). Overrides: entityDefinitionStorage, entityDefinitionModelRoot if specified.	"useCdmStandardModelRoot"
cdmSource	Defines how the 'cdm' alias if present in CDM definition files is resolved. If this option is used, it overrides any *cdm* adapter specified in the config.json file. Values are "builtin" or "referenced". Default value is "referenced" If set to *referenced*, then the latest published standard CDM definitions at `https://cdm-schema.microsoft.com/logical/` are used. If set to *builtin* then the CDM base definitions built in to the CDM object model used by the Spark CDM Connector will be used. Note: 1). The Spark CDM Connector may not be using the latest CDM SDK so may not contain the latest published standard definitions. 2). The built-in definitions only include the top-level CDM content such as foundations.cdm.json, primitives.cdm.json, etc. If you wish to use lower-level standard CDM definitions, either use *referenced* or include a cdm adapter in the config.json.	"builtin"|"referenced".

In the example above, the full path to the customer entity definition object is: `https://myAccount.dfs.core.windows.net/models/crm/core/sales/customer.cdm.json/customer`, where 'models' is the container in ADLS.

Implicit write options

If a logical entity definition isn't specified on write, the entity will be written implicitly, based on the dataframe schema.

When writing implicitly, a timestamp column will normally be interpreted as a CDM DateTime datatype. This can be overridden to create an attribute of CDM Time datatype by providing a metadata object associated with the column that specifies the datatype. See Handling CDM Time data below for details.

Initially, this is supported for CSV files only. Support for writing time data to Parquet will be added in a later release.

Folder structure and data format options

Folder organization and file format can be changed with the following options.

OPTION	DESCRIPTION	PATTERN / EXAMPLE USAGE
useSubManifest	If true, causes the target entity to be included in the root manifest via a submanifest. The submanifest and the entity definition are written into an entity folder beneath the root. Default is false.	"true"|"false"
format	Defines the file format. Current supported file formats are CSV and Parquet. Default is "csv"	"csv"|"parquet"
delimiter	CSV only. Defines the delimiter used. Default is comma.	"|"
columnHeaders	CSV only. If true, will add a first row to data files with column headers. Default is "true"	"true"|"false"
compression	Write only. Parquet only. Defines the compression format used. Default is "snappy"	"uncompressed" | "snappy" | "gzip" | "lzo".
dataFolderFormat	Allows user-definable data folder structure within an entity folder. Allows the use of date and time values to be substituted into folder names using DateTimeFormatter formatting. Non-formatter content must be enclosed in single quotes. Default format is `"yyyy"-"MM"-"dd"` producing folder names like 2020-07-30	`year "yyyy" / month "MM"` `"Data"`

Save mode

The save mode specifies how existing entity data in the CDM folder is handled when writing a dataframe. Options are to overwrite, append to, or error if data already exists. The default save mode is ErrorIfExists

MODE	DESCRIPTION
SaveMode.Overwrite	Will overwrite the existing entity definition if it's changed and replace existing data partitions with the data partitions being written.

MODE	DESCRIPTION
SaveMode.Append	Will append data being written in new partitions alongside the existing partitions. Note: append doesn't support changing the schema; if the schema of the data being written is incompatible with the existing entity definition an error will be thrown.
SaveMode.ErrorIfExists	Will return an error if partitions already exist.

See *Folder and file organization* below for details of how data files are named and organized on write.

Authentication

There are three modes of authentication that can be used with the Spark CDM Connector to read/write the CDM metadata and data partitions: Credential Passthrough, SasToken, and App Registration.

Credential pass-through

In Synapse, the Spark CDM Connector supports use of Managed identities for Azure resource to mediate access to the Azure datalake storage account containing the CDM folder. A managed identity is automatically created for every Synapse workspace. The connector uses the managed identity of the workspace that contains the notebook in which the connector is called to authenticate to the storage accounts being addressed.

You must ensure the identity used is granted access to the appropriate storage accounts. Grant **Storage Blob Data Contributor** to allow the library to write to CDM folders, or **Storage Blob Data Reader** to allow only read access. In both cases, no extra connector options are required.

SAS token access control options

SaS Token Credential authentication to storage accounts is an extra option for authentication to storage. With SAS token authentication, the SaS token can be at the container or folder level. The appropriate permissions (read/write) are required – read manifest/partition only needs read level support, while write requires read and write support.

OPTION	DESCRIPTION	PATTERN AND EXAMPLE USAGE
sasToken	The sastoken to access the relative storageAccount with the correct permissions	<token>

Credential-based access control options

As an alternative to using a managed identity or a user identity, explicit credentials can be provided to enable the Spark CDM connector to access data. In Azure Active Directory, create an App Registration and then grant this App Registration access to the storage account using either of the following roles: **Storage Blob Data Contributor** to allow the library to write to CDM folders, or **Storage Blob Data Reader** to allow only read.

Once permissions are created, you can pass the app ID, app key, and tenant ID to the connector on each call to it using the options below. It's recommended to use Azure Key Vault to secure these values to ensure they aren't stored in clear text in your notebook file.

OPTION	DESCRIPTION	PATTERN AND EXAMPLE USAGE
appId	The app registration ID used to authenticate to the storage account	<guid>

OPTION	DESCRIPTION	PATTERN AND EXAMPLE USAGE
appKey	The registered app key or secret	<encrypted secret>
tenantId	The Azure Active Directory tenant ID under which the app is registered.	<guid>

Examples

The following examples all use appId, appKey and tenantId variables initialized earlier in the code based on an Azure app registration that has been given Storage Blob Data Contributor permissions on the storage for write and Storage Blob Data Reader permissions for read.

Read

This code reads the Person entity from the CDM folder with manifest in `mystorage.dfs.core.windows.net/cdmdata/contacts/root.manifest.cdm.json`.

```
val df = spark.read.format("com.microsoft.cdm")
 .option("storage", "mystorage.dfs.core.windows.net")
 .option("manifestPath", "cdmdata/contacts/root.manifest.cdm.json")
 .option("entity", "Person")
 .load()
```

Implicit Write – using dataframe schema only

This code writes the dataframe *df* to a CDM folder with a manifest to `mystorage.dfs.core.windows.net/cdmdata/Contacts/default.manifest.cdm.json` with an Event entity.

Event data is written as parquet files, compressed with gzip, that are appended to the folder (new files are added without deleting existing files).

```
df.write.format("com.microsoft.cdm")
 .option("storage", "mystorage.dfs.core.windows.net")
 .option("manifestPath", "cdmdata/Contacts/default.manifest.cdm.json")
 .option("entity", "Event")
 .option("format", "parquet")
 .option("compression", "gzip")
 .mode(SaveMode.Append)
 .save()
```

Explicit Write - using an entity definition stored in ADLS

This code writes the dataframe *df* to a CDM folder with manifest at `https://_mystorage_.dfs.core.windows.net/cdmdata/Contacts/root.manifest.cdm.json` with the entity Person. Person data is written as new CSV files (by default) which overwrite existing files in the folder. The Person entity definition is retrieved from `https://_mystorage_.dfs.core.windows.net/models/cdmmodels/core/Contacts/Person.cdm.json`

```
df.write.format("com.microsoft.cdm")
 .option("storage", "mystorage.dfs.core.windows.net")
 .option("manifestPath", "cdmdata/contacts/root.manifest.cdm.json")
 .option("entity", "Person")
 .option("entityDefinitionModelRoot", "cdmmodels/core")
 .option("entityDefinitionPath", "/Contacts/Person.cdm.json/Person")
 .mode(SaveMode.Overwrite)
 .save()
```

Explicit Write - using an entity defined in the CDM GitHub

This code writes the dataframe *df* to a CDM folder with the manifest at `https://_mystorage_.dfs.core.windows.net/cdmdata/Teams/root.manifest.cdm.json` and a submanifest containing the TeamMembership entity, created in a TeamMembership subdirectory. TeamMembership data is written to CSV files (the default) that overwrite any existing data files. The TeamMembership entity definition is retrieved from the CDM CDN, at: https://cdm-schema.microsoft.com/logical/core/applicationCommon/TeamMembership.cdm.json

```
df.write.format("com.microsoft.cdm")
 .option("storage", "mystorage.dfs.core.windows.net")
 .option("manifestPath", "cdmdata/Teams/root.manifest.cdm.json")
 .option("entity", "TeamMembership")
 .option("useCdmStandardModelRoot", true)
 .option("entityDefinitionPath", "core/applicationCommon/TeamMembership.cdm.json/Tea
mMembership")
 .option("useSubManifest", true)
 .mode(SaveMode.Overwrite)
 .save()
```

Other considerations

Spark to CDM datatype mapping

The following datatype mappings are applied when converting CDM to/from Spark.

SPARK	CDM
ShortType	SmallInteger
IntegerType	Integer
LongType	BigInteger
DateType	Date
Timestamp	DateTime (optionally Time, see below)
StringType	String
DoubleType	Double
DecimalType(x,y)	Decimal (x,y) (default scale and precision are 18,4)
FloatType	Float
BooleanType	Boolean

SPARK	CDM
ByteType	Byte

The CDM Binary datatype isn't supported.

Handling CDM Date, DateTime, and DateTimeOffset data

CDM Date and DateTime datatype values are handled as normal for Spark and Parquet, and in CSV are read/written in ISO 8601 format.

CDM *DateTime* datatype values are *interpreted as UTC*, and in CSV written in ISO 8601 format, for example, 2020-03-13 09:49:00Z.

CDM *DateTimeOffset* values intended for recording local time instants are handled differently in Spark and parquet from CSV. While CSV and other formats can express a local time instant as a structure, comprising a datetime and a UTC offset, formatted in CSV like, 2020-03-13 09:49:00-08:00, Parquet and Spark don't support such structures. Instead, they use a TIMESTAMP datatype that allows an instant to be recorded in UTC time (or in some unspecified time zone).

The Spark CDM connector will convert a DateTimeOffset value in CSV to a UTC timestamp. This will be persisted as a Timestamp in parquet and if subsequently persisted to CSV, the value will be serialized as a DateTimeOffset with a +00:00 offset. Importantly, there's no loss of temporal accuracy – the serialized values represent the same instant as the original values, although the offset is lost. Spark systems use their system time as the baseline and normally express time using that local time. UTC times can always be computed by applying the local system offset. For Azure systems in all regions, system time is always UTC, so all timestamp values will normally be in UTC.

As Azure system values are always UTC, when using implicit write, where a CDM definition is derived from a dataframe, timestamp columns are translated to attributes with CDM DateTime datatype, which implies a UTC time.

If it's important to persist a local time and the data will be processed in Spark or persisted in parquet, then it's recommended to use a DateTime attribute and keep the offset in a separate attribute, for example as a signed integer value representing minutes. In CDM, DateTime values are UTC, so the offset must be applied when needed to compute local time.

In most cases, persisting local time isn't important. Local times are often only required in a UI for user convenience and based on the user's time zone, so not storing a UTC time is often a better solution.

Handling CDM time data

Spark doesn't support an explicit Time datatype. An attribute with the CDM *Time* datatype is represented in a Spark dataframe as a column with a Timestamp datatype in a dataframe. When a time value is read, the timestamp in the dataframe will be initialized with the Spark epoch date 01/01/1970 plus the time value as read from the source.

When using explicit write, a timestamp column can be mapped to either a DateTime or Time attribute. If a timestamp is mapped to a Time attribute, the date portion of the timestamp is stripped off.

When using implicit write, a Timestamp column is mapped by default to a DateTime attribute. To map a timestamp column to a Time attribute, you must add a metadata object to the column in the dataframe that indicates that the timestamp should be interpreted as a time value. The code below shows how this is done in Scala.

```
val md = new MetadataBuilder().putString("dataType", "Time")
val schema = StructType(List(
StructField("ATimeColumn", TimeStampType, true, md))
```

Time value accuracy

The Spark CDM Connector supports time values in either DateTime or Time with seconds having up to six decimal places, based on the format of the data either in the file being read (CSV or Parquet) or as defined in the dataframe, enabling accuracy from single seconds to microseconds.

Folder and file naming and organization

When writing CDM folders, the default folder organization illustrated below is used.

By default, data files are written into folders created for the current date, named like '2010-07-31'. The folder structure and names can be customized using the dateFolderFormat option, described earlier.

Data file names are based on the following pattern: <entity>-<jobid>-*.<fileformat>.

The number of data partitions written can be controlled using the sparkContext.parallelize() method. The number of partitions is either determined by the number of executors in the Spark cluster or can be specified explicitly. The Scala example below creates a dataframe with two partitions.

```
val df= spark.createDataFrame(spark.sparkContext.parallelize(data, 2), schema)
```

Explicit Write (defined by a referenced entity definition)

```
+-- <CDMFolder>
     |-- default.manifest.cdm.json     << with entity ref and partition info
     +-- <Entity>
          |-- <entity>.cdm.json        << resolved physical entity definition
          |-- <data folder>
          |-- <data folder>
          +-- ...
```

Explicit Write with sub-manifest:

```
+-- <CDMFolder>
    |-- default.manifest.cdm.json       << contains reference to sub-manifest
    +-- <Entity>
         |-- <entity>.cdm.json
         |-- <entity>.manifest.cdm.json << sub-manifest with partition info
         |-- <data folder>
         |-- <data folder>
         +-- ...
```

Implicit (entity definition is derived from dataframe schema):

```
+-- <CDMFolder>
    |-- default.manifest.cdm.json
    +-- <Entity>
         |-- <entity>.cdm.json          << resolved physical entity definition
         +-- LogicalDefinition
         |   +-- <entity>.cdm.json      << logical entity definition(s)
         |-- <data folder>
         |-- <data folder>
         +-- ...
```

Implicit Write with sub-manifest:

```
+-- <CDMFolder>
    |-- default.manifest.cdm.json       << contains reference to sub-manifest
    +-- <Entity>
        |-- <entity>.cdm.json           << resolved physical entity definition
        |-- <entity>.manifest.cdm.json  << sub-manifest with reference to the entity and partition info
        +-- LogicalDefinition
        |   +-- <entity>.cdm.json       << logical entity definition(s)
        |-- <data folder>
        |-- <data folder>
        +-- ...
```

Samples

See https://github.com/Azure/spark-cdm-connector/tree/master/samples for sample code and CDM files.

Examples

The following examples all use appId, appKey and tenantId variables initialized earlier in the code based on an Azure app registration that has been given Storage Blob Data Contributor permissions on the storage for write and Storage Blob Data Reader permissions for read.

Read

This code reads the Person entity from the CDM folder with manifest in `mystorage.dfs.core.windows.net/cdmdata/contacts/root.manifest.cdm.json`.

```
val df = spark.read.format("com.microsoft.cdm")
 .option("storage", "mystorage.dfs.core.windows.net")
 .option("manifestPath", "cdmdata/contacts/root.manifest.cdm.json")
 .option("entity", "Person")
 .load()
```

Implicit write – using dataframe schema only

This code writes the dataframe *df* to a CDM folder with a manifest to `mystorage.dfs.core.windows.net/cdmdata/Contacts/default.manifest.cdm.json` with an Event entity.

Event data is written as Parquet files, compressed with gzip, that are appended to the folder (new files are added without deleting existing files).

```
df.write.format("com.microsoft.cdm")
 .option("storage", "mystorage.dfs.core.windows.net")
 .option("manifestPath", "cdmdata/Contacts/default.manifest.cdm.json")
 .option("entity", "Event")
 .option("format", "parquet")
 .option("compression", "gzip")
 .mode(SaveMode.Append)
 .save()
```

Explicit write - using an entity definition stored in ADLS

This code writes the dataframe *df* to a CDM folder with manifest at `https://mystorage.dfs.core.windows.net/cdmdata/Contacts/root.manifest.cdm.json` with the entity Person. Person data is written as new CSV files (by default) which overwrite existing files in the folder. The Person entity definition is retrieved from `https://mystorage.dfs.core.windows.net/models/cdmmodels/core/Contacts/Person.cdm.json`

```
df.write.format("com.microsoft.cdm")
 .option("storage", "mystorage.dfs.core.windows.net")
 .option("manifestPath", "cdmdata/contacts/root.manifest.cdm.json")
 .option("entity", "Person")
 .option("entityDefinitionModelRoot", "cdmmodels/core")
 .option("entityDefinitionPath", "/Contacts/Person.cdm.json/Person")
 .mode(SaveMode.Overwrite)
 .save()
```

Explicit write - using an entity defined in the CDM GitHub

This code writes the dataframe *df* to a CDM folder with the manifest at `https://_mystorage_.dfs.core.windows.net/cdmdata/Teams/root.manifest.cdm.json` and a submanifest containing the TeamMembership entity, created in a TeamMembership subdirectory. TeamMembership data is written to CSV files (the default) that overwrite any existing data files. The TeamMembership entity definition is retrieved from the CDM CDN, at: https://cdm-schema.microsoft.com/logical/core/applicationCommon/TeamMembership.cdm.json

```
df.write.format("com.microsoft.cdm")
 .option("storage", "mystorage.dfs.core.windows.net")
 .option("manifestPath", "cdmdata/Teams/root.manifest.cdm.json")
 .option("entity", "TeamMembership")
 .option("useCdmStandardModelRoot", true)
 .option("entityDefinitionPath", "core/applicationCommon/TeamMembership.cdm.json/Tea
mMembership")
 .option("useSubManifest", true)
 .mode(SaveMode.Overwrite)
 .save()
```

Other considerations

Spark to CDM datatype mapping

The following datatype mappings are applied when converting CDM to/from Spark.

SPARK	CDM
ShortType	SmallInteger
IntegerType	Integer
LongType	BigInteger
DateType	Date
Timestamp	DateTime (optionally Time, see below)
StringType	String
DoubleType	Double
DecimalType(x,y)	Decimal (x,y) (default scale and precision are 18,4)
FloatType	Float
BooleanType	Boolean

SPARK	CDM
ByteType	Byte

The CDM Binary datatype isn't supported.

Troubleshooting and known issues

- Ensure the decimal precision and scale of decimal data type fields used in the dataframe match the data type used in the CDM entity definition - requires precision and scale traits are defined on the data type. If the precision and scale aren't defined explicitly in CDM, the default used is Decimal(18,4). For model.json files, Decimal is assumed to be Decimal(18,4).
- Folder and file names in the options below shouldn't include spaces or special characters, such as "=": manifestPath, entityDefinitionModelRoot, entityDefinitionPath, dataFolderFormat.

Unsupported features

The following features aren't yet supported:

- Overriding a timestamp column to be interpreted as a CDM Time rather than a DateTime is initially supported for CSV files only. Support for writing Time data to Parquet will be added in a later release.
- Parquet Map type and arrays of primitive types and arrays of array types aren't currently supported by CDM so aren't supported by the Spark CDM Connector.

Next steps

You can now look at the other Apache Spark connectors:

- Apache Spark Kusto connector
- Apache Spark SQL connector

Azure Synapse Dedicated SQL Pool Connector for Apache Spark

8/19/2022 • 21 minutes to read • Edit Online

Introduction

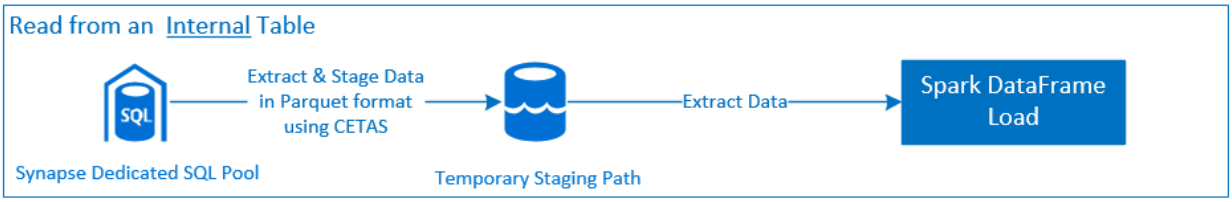
The Azure Synapse Dedicated SQL Pool Connector for Apache Spark in Azure Synapse Analytics enables efficient transfer of large data sets between the Apache Spark runtime and the Dedicated SQL pool. The connector is shipped as a default library with Azure Synapse Workspace. The connector is implemented using `Scala` language. The connector supports Scala and Python. To use the Connector with other notebook language choices, use the Spark magic command - `%%spark`.

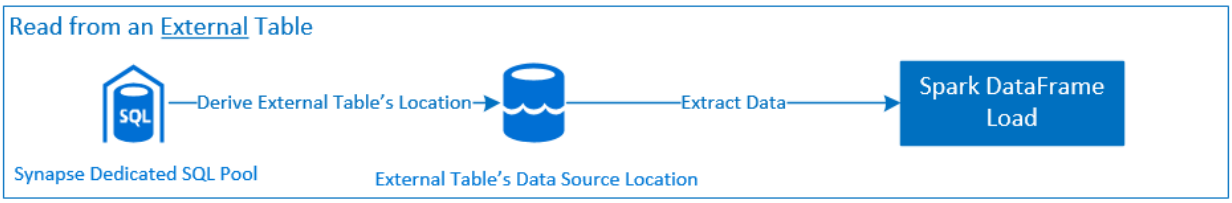
At a high-level, the connector provides the following capabilities:

- Read from Azure Synapse Dedicated SQL Pool:
  - Read large data sets from Synapse Dedicated SQL Pool Tables (Internal and External) and Views.
  - Comprehensive predicate push down support, where filters on DataFrame get mapped to corresponding SQL predicate push down.
  - Support for column pruning.
- Write to Azure Synapse Dedicated SQL Pool:
  - Ingest large volume data to Internal and External table types.
  - Supports following DataFrame save mode preferences:
    - `Append`
    - `ErrorIfExists`
    - `Ignore`
    - `Overwrite`
  - Write to External Table type supports Parquet and Delimited Text file format (example - CSV).
  - To write data to internal tables, the connector now uses COPY statement instead of CETAS/CTAS approach.
  - Enhancements to optimize end-to-end write throughput performance.
  - Introduces an optional call-back handle (a Scala function argument) that clients can use to receive post-write metrics.
    - Few examples include - number of records, duration to complete certain action, and failure reason.

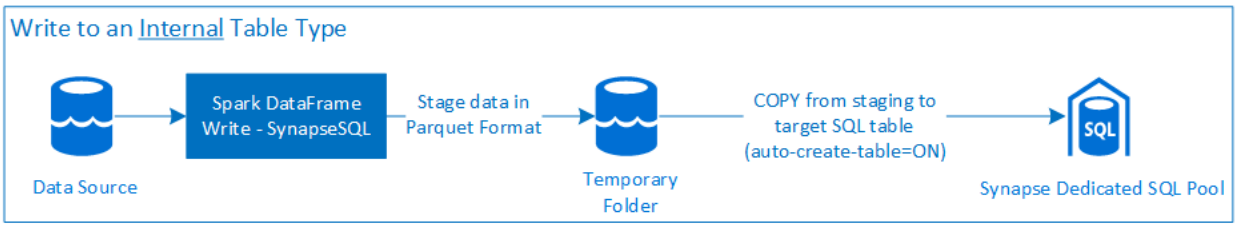
Orchestration approach

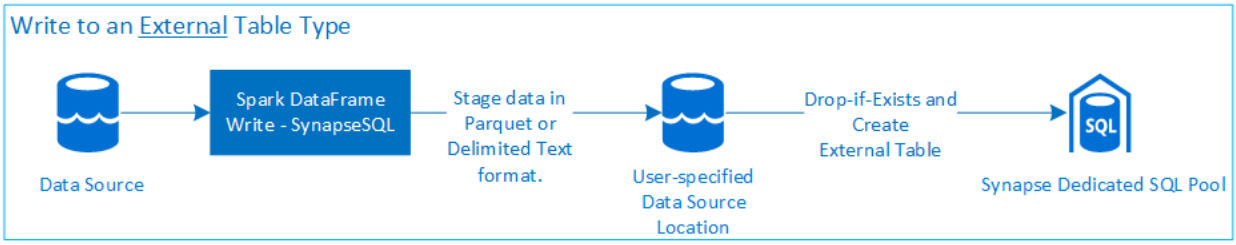
Read





Write





Pre-requisites

Pre-requisites such as setting up required Azure resources and steps to configure them are discussed in this section.

Azure resources

Review and setup following dependent Azure Resources:

- Azure Data Lake Storage - used as the primary storage account for the Azure Synapse Workspace.
- Azure Synapse Workspace - create notebooks, build and deploy DataFrame based ingress-egress workflows.
- Dedicated SQL Pool (formerly SQL DW) - provides enterprise Data Warehousing features.
- Azure Synapse Serverless Spark Pool - Spark runtime where the jobs are executed as Spark Applications.

Prepare the database

Connect to the Synapse Dedicated SQL Pool database and run following setup statements:

- Create a database user that is mapped to the Azure Active Directory User Identity used to sign in to the Azure Synapse Workspace.

```
CREATE USER [username@domain.com] FROM EXTERNAL PROVIDER;
```

- Create schema in which tables will be defined, such that the Connector can successfully write-to and read-from respective tables.

```
CREATE SCHEMA [<schema_name>];
```

Authentication

Azure Active Directory based authentication

Azure Active Directory based authentication is an integrated authentication approach. The user is required to successfully log in to the Azure Synapse Analytics Workspace.

Basic authentication

A basic authentication approach requires user to configure `username` and `password` options. Refer to the section - Configuration options to learn about relevant configuration parameters for reading from and writing to tables in Azure Synapse Dedicated SQL Pool.

Authorization

Azure Data Lake Storage Gen2

There are two ways to grant access permissions to Azure Data Lake Storage Gen2 - Storage Account:

- Role based Access Control role - Storage Blob Data Contributor role
  - Assigning the `Storage Blob Data Contributor Role` grants the User permissions to read, write and delete from the Azure Storage Blob Containers.
  - RBAC offers a coarse control approach at the container level.
- Access Control Lists (ACL)
  - ACL approach allows for fine-grained controls over specific paths and/or files under a given folder.
  - ACL checks aren't enforced if the User is already granted permissions using RBAC approach.
  - There are two broad types of ACL permissions:
    - Access Permissions (applied at a specific level or object).
    - Default Permissions (automatically applied for all child objects at the time of their creation).
  - Type of permissions include:
    - `Execute` enables ability to traverse or navigate the folder hierarchies.
    - `Read` enables ability to read.
    - `Write` enables ability to write.
  - It's important to configure ACLs such that the Connector can successfully write and read from the storage locations.

> **NOTE**
>
> - If you'd like to run notebooks using Synapse Workspace pipelines you must also grant above listed access permissions to the Synapse Workspace default managed identity. The workspace's default managed identity name is same as the name of the workspace.
> - To use the Synapse workspace with secured storage accounts, a managed private end point must be configured from the notebook. The managed private end point must be approved from the ADLS Gen2 storage account's `Private endpoint connections` section in the `Networking` pane.

Azure Synapse Dedicated SQL Pool

To enable successful interaction with Azure Synapse Dedicated SQL Pool, following authorization is necessary unless you're a user also configured as an `Active Directory Admin` on the Dedicated SQL End Point:

- Read scenario
  - Grant the user `db_exporter` using the system stored procedure `sp_addrolemember`.

```
EXEC sp_addrolemember 'db_exporter', [<your_domain_user>@<your_domain_name>.com];
```

- Write scenario
  - Connector uses the COPY command to write data from staging to the internal table's managed

location.

- Configure required permissions described here.
- Following is a quick access snippet of the same:

```
--Make sure your user has the permissions to CREATE tables in the [dbo] schema
GRANT CREATE TABLE TO [<your_domain_user>@<your_domain_name>.com];
GRANT ALTER ON SCHEMA::<target_database_schema_name> TO
[<your_domain_user>@<your_domain_name>.com];

--Make sure your user has ADMINISTER DATABASE BULK OPERATIONS permissions
GRANT ADMINISTER DATABASE BULK OPERATIONS TO
[<your_domain_user>@<your_domain_name>.com];

--Make sure your user has INSERT permissions on the target table
GRANT INSERT ON <your_table> TO [<your_domain_user>@<your_domain_name>.com]
```

API documentation

Azure Synapse Dedicated SQL Pool Connector for Apache Spark - API Documentation.

Configuration options

To successfully bootstrap and orchestrate the read or write operation, the Connector expects certain configuration parameters. The object definition - `com.microsoft.spark.sqlanalytics.utils.Constants` provides a list of standardized constants for each parameter key.

Following is the list of configuration options based on usage scenario:

- **Read using Azure AD based authentication**
  - Credentials are auto-mapped, and user isn't required to provide specific configuration options.
  - Three-part table name argument on `synapsesql` method is required to read from respective table in Azure Synapse Dedicated SQL Pool.
- **Read using basic authentication**
  - Azure Synapse Dedicated SQL End Point
    - `Constants.SERVER` - Synapse Dedicated SQL Pool End Point (Server FQDN)
    - `Constants.USER` - SQL User Name.
    - `Constants.PASSWORD` - SQL User Password.
  - Azure Data Lake Storage (Gen 2) End Point - Staging Folders
    - `Constants.DATA_SOURCE` - Storage path set on the data source location parameter is used for data staging.
- **Write using Azure AD based authentication**
  - Azure Synapse Dedicated SQL End Point
    - By default, the Connector infers the Synapse Dedicated SQL end point by using the database name set on the `synapsesql` method's three-part table name parameter.
    - Alternatively, users can use the `Constants.SERVER` option to specify the sql end point. Ensure the end point hosts the corresponding database with respective schema.
  - Azure Data Lake Storage (Gen 2) End Point - Staging Folders
    - For Internal Table Type:
      - Configure either `Constants.TEMP_FOLDER` or `Constants.DATA_SOURCE` option.
      - If user chose to provide `Constants.DATA_SOURCE` option, staging folder will be derived by using the `location` value from the DataSource.
      - If both are provided, then the `Constants.TEMP_FOLDER` option value will be used.

    - In the absence of a staging folder option, the Connector will derive one based on the runtime configuration - `spark.sqlanalyticsconnector.stagingdir.prefix`.
  - For External Table Type:
    - `Constants.DATA_SOURCE` is a required configuration option.
    - The connector uses the storage path set on the data source's location parameter in combination with the `location` argument to the `synapsesql` method and derives the absolute path to persist external table data.
    - If the `location` argument to `synapsesql` method isn't specified, then the connector will derive the location value as `<base_path>/dbName/schemaName/tableName`.
- **Write using basic authentication**
  - Azure Synapse Dedicated SQL End Point
    - `Constants.SERVER` - - Synapse Dedicated SQL Pool End Point (Server FQDN).
    - `Constants.USER` - SQL User Name.
    - `Constants.PASSWORD` - SQL User Password.
    - `Constants.STAGING_STORAGE_ACCOUNT_KEY` associated with Storage Account that hosts `Constants.TEMP_FOLDERS` (internal table types only) or `Constants.DATA_SOURCE`.
  - Azure Data Lake Storage (Gen 2) End Point - Staging Folders
    - SQL basic authentication credentials don't apply to access storage end points.
    - Hence, ensure to assign relevant storage access permissions as described in the section Azure Data Lake Storage Gen2.

Code templates

This section presents reference code templates to describe how to use and invoke the Azure Synapse Dedicated SQL Pool Connector for Apache Spark.

> **NOTE**
>
> Using the Connector in Python-
>
> - The connector is supported in Python for Spark 3 only. For Spark 2.4, we can use the Scala connector API to interact with content from a DataFrame in PySpark by using DataFrame.createOrReplaceTempView or DataFrame.createOrReplaceGlobalTempView. See Section - Using materialized data across cells.
> - The call back handle is not available in Python.

Read from Azure Synapse Dedicated SQL Pool

Read Request - `synapsesql` method signature

- Scala
- Python

```
synapsesql(tableName:String) => org.apache.spark.sql.DataFrame
```

Read using Azure AD based authentication

- Scala
- Python

```
//Use case is to read data from an internal table in Synapse Dedicated SQL Pool DB
//Azure Active Directory based authentication approach is preferred here.
import org.apache.spark.sql.DataFrame
import com.microsoft.spark.sqlanalytics.utils.Constants
import org.apache.spark.sql.SqlAnalyticsConnector._

//Read from existing internal table
val dfToReadFromTable:DataFrame = spark.read.
    //If `Constants.SERVER` is not provided, the `<database_name>` from the three-part table name argument
    //to `synapsesql` method is used to infer the Synapse Dedicated SQL End Point.
    option(Constants.SERVER, "<sql-server-name>.sql.azuresynapse.net").
    //Defaults to storage path defined in the runtime configurations
    option(Constants.TEMP_FOLDER,
"abfss://<container_name>@<storage_account_name>.dfs.core.windows.net/<some_base_path_for_temporary_staging_
folders>").
    //Three-part table name from where data will be read.
    synapsesql("<database_name>.<schema_name>.<table_name>").
    //Column-pruning i.e., query select column values.
    select("<some_column_1>", "<some_column_5>", "<some_column_n>").
    //Push-down filter criteria that gets translated to SQL Push-down Predicates.
    filter(col("Title").startsWith("E")).
    //Fetch a sample of 10 records
    limit(10)

//Show contents of the dataframe
dfToReadFromTable.show()
```

Read using basic authentication

- Scala
- Python

```
//Use case is to read data from an internal table in Synapse Dedicated SQL Pool DB
//Azure Active Directory based authentication approach is preferred here.
import org.apache.spark.sql.DataFrame
import com.microsoft.spark.sqlanalytics.utils.Constants
import org.apache.spark.sql.SqlAnalyticsConnector._

//Read from existing internal table
val dfToReadFromTable:DataFrame = spark.read.
    //If `Constants.SERVER` is not provided, the `<database_name>` from the three-part table name argument
    //to `synapsesql` method is used to infer the Synapse Dedicated SQL End Point.
    option(Constants.SERVER, "<sql-server-name>.sql.azuresynapse.net").
    //Set database user name
    option(Constants.USER, "<user_name>").
    //Set user's password to the database
    option(Constants.PASSWORD, "<user_password>").
    //Set name of the data source definition that is defined with database scoped credentials.
    //Data extracted from the SQL query will be staged to the storage path defined on the data source's
location setting.
    option(Constants.DATA_SOURCE, "<data_source_name>").
    //Three-part table name from where data will be read.
    synapsesql("<database_name>.<schema_name>.<table_name>").
    //Column-pruning i.e., query select column values.
    select("<some_column_1>", "<some_column_5>", "<some_column_n>").
    //Push-down filter criteria that gets translated to SQL Push-down Predicates.
    filter(col("Title").startsWith("E")).
    //Fetch a sample of 10 records
    limit(10)

//Show contents of the dataframe
dfToReadFromTable.show()
```

Write to Azure Synapse Dedicated SQL Pool

Write Request - `synapsesql` method signature

The method signature for the Connector version built for Spark 2.4.8 has one less argument, than that applied to the Spark 3.1.2 version. Following are the two method signatures:

- Spark Pool Version 2.4.8

```
synapsesql(tableName:String,
           tableType:String = Constants.INTERNAL,
           location:Option[String] = None):Unit
```

- Spark Pool Version 3.1.2

- Scala
- Python

```
synapsesql(tableName:String,
           tableType:String = Constants.INTERNAL,
           location:Option[String] = None,
           callBackHandle=Option[(Map[String, Any], Option[Throwable])=>Unit]):Unit
```

Write using Azure AD based authentication

Following is a comprehensive code template that describes how to use the Connector for write scenarios:

- Scala
- Python

```
//Add required imports
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.SaveMode
import com.microsoft.spark.sqlanalytics.utils.Constants
import org.apache.spark.sql.SqlAnalyticsConnector._

//Define read options for example, if reading from CSV source, configure header and delimiter options.
val
pathToInputSource="abfss://<storage_container_name>@<storage_account_name>.dfs.core.windows.net/<some_folder
>/<some_dataset>.csv"

//Define read configuration for the input CSV
val dfReadOptions:Map[String, String] = Map("header" -> "true", "delimiter" -> ",")

//Initialize DataFrame that reads CSV data from a given source
val readDF:DataFrame=spark.
            read.
            options(dfReadOptions).
            csv(pathToInputSource).
            limit(1000) //Reads first 1000 rows from the source CSV input.

//Setup and trigger the read DataFrame for write to Synapse Dedicated SQL Pool.
//Fully qualified SQL Server DNS name can be obtained using one of the following methods:
//    1. Synapse Workspace - Manage Pane - SQL Pools - <Properties view of the corresponding Dedicated SQL
Pool>
//    2. From Azure Portal, follow the bread-crumbs for <Portal_Home> -> <Resource_Group> -> <Dedicated SQL
Pool> and then go to Connection Strings/JDBC tab.
//If `Constants.SERVER` is not provided, the value will be inferred by using the `database_name` in the
three-part table name argument to the `synapsesql` method.
//Like-wise, if `Constants.TEMP_FOLDER` is not provided, the connector will use the runtime staging
directory config (see section on Configuration Options for details).
val writeOptionsWithAADAuth:Map[String, String] = Map(Constants.SERVER -> "<dedicated-pool-sql-server-
name>.sql.azuresynapse.net",
                                            Constants.TEMP_FOLDER ->
"abfss://<storage_container_name>@<storage_account_name>.dfs.core.windows.net/<some_temp_folder>")

//Setup optional callback/feedback function that can receive post write metrics of the job performed.
var errorDuringWrite:Option[Throwable] = None
val callBackFunctionToReceivePostWriteMetrics: (Map[String, Any], Option[Throwable]) => Unit =
    (feedback: Map[String, Any], errorState: Option[Throwable]) => {
    println(s"Feedback map - ${feedback.map{case(key, value) => s"$key -> $value"}.mkString("
{",",\n","}")}")
    errorDuringWrite = errorState
}

//Configure and submit the request to write to Synapse Dedicated SQL Pool (note - default SaveMode is set to
ErrorIfExists)
//Sample below is using AAD-based authentication approach; See further examples to leverage SQL Basic auth.
readDF.
    write.
    //Configure required configurations.
    options(writeOptionsWithAADAuth).
    //Choose a save mode that is apt for your use case.
    mode(SaveMode.Overwrite).
    synapsesql(tableName = "<database_name>.<schema_name>.<table_name>",
                //For external table type value is Constants.EXTERNAL
                tableType = Constants.INTERNAL,
                //Optional parameter that is used to specify external table's base folder; defaults to
`database_name/schema_name/table_name`
                location = None,
                //Optional parameter to receive a callback.
                callBackHandle = Some(callBackFunctionToReceivePostWriteMetrics))

//If write request has failed, raise an error and fail the Cell's execution.
if(errorDuringWrite.isDefined) throw errorDuringWrite.get
```

Write using basic authentication

Following code snippet replaces the write definition described in the Write using Azure AD based authentication section, to submit write request using SQL basic authentication approach:

- Scala
- Python

```
//Define write options to use SQL basic authentication
val writeOptionsWithBasicAuth:Map[String, String] = Map(Constants.SERVER -> "<dedicated-pool-sql-server-
name>.sql.azuresynapse.net",
                                        //Set database user name
                                        Constants.USER -> "<user_name>",
                                        //Set database user's password
                                        Constants.PASSWORD -> "<user_password>",
                                        //Required only when writing to an external table. For write to
internal table, this can be used instead of TEMP_FOLDER option.
                                        Constants.DATA_SOURCE -> "<Name of the datasource as defined in
the target database>"
                                        //To be used only when writing to internal tables. Storage path
will be used for data staging.
                                        Constants.TEMP_FOLDER ->
"abfss://<storage_container_name>@<storage_account_name>.dfs.core.windows.net/<some_temp_folder>")

//Configure and submit the request to write to Synapse Dedicated SQL Pool.
readDF.
    write.
    options(writeOptionsWithBasicAuth).
    //Choose a save mode that is apt for your use case.
    mode(SaveMode.Overwrite).
    synapsesql(tableName = "<database_name>.<schema_name>.<table_name>",
                //For external table type value is Constants.EXTERNAL
                tableType = Constants.INTERNAL,
                //Not required for writing to an internal table
                location = None,
                //Optional parameter.
                callBackHandle = Some(callBackFunctionToReceivePostWriteMetrics))
```

In a basic authentication approach, in order to read data from a source storage path other configuration options are required. Following code snippet provides an example to read from an Azure Data Lake Storage Gen2 data source using Service Principal credentials:

```
//Specify options that Spark runtime must support when interfacing and consuming source data
val storageAccountName="<storageAccountName>"
val storageContainerName="<storageContainerName>"
val subscriptionId="<AzureSubscriptionID>"
val spnClientId="<ServicePrincipalClientID>"
val spnSecretKeyUsedAsAuthCred="<spn_secret_key_value>"
val dfReadOptions:Map[String, String]=Map("header"->"true",
                                "delimiter"->",",
                                "fs.defaultFS" ->
s"abfss://$storageContainerName@$storageAccountName.dfs.core.windows.net",
                                s"fs.azure.account.auth.type.$storageAccountName.dfs.core.windows.net" ->
"OAuth",

s"fs.azure.account.oauth.provider.type.$storageAccountName.dfs.core.windows.net" ->
                                    "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
                                "fs.azure.account.oauth2.client.id" -> s"$spnClientId",
                                "fs.azure.account.oauth2.client.secret" -> s"$spnSecretKeyUsedAsAuthCred",
                                "fs.azure.account.oauth2.client.endpoint" ->
s"https://login.microsoftonline.com/$subscriptionId/oauth2/token",
                                "fs.AbstractFileSystem.abfss.impl" -> "org.apache.hadoop.fs.azurebfs.Abfs",
                                "fs.abfss.impl" -> "org.apache.hadoop.fs.azurebfs.SecureAzureBlobFileSystem")
//Initialize the Storage Path string, where source data is maintained/kept.
val
pathToInputSource=s"abfss://$storageContainerName@$storageAccountName.dfs.core.windows.net/<base_path_for_so
urce_data>/<specific_file (or) collection_of_files>"
//Define data frame to interface with the data source
val df:DataFrame = spark.
           read.
           options(dfReadOptions).
           csv(pathToInputSource).
           limit(100)
```

Supported DataFrame save modes

Following save modes are supported when writing source data to a destination table in Azure Synapse Dedicated SQL Pool:

- ErrorIfExists (default save mode)
  - If destination table exists, then the write is aborted with an exception returned to the callee. Else, a new table is created with data from the staging folders.
- Ignore
  - If the destination table exists, then the write will ignore the write request without returning an error. Else, a new table is created with data from the staging folders.
- Overwrite
  - If the destination table exists, then existing data in the destination is replaced with data from the staging folders. Else, a new table is created with data from the staging folders.
- Append
  - If the destination table exists, then the new data is appended to it. Else, a new table is created with data from the staging folders.

Write request callback handle

The new write path API changes introduced an experimental feature to provide the client with a key->value map of post-write metrics. Keys for the metrics are defined in the new Object definition - `Constants.FeedbackConstants`. Metrics can be retrieved as a JSON string by passing in the callback handle (a `Scala Function`). Following is the function signature:

```
//Function signature is expected to have two arguments - a `scala.collection.immutable.Map[String, Any]` and
an Option[Throwable]
//Post-write if there's a reference of this handle passed to the `synapsesql` signature, it will be invoked
by the closing process.
//These arguments will have valid objects in either Success or Failure case. In case of Failure the second
argument will be a `Some(Throwable)`.
(Map[String, Any], Option[Throwable]) => Unit
```

Following are some notable metrics (presented in camel case):

- `WriteFailureCause`
- `DataStagingSparkJobDurationInMilliseconds`
- `NumberOfRecordsStagedForSQLCommit`
- `SQLStatementExecutionDurationInMilliseconds`
- `rows_processed`

Following is a sample JSON string with post-write metrics:

```
{
 SparkApplicationId -> <spark_yarn_application_id>,
 SQLStatementExecutionDurationInMilliseconds -> 10113,
 WriteRequestReceivedAtEPOCH -> 1647523790633,
 WriteRequestProcessedAtEPOCH -> 1647523808379,
 StagingDataFileSystemCheckDurationInMilliseconds -> 60,
 command -> "COPY INTO [schema_name].[table_name] ...",
 NumberOfRecordsStagedForSQLCommit -> 100,
 DataStagingSparkJobEndedAtEPOCH -> 1647523797245,
 SchemaInferenceAssertionCompletedAtEPOCH -> 1647523790920,
 DataStagingSparkJobDurationInMilliseconds -> 5252,
 rows_processed -> 100,
 SaveModeApplied -> TRUNCATE_COPY,
 DurationInMillisecondsToValidateFileFormat -> 75,
 status -> Completed,
 SparkApplicationName -> <spark_application_name>,
 ThreePartFullyQualifiedTargetTableName -> <database_name>.<schema_name>.<table_name>,
 request_id -> <query_id_as_retrieved_from_synapse_dedicated_sql_db_query_reference>,
 StagingFolderConfigurationCheckDurationInMilliseconds -> 2,
 JDBCConfigurationsSetupAtEPOCH -> 193,
 StagingFolderConfigurationCheckCompletedAtEPOCH -> 1647523791012,
 FileFormatValidationsCompletedAtEPOCHTime -> 1647523790995,
 SchemaInferenceCheckDurationInMilliseconds -> 91,
 SaveModeRequested -> Overwrite,
 DataStagingSparkJobStartedAtEPOCH -> 1647523791993,
 DurationInMillisecondsTakenToGenerateWriteSQLStatements -> 4
}
```

More code samples

Using materialized data across cells

Spark DataFrame's `createOrReplaceTempView` can be used to access data fetched in another cell, by registering a temporary view.

- Cell where data is fetched (say with Notebook language preference as `Scala`)

```
//Necessary imports
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.SaveMode
import com.microsoft.spark.sqlanalytics.utils.Constants
import org.apache.spark.sql.SqlAnalyticsConnector._

//Configure options and read from Synapse Dedicated SQL Pool.
val readDF = spark.read.
    //Set Synapse Dedicated SQL End Point name.
    option(Constants.SERVER, "<synapse-dedicated-sql-end-point>.sql.azuresynapse.net").
    //Set database user name.
    option(Constants.USER, "<user_name>").
    //Set database user's password.
    option(Constants.PASSWORD, "<user_password>").
    //Set name of the data source definition that is defined with database scoped credentials.
    option(Constants.DATA_SOURCE,"<data_source_name>").
    //Set the three-part table name from which the read must be performed.
    synapsesql("<database_name>.<schema_name>.<table_name>").
    //Optional - specify number of records the DataFrame would read.
    limit(10)
//Register the temporary view (scope - current active Spark Session)
readDF.createOrReplaceTempView("<temporary_view_name>")
```

- Now, change the language preference on the Notebook to `PySpark (Python)` and fetch data from the registered view `<temporary_view_name>`

```
spark.sql("select * from <temporary_view_name>").show()
```

Response handling

Invoking `synapsesql` has two possible end states - Success or a Failed State. This section describes how to handle the request response for each scenario.

Read request response

Upon completion, the read response snippet is displayed in the cell's output. Failure in the current cell will also cancel subsequent cell executions. Detailed error information is available in the Spark Application Logs.

Write request response

By default, a write response is printed to the cell output. On failure, the current cell is marked as failed, and subsequent cell executions will be aborted. The other approach is to pass the callback handle option to the `synapsesql` method. The callback handle will provide programmatic access to the write response.

Other considerations

- When reading from the Azure Synapse Dedicated SQL Pool tables:
  - Consider applying necessary filters on the DataFrame to take advantage of the Connector's column-pruning feature.
  - Read scenario doesn't support the `TOP(n-rows)` clause, when framing the `SELECT` query statements. The choice to limit data is to use the DataFrame's limit(.) clause.
    - Refer the example - Using materialized data across cells section.
- When writing to the Azure Synapse Dedicated SQL Pool tables:
  - For internal table types:
    - Tables are created with ROUND_ROBIN data distribution.
    - Column types are inferred from the DataFrame that would read data from source. String columns are mapped to `NVARCHAR(4000)`.
  - For external table types:
    - DataFrame's initial parallelism drives the data organization for the external table.

    - Column types are inferred from the DataFrame that would read data from source.
  - Better data distribution across executors can be achieved by tuning the `spark.sql.files.maxPartitionBytes` and the DataFrame's `repartition` parameter.
  - When writing large data sets, it's important to factor in the impact of DWU Performance Level setting that limits transaction size.
- Monitor Azure Data Lake Storage Gen2 utilization trends to spot throttling behaviors that can impact read and write performance.

References

- Runtime library versions
- Azure Storage
- Dedicated SQL pool

Azure SQL Database and SQL Server connector for Apache Spark

8/19/2022 • 2 minutes to read • Edit Online

The Apache Spark connector for Azure SQL Database and SQL Server enables these databases to act as input data sources and output data sinks for Apache Spark jobs. It allows you to use real-time transactional data in big data analytics and persist results for ad-hoc queries or reporting.

Compared to the built-in JDBC connector, this connector provides the ability to bulk insert data into SQL databases. It can outperform row-by-row insertion with 10x to 20x faster performance. The Spark connector for SQL Server and Azure SQL Database also supports Azure Active Directory (Azure AD) authentication, enabling you to connect securely to your Azure SQL databases from Azure Synapse Analytics.

This article covers how to use the DataFrame API to connect to SQL databases using the MS SQL connector. This article provides detailed examples using the PySpark API. For all of the supported arguments and samples for connecting to SQL databases using the MS SQL connector, see Azure Data SQL samples.

Connection details

In this example, we will use the Microsoft Spark utilities to facilitate acquiring secrets from a pre-configured Key Vault. To learn more about Microsoft Spark utilities, please visit introduction to Microsoft Spark Utilities.

```
servername = "<< server name >>"
dbname = "<< database name >>"
url = servername + ";" + "databaseName=" + dbname + ";"
dbtable = "<< table name >> "
user = "<< username >>"
password = mssparkutils.credentials.getSecret('azure key vault name','secret name')
```

NOTE

Currently, there is no linked service or AAD pass-through support with the Azure SQL connector.

Use the Azure SQL and SQL Server connector

Read data

```
#Read from SQL table using MS SQL Connector
print("read data from SQL server table  ")
jdbcDF = spark.read \
        .format("com.microsoft.sqlserver.jdbc.spark") \
        .option("url", url) \
        .option("dbtable", dbtable) \
        .option("user", user) \
        .option("password", password).load()

jdbcDF.show(5)
```

Write data

```
try:
  df.write \
    .format("com.microsoft.sqlserver.jdbc.spark") \
    .mode("overwrite") \
    .option("url", url) \
    .option("dbtable", dbtable) \
    .option("user", user) \
    .option("password", password) \
    .save()
except ValueError as error :
    print("MSSQL Connector write failed", error)

print("MSSQL Connector write(overwrite) succeeded  ")
```

Append data

```
try:
  df.write \
    .format("com.microsoft.sqlserver.jdbc.spark") \
    .mode("append") \
    .option("url", url) \
    .option("dbtable", table_name) \
    .option("user", username) \
    .option("password", password) \
    .save()
except ValueError as error :
    print("Connector write failed", error)
```

Azure Active Directory authentication

Python example with service principal

```
import adal

# Located in App Registrations from Azure Portal
tenant_id = "<< tenant id >> "

# Located in App Registrations from Azure Portal
resource_app_id_url = "https://database.windows.net/"

# Authority
authority = "https://login.windows.net/" + tenant_id

context = adal.AuthenticationContext(authority)
token = context.acquire_token_with_client_credentials(resource_app_id_url, service_principal_id,
service_principal_secret)
access_token = token["accessToken"]

jdbc_df = spark.read \
        .format("com.microsoft.sqlserver.jdbc.spark") \
        .option("url", url) \
        .option("dbtable", dbtable) \
        .option("accessToken", access_token) \
        .option("encrypt", "true") \
        .option("hostNameInCertificate", "*.database.windows.net") \
        .load()
```

Python example with Active Directory password

```
jdbc_df = spark.read \
        .format("com.microsoft.sqlserver.jdbc.spark") \
        .option("url", url) \
        .option("dbtable", table_name) \
        .option("authentication", "ActiveDirectoryPassword") \
        .option("user", user_name) \
        .option("password", password) \
        .option("encrypt", "true") \
        .option("hostNameInCertificate", "*.database.windows.net") \
        .load()
```

Next steps

- Learn more about the SQL Server and Azure SQL connector
- View Azure Data SQL Samples

Use external Hive Metastore for Synapse Spark Pool

8/19/2022 • 8 minutes to read • Edit Online

Azure Synapse Analytics allows Apache Spark pools in the same workspace to share a managed HMS (Hive Metastore) compatible metastore as their catalog. When customers want to persist the Hive catalog metadata outside of the workspace, and share catalog objects with other computational engines outside of the workspace, such as HDInsight and Azure Databricks, they can connect to an external Hive Metastore. In this article, you can learn how to connect Synapse Spark to an external Apache Hive Metastore.

Supported Hive Metastore versions

The feature works with both Spark 2.4 and Spark 3.1. The following table shows the supported Hive Metastore versions for each Spark version.

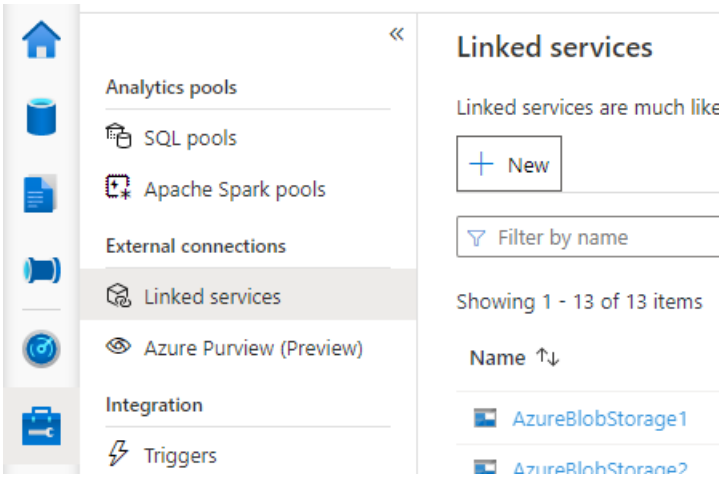
SPARK VERSION	HMS 0.13.X	HMS 1.2.X	HMS 2.1.X	HMS 2.3.X	HMS 3.1.X
2.4	Yes	Yes	Yes	Yes	No
3.1	Yes	Yes	Yes	Yes	Yes

Set up linked service to Hive Metastore

> **NOTE**
> Only Azure SQL Database and Azure Database for MySQL are supported as an external Hive Metastore. And currently we only support User-Password authentication. If the provided database is blank, please provision it via Hive Schema Tool to create database schema.

Follow below steps to set up a linked service to the external Hive Metastore in Synapse workspace.

1. Open Synapse Studio, go to **Manage** > **Linked services** at left, click **New** to create a new linked service.



2. Choose **Azure SQL Database** or **Azure Database for MySQL** based on your database type, click **Continue**.

3. Provide **Name** of the linked service. Record the name of the linked service, this info will be used to

configure Spark shortly.

4. You can either select **Azure SQL Database/Azure Database for MySQL** for the external Hive Metastore from Azure subscription list, or enter the info manually.
5. Provide **User name** and **Password** to set up the connection.
6. **Test connection** to verify the username and password.
7. Click **Create** to create the linked service.

Test connection and get the metastore version in notebook

Some network security rule settings may block access from Spark pool to the external Hive Metastore DB. Before you configure the Spark pool, run below code in any Spark pool notebook to test connection to the external Hive Metastore DB.

You can also get your Hive Metastore version from the output results. The Hive Metastore version will be used in the Spark configuration.

Connection testing code for Azure SQL

```
%%spark
import java.sql.DriverManager
/** this JDBC url could be copied from Azure portal > Azure SQL database > Connection strings > JDBC **/
val url = s"jdbc:sqlserver://{your_servername_here}.database.windows.net:1433;database=
{your_database_here};user={your_username_here};password=
{your_password_here};encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.database.windows.net;
loginTimeout=30;"
try {
    val connection = DriverManager.getConnection(url)
    val result = connection.createStatement().executeQuery("select t.SCHEMA_VERSION from VERSION t")
    result.next();
    println(s"Successful to test connection. Hive Metastore version is ${result.getString(1)}")
} catch {
    case ex: Throwable => println(s"Failed to establish connection:\n $ex")
}
```

Connection testing code for Azure Database for MySQL

```
%%spark
import java.sql.DriverManager
/** this JDBC url could be copied from Azure portal > Azure Database for MySQL > Connection strings > JDBC
**/
val url = s"jdbc:mysql://{your_servername_here}.mysql.database.azure.com:3306/{your_database_here}?
useSSL=true"
try {
    val connection = DriverManager.getConnection(url, "{your_username_here}", "{your_password_here}");
    val result = connection.createStatement().executeQuery("select t.SCHEMA_VERSION from VERSION t")
    result.next();
    println(s"Successful to test connection. Hive Metastore version is ${result.getString(1)}")
} catch {
    case ex: Throwable => println(s"Failed to establish connection:\n $ex")
}
```

Configure Spark to use the external Hive Metastore

After creating the linked service to the external Hive Metastore successfully, you need to setup a few Spark configurations to use the external Hive Metastore. You can both set up the configuration at Spark pool level, or at Spark session level.

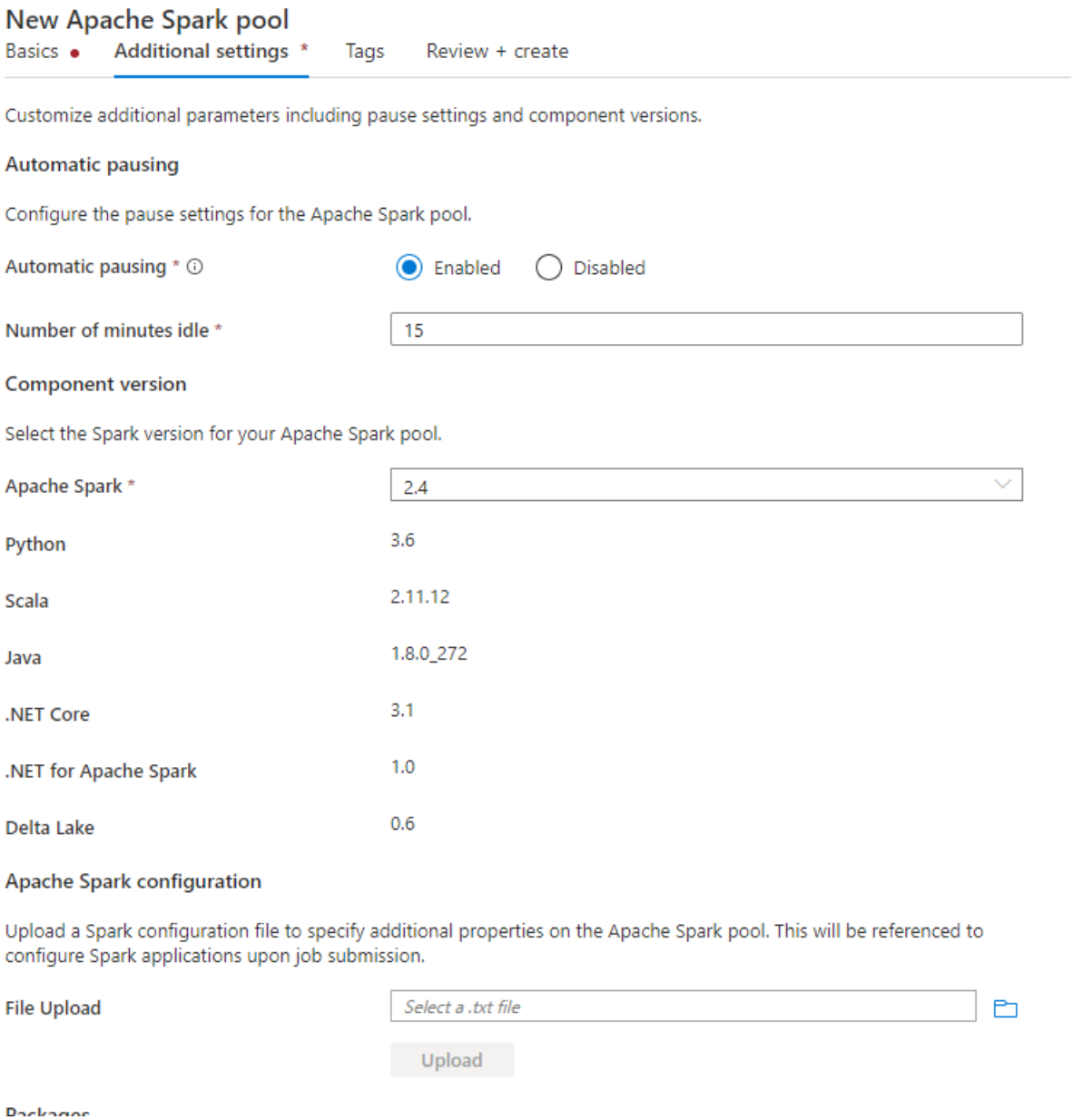
Here are the configurations and descriptions:

> **NOTE**
>
> Synapse aims to work smoothly with computes from HDI. However HMS 3.1 in HDI 4.0 is not fully compatible with the OSS HMS 3.1. For OSS HMS 3.1, please check here.

SPARK CONFIG	DESCRIPTION
`spark.sql.hive.metastore.version`	Supported versions: • `0.13` • `1.2` • `2.1` • `2.3` • `3.1` Make sure you use the first 2 parts without the 3rd part
`spark.sql.hive.metastore.jars`	• Version 0.13: `/opt/hive-metastore/lib-0.13/*:/usr/hdp/current/hadoop-client/lib/*` • Version 1.2: `/opt/hive-metastore/lib-1.2/*:/usr/hdp/current/hadoop-client/lib/*` • Version 2.1: `/opt/hive-metastore/lib-2.1/*:/usr/hdp/current/hadoop-client/lib/*` • Version 2.3: `/opt/hive-metastore/lib-2.3/*:/usr/hdp/current/hadoop-client/lib/*` • Version 3.1: `/opt/hive-metastore/lib-3.1/*:/usr/hdp/current/hadoop-client/lib/*`
`spark.hadoop.hive.synapse.externalmetastore.linkedservice.name`	Name of your linked service

Configure at Spark pool level

When creating the Spark pool, under **Additional Settings** tab, put below configurations in a text file and upload it in **Apache Spark configuration** section. You can also use the context menu for an existing Spark pool, choose Apache Spark configuration to add these configurations.



Update metastore version and linked service name, and save below configs in a text file for Spark pool configuration:

```
spark.sql.hive.metastore.version <your hms version, Make sure you use the first 2 parts without the 3rd part>
spark.hadoop.hive.synapse.externalmetastore.linkedservice.name <your linked service name>
spark.sql.hive.metastore.jars /opt/hive-metastore/lib-<your hms version, 2 parts>/*:/usr/hdp/current/hadoop-client/lib/*
```

Here is an example for metastore version 2.1 with linked service named as HiveCatalog21:

```
spark.sql.hive.metastore.version 2.1
spark.hadoop.hive.synapse.externalmetastore.linkedservice.name HiveCatalog21
spark.sql.hive.metastore.jars /opt/hive-metastore/lib-2.1/*:/usr/hdp/current/hadoop-client/lib/*
```

Configure at Spark session level

For notebook session, you can also configure the Spark session in notebook using `%%configure` magic command. Here is the code.

```
%%configure -f
{
    "conf":{
        "spark.sql.hive.metastore.version":"<your hms version, 2 parts>",
        "spark.hadoop.hive.synapse.externalmetastore.linkedservice.name":"<your linked service name>",
        "spark.sql.hive.metastore.jars":"/opt/hive-metastore/lib-<your hms version, 2
parts>/*:/usr/hdp/current/hadoop-client/lib/*"
    }
}
```

For batch job, same configuration can also be applied via `SparkConf`.

Run queries to verify the connection

After all these settings, try listing catalog objects by running below query in Spark notebook to check the connectivity to the external Hive Metastore.

```
spark.sql("show databases").show()
```

Set up storage connection

The linked service to Hive Metastore database just provides access to Hive catalog metadata. To query the existing tables, you need to set up connection to the storage account that stores the underlying data for your Hive tables as well.

Set up connection to Azure Data Lake Storage Gen 2

Workspace primary storage account

If the underlying data of your Hive tables is stored in the workspace primary storage account, you don't need to do extra settings. It will just work as long as you followed storage setting up instructions during workspace creation.

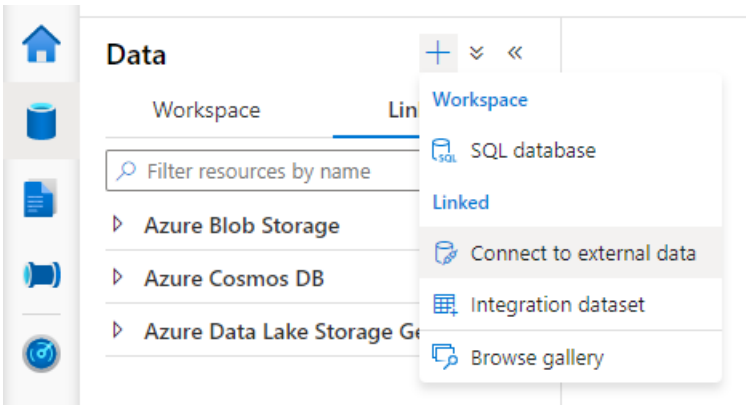
Other ADLS Gen 2 account

If the underlying data of your Hive catalogs is stored in another ADLS Gen 2 account, you need to make sure the users who run Spark queries have **Storage Blob Data Contributor** role on the ADLS Gen2 storage account.

Set up connection to Blob Storage

If the underlying data of your Hive tables are stored in Azure Blob storage account, set up the connection follow below steps:

1. Open Synapse Studio, go to **Data** > **Linked tab** > **Add** button > **Connect to external data**.



2. Choose **Azure Blob Storage** and click **Continue**.
3. Provide **Name** of the linked service. Record the name of the linked service, this info will be used in Spark configuration shortly.

4. Select the Azure Blob Storage account. Make sure Authentication method is **Account key**. Currently Spark pool can only access Blob Storage account via account key.
5. **Test connection** and click **Create**.
6. After creating the linked service to Blob Storage account, when you run Spark queries, make sure you run below Spark code in the notebook to get access to the the Blob Storage account for the Spark session. Learn more about why you need to do this here.

```
%%pyspark
blob_account_name = "<your blob storage account name>"
blob_container_name = "<your container name>"
from pyspark.sql import SparkSession
sc = SparkSession.builder.getOrCreate()
token_library = sc._jvm.com.microsoft.azure.synapse.tokenlibrary.TokenLibrary
blob_sas_token = token_library.getConnectionString("<blob storage linked service name>")
spark.conf.set('fs.azure.sas.%s.%s.blob.core.windows.net' % (blob_container_name, blob_account_name),
blob_sas_token)
```

After setting up storage connections, you can query the existing tables in the Hive Metastore.

Known limitations

- Synapse Studio object explorer will continue to show objects in managed Synapse metastore instead of the external HMS.
- SQL <-> Spark synchronization doesn't work when using external HMS.
- Only Azure SQL Database and Azure Database for MySQL are supported as external Hive Metastore database. Only SQL authorization is supported.
- Currently Spark only works on external Hive tables and non-transactional/non-ACID managed Hive tables. It doesn't support Hive ACID/transactional tables.
- Apache Ranger integration is not supported.

Troubleshooting

See below error when querying a Hive table with data stored in Blob Storage

```
No credentials found for account xxxxx.blob.core.windows.net in the configuration, and its container xxxxx
is not accessible using anonymous credentials. Please check if the container exists first. If it is not
publicly available, you have to provide account credentials.
```

When use key authentication to your storage account via linked service, you need to take an extra step to get the token for Spark session. Run below code to configure your Spark session before running the query. Learn more about why you need to do this here.

```
%%pyspark
blob_account_name = "<your blob storage account name>"
blob_container_name = "<your container name>"
from pyspark.sql import SparkSession
sc = SparkSession.builder.getOrCreate()
token_library = sc._jvm.com.microsoft.azure.synapse.tokenlibrary.TokenLibrary
blob_sas_token = token_library.getConnectionString("<blob storage linked service name>")
spark.conf.set('fs.azure.sas.%s.%s.blob.core.windows.net' % (blob_container_name, blob_account_name),
blob_sas_token)
```

See below error when query a table stored in ADLS Gen2 account

```
Operation failed: "This request is not authorized to perform this operation using this permission.", 403,
HEAD
```

This could happen because the user who runs Spark query doesn't have enough access to the underlying storage account. Make sure the user who runs Spark queries has **Storage Blob Data Contributor** role on the ADLS Gen2 storage account. This step can be done after creating the linked service.

HMS schema related settings

To avoid changing HMS backend schema/version, following hive configs are set by system by default:

```
spark.hadoop.hive.metastore.schema.verification true
spark.hadoop.hive.metastore.schema.verification.record.version false
spark.hadoop.datanucleus.fixedDatastore true
spark.hadoop.datanucleus.schema.autoCreateAll false
```

If your HMS version is `1.2.1` or `1.2.2`, there's an issue in Hive that claims requiring only `1.2.0` if you turn `spark.hadoop.hive.metastore.schema.verification` to `true`. Our suggestion is either you can modify your HMS version to `1.2.0`, or overwrite below two configurations to work around:

```
spark.hadoop.hive.metastore.schema.verification false
spark.hadoop.hive.synapse.externalmetastore.schema.usedefault false
```

If you need to migrate your HMS version, we recommend using hive schema tool. And if the HMS has been used by HDInsight clusters, we suggest using HDI provided version.

HMS schema change for OSS HMS 3.1

Synapse aims to work smoothly with computes from HDI. However HMS 3.1 in HDI 4.0 is not fully compatible with the OSS HMS 3.1. So please apply the following manually to your HMS 3.1 if it's not provisioned by HDI.

```
-- HIVE-19416
ALTER TABLE TBLS ADD WRITE_ID bigint NOT NULL DEFAULT(0);
ALTER TABLE PARTITIONS ADD WRITE_ID bigint NOT NULL DEFAULT(0);
```

When sharing the metastore with HDInsight 4.0 Spark cluster, I cannot see the tables

If you want to share the Hive catalog with a spark cluster in HDInsight 4.0, please ensure your property `spark.hadoop.metastore.catalog.default` in Synapse spark aligns with the value in HDInsight spark. The default value for HDI spark is `spark` and the default value for Synapse spark is `hive`.

When sharing the Hive Metastore with HDInsight 4.0 Hive cluster, I can list the tables successfully, but only get empty result when I query the table

As mentioned in the limitations, Synapse Spark pool only supports external hive tables and non-transactional/ACID managed tables, it doesn't support Hive ACID/transactional tables currently. In HDInsight 4.0 Hive clusters, all managed tables are created as ACID/transactional tables by default, that's why you get empty results when querying those tables.

What is Azure Synapse Link for Azure Cosmos DB?

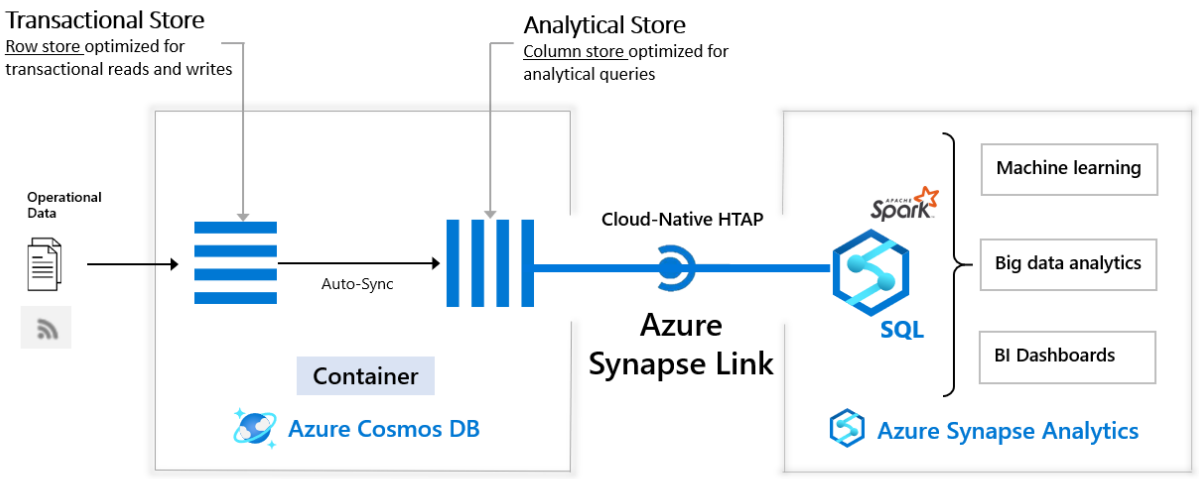
8/19/2022 • 9 minutes to read • Edit Online

APPLIES TO: SQL API Azure Cosmos DB API for MongoDB

Azure Synapse Link for Azure Cosmos DB is a cloud-native hybrid transactional and analytical processing (HTAP) capability that enables near real time analytics over operational data in Azure Cosmos DB. Azure Synapse Link creates a tight seamless integration between Azure Cosmos DB and Azure Synapse Analytics.

Using Azure Cosmos DB analytical store, a fully isolated column store, Azure Synapse Link enables no Extract-Transform-Load (ETL) analytics in Azure Synapse Analytics against your operational data at scale. Business analysts, data engineers, and data scientists can now use Synapse Spark or Synapse SQL interchangeably to run near real time business intelligence, analytics, and machine learning pipelines. You can achieve this without impacting the performance of your transactional workloads on Azure Cosmos DB.

The following image shows the Azure Synapse Link integration with Azure Cosmos DB and Azure Synapse Analytics:



Benefits

To analyze large operational datasets while minimizing the impact on the performance of mission-critical transactional workloads, traditionally, the operational data in Azure Cosmos DB is extracted and processed by Extract-Transform-Load (ETL) pipelines. ETL pipelines require many layers of data movement resulting in much operational complexity, and performance impact on your transactional workloads. It also increases the latency to analyze the operational data from the time of origin.

When compared to the traditional ETL-based solutions, Azure Synapse Link for Azure Cosmos DB offers several advantages such as:

Reduced complexity with No ETL jobs to manage

Azure Synapse Link allows you to directly access Azure Cosmos DB analytical store using Azure Synapse Analytics without complex data movement. Any updates made to the operational data are visible in the analytical store in near real-time with no ETL or change feed jobs. You can run large-scale analytics against analytical store, from Azure Synapse Analytics, without additional data transformation.

Near real-time insights into your operational data

You can now get rich insights on your operational data in near real-time, using Azure Synapse Link. ETL-based systems tend to have higher latency for analyzing your operational data, due to many layers needed to extract, transform, and load the operational data. With native integration of Azure Cosmos DB analytical store with Azure Synapse Analytics, you can analyze operational data in near real-time enabling new business scenarios.

No impact on operational workloads

With Azure Synapse Link, you can run analytical queries against an Azure Cosmos DB analytical store, a column store representation of you data, while the transactional operations are processed using provisioned throughput for the transactional workload, over the Cosmos DB row-based transactional store. The analytical workload is independent of the transactional workload traffic, not consuming any of the provisioned throughput of your operational data.

Optimized for large-scale analytics workloads

Azure Cosmos DB analytical store is optimized to provide scalability, elasticity, and performance for analytical workloads without any dependency on the compute run-times. The storage technology is self-managed to optimize your analytics workloads. With built-in support into Azure Synapse Analytics, accessing this storage layer provides simplicity and high performance.

Cost-effective

With Azure Synapse Link, you can get a cost-optimized, fully managed solution for operational analytics. It eliminates extra storage and compute layers required in traditional ETL pipelines for analyzing operational data.

Azure Cosmos DB analytical store follows a consumption-based pricing model, which is based on data storage and analytical read/write operations and queries executed. It doesn't require you to provision any throughput, as you do today for the transactional workloads. Accessing your data with highly elastic compute engines from Azure Synapse Analytics makes the overall cost of running storage and compute very efficient.

Analytics for locally available, globally distributed, multi-region writes

You can run analytical queries effectively against the nearest regional copy of the data in Azure Cosmos DB. Azure Cosmos DB provides the state-of-the-art capability to run the globally distributed analytical workloads along with transactional workloads in an active-active manner.

Enable HTAP scenarios for your operational data

Synapse Link brings together Azure Cosmos DB analytical store with Azure Synapse Analytics runtime support. This integration enables you to build cloud native HTAP (Hybrid transactional/analytical processing) solutions that generate insights based on real-time updates to your operational data over large datasets. It unlocks new business scenarios to raise alerts based on live trends, build near real-time dashboards, and business experiences based on user behavior.

Azure Cosmos DB analytical store

Azure Cosmos DB analytical store is a column-oriented representation of your operational data in Azure Cosmos DB. This analytical store is suitable for fast, cost-effective queries on large operational data sets, without copying data and impacting the performance of your transactional workloads.

Analytical store automatically picks up high frequency inserts, updates, deletes in your transactional workloads in near real-time, as a fully managed capability ("auto-sync") of Azure Cosmos DB. No change feed or ETL is required.

If you have a globally distributed Azure Cosmos DB account, after you enable analytical store for a container, it will be available in all regions for that account. For more information on the analytical store, see Azure Cosmos DB Analytical store overview article.

Integration with Azure Synapse Analytics

With Synapse Link, you can now directly connect to your Azure Cosmos DB containers from Azure Synapse

Analytics and access the analytical store with no separate connectors. Azure Synapse Analytics currently supports Synapse Link with Synapse Apache Spark and serverless SQL pool.

You can query the data from Azure Cosmos DB analytical store simultaneously, with interop across different analytics run times supported by Azure Synapse Analytics. No additional data transformations are required to analyze the operational data. You can query and analyze the analytical store data using:

- Synapse Apache Spark with full support for Scala, Python, SparkSQL, and C#. Synapse Spark is central to data engineering and data science scenarios
- Serverless SQL pool with T-SQL language and support for familiar BI tools (for example, Power BI Premium, etc.)

> **NOTE**
> From Azure Synapse Analytics, you can access both analytical and transactional stores in your Azure Cosmos DB container. However, if you want to run large-scale analytics or scans on your operational data, we recommend that you use analytical store to avoid performance impact on transactional workloads.

> **NOTE**
> You can run analytics with low latency in an Azure region by connecting your Azure Cosmos DB container to Synapse runtime in that region.

This integration enables the following HTAP scenarios for different users:

- A BI Engineer, who wants to model and publish a Power BI report to access the live operational data in Azure Cosmos DB directly through Synapse SQL.
- A Data Analyst, who wants to derive insights from the operational data in an Azure Cosmos DB container by querying it with Synapse SQL, read the data at scale and combine those findings with other data sources.
- A Data Scientist, who wants to use Synapse Spark to find a feature to improve their model and train that model without doing complex data engineering. They can also write the results of the model post inference into Azure Cosmos DB for real-time scoring on the data through Spark Synapse.
- A Data Engineer, who wants to make data accessible for consumers, by creating SQL or Spark tables over Azure Cosmos DB containers, without manual ETL processes.

For more information on Azure Synapse Analytics runtime support for Azure Cosmos DB, see Azure Synapse Analytics for Cosmos DB support.

When to use Azure Synapse Link for Azure Cosmos DB?

Synapse Link is recommended in the following cases:

- If you're an Azure Cosmos DB customer and you want to run analytics, BI, and machine learning over your operational data. In such cases, Synapse Link provides a more integrated analytics experience without impacting your transactional store's provisioned throughput. For example:
  - If you're running analytics or BI on your Azure Cosmos DB operational data directly using separate connectors today, or
  - If you're running ETL processes to extract operational data into a separate analytics system.

In such cases, Synapse Link provides a more integrated analytics experience without impacting your

transactional store's provisioned throughput.

Synapse Link isn't recommended if you're looking for traditional data warehouse requirements such as high concurrency, workload management, and persistence of aggregates across multiple data sources. For more information, see common scenarios that can be powered with Azure Synapse Link for Azure Cosmos DB.

Limitations

- Azure Synapse Link for Azure Cosmos DB is not supported for Gremlin API, Cassandra API, and Table API. It is supported for SQL API and API for Mongo DB.
- Accessing the Azure Cosmos DB analytics store with Azure Synapse Dedicated SQL Pool currently isn't supported.
- Enabling Synapse Link on existing Cosmos DB containers is only supported for SQL API accounts. Synapse Link can be enabled on new containers for both SQL API and MongoDB API accounts.
- Although analytical store data is not backed up, and therefore cannot be restored, you can rebuild your analytical store by reenabling Synapse Link in the restored container. Click here for more information.
- Currently Synapse Link isn't fully compatible with continuous backup mode. Click here for more information.
- Granular Role-based Access (RBAC)s isn't supported when querying from Synapse. Users that have access to your Synapse workspace and have access to the Cosmos DB account can access all containers within that account. We currently don't support more granular access to the containers.
- Currently Azure Synapse Workspaces don't support linked services using `Managed Identity`. Always use the `MasterKey` option.

Security

Synapse Link enables you to run near real-time analytics over your mission-critical data in Azure Cosmos DB. It is vital to make sure that critical business data is stored securely across both transactional and analytical stores. Azure Synapse Link for Azure Cosmos DB is designed to help meet these security requirements through the following features:

- **Network isolation using private endpoints** - You can control network access to the data in the transactional and analytical stores independently. Network isolation is done using separate managed private endpoints for each store, within managed virtual networks in Azure Synapse workspaces. To learn more, see how to Configure private endpoints for analytical store article.
- **Data encryption with customer-managed keys** - You can seamlessly encrypt the data across transactional and analytical stores using the same customer-managed keys in an automatic and transparent manner. Azure Synapse Link only supports configuring customer-managed keys using your Azure Cosmos DB account's managed identity. You must configure your account's managed identity in your Azure Key Vault access policy before enabling Azure Synapse Link on your account. To learn more, see how to Configure customer-managed keys using Azure Cosmos DB accounts' managed identities article.
- **Secure key management** - Accessing the data in analytical store from Synapse Spark and Synapse serverless SQL pools requires managing Azure Cosmos DB keys within Synapse Analytics workspaces. Instead of using the Azure Cosmos DB account keys inline in Spark jobs or SQL scripts, Azure Synapse Link provides more secure capabilities:
  - When using Synapse serverless SQL pools, you can query the Azure Cosmos DB analytical store by pre-creating SQL credentials storing the account keys and referencing these in the `OPENROWSET`

function. To learn more, see Query with a serverless SQL pool in Azure Synapse Link article.

- When using Synapse Spark, you can store the account keys in linked service objects pointing to an Azure Cosmos DB database and reference this in the Spark configuration at runtime. To learn more, see Copy data into a dedicated SQL pool using Apache Spark article.

Pricing

The billing model of Azure Synapse Link includes the costs incurred by using the Azure Cosmos DB analytical store and the Synapse runtime. To learn more, see the Azure Cosmos DB analytical store pricing and Azure Synapse Analytics pricing articles.

Next steps

To learn more, see the following docs:

- Azure Cosmos DB analytical store overview
- Check out the learn module on how to Design hybrid transactional and analytical processing using Azure Synapse Analytics
- Get started with Azure Synapse Link for Azure Cosmos DB
- What is supported in Azure Synapse Analytics run time
- Frequently asked questions about Azure Synapse Link for Azure Cosmos DB
- Azure Synapse Link for Azure Cosmos DB Use cases

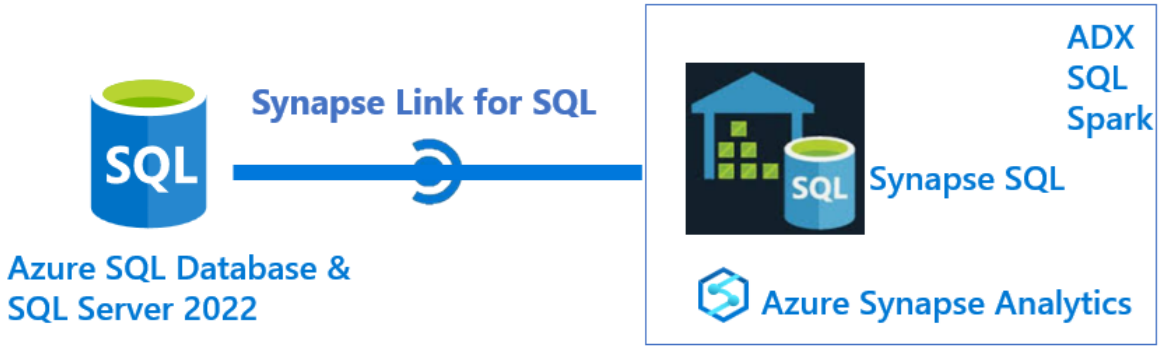
What is Azure Synapse Link for SQL? (Preview)

8/19/2022 • 2 minutes to read • Edit Online

Azure Synapse Link for SQL enables near real time analytics over operational data in Azure SQL Database or SQL Server 2022. With a seamless integration between operational stores including Azure SQL Database and SQL Server 2022 and Azure Synapse Analytics, Azure Synapse Link for SQL enables you to run analytics, business intelligence and machine learning scenarios on your operational data with minimum impact on source databases with a new change feed technology.

> **IMPORTANT**
>
> Azure Synapse Link for Azure SQL is currently in PREVIEW. See the Supplemental Terms of Use for Microsoft Azure Previews for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

The following image shows the Azure Synapse Link integration with Azure SQL DB, SQL Server 2022, and Azure Synapse Analytics:



Benefit

Azure Synapse Link for SQL provides fully managed and turnkey experience for you to land operational data in Azure Synapse Analytics dedicated SQL pools. It does this by continuously replicating the data from Azure SQL Database or SQL Server 2022 with full consistency. By using Azure Synapse Link for SQL, you can get the following benefits:

- **Minimum impact on operational workload** With the new change feed technology in Azure SQL Database and SQL Server 2022, Azure Synapse Link for SQL can automatically extract incremental changes from Azure SQL Database or SQL Server 2022. It then replicates to Azure Synapse Analytics dedicated SQL pool with minimal impact on the operational workload.
- **Reduced complexity with No ETL jobs to manage** After going through a few clicks including selecting your operational database and tables, updates made to the operational data in Azure SQL Database or SQL Server 2022 are visible in the Azure Synapse Analytics dedicated SQL pool. They're available in near real-time with no ETL or data integration logic. You can focus on analytical and reporting logic against operational data via all the capabilities within Azure Synapse Analytics.
- **Near real-time insights into your operational data** You can now get rich insights by analyzing operational data in Azure SQL Database or SQL Server 2022 in near real-time to enable new business scenarios including operational BI reporting, real time scoring and personalization, or supply chain

forecasting etc. via Azure Synapse link for SQL.

Next steps

- Azure Synapse Link for Azure SQL Database (Preview).
- Azure Synapse Link for SQL Server 2022 (Preview).
- How to Configure Azure Synapse Link for SQL Server 2022 (Preview).
- How to Configure Azure Synapse Link for Azure SQL Database (Preview).

Quickstart: Connect to Azure Synapse Link for Azure Cosmos DB

8/19/2022 • 2 minutes to read • Edit Online

This article describes how to access an Azure Cosmos DB database from Azure Synapse Analytics studio with Synapse Link.

Prerequisites

Before you connect an Azure Cosmos DB account to your workspace, there are a few things that you need.

- Existing Azure Cosmos DB account or create a new account following this quickstart
- Existing Synapse workspace or create a new workspace following this quickstart

Enable Azure Cosmos DB analytical store

To run large-scale analytics into Azure Cosmos DB without impacting your operational performance, we recommend enabling Synapse Link for Azure Cosmos DB. This function brings HTAP capability to a container and built-in support in Azure Synapse. Follow this quickstart to enable Synapse Link for Cosmos DB containers.

Navigate to Synapse Studio

From your Synapse workspace, select **Launch Synapse Studio**. On the Synapse Studio home page, select **Data, which will take you to the **Data Object Explorer**.

Connect an Azure Cosmos DB database to a Synapse workspace

Connecting an Azure Cosmos DB database is done as linked service. A Cosmos DB linked service enables users to browse and explore data, read, and write from Apache Spark for Azure Synapse Analytics or SQL into Azure Cosmos DB.

From the Data Object Explorer, you can directly connect an Azure Cosmos DB database by doing the following steps:

1. Select **+** icon near Data
2. Select **Connect to external data**
3. Select the API that you want to connect to: SQL or MongoDB
4. Select ***Continue***
5. Name the linked service. The name will be displayed in the Object Explorer and used by Synapse run-times to connect to the database and containers. We recommend using a friendly name.
6. Select the **Cosmos DB account name** and **database name**
7. (Optional) If no region is specified, Synapse run-time operations will be routed toward the nearest region where the analytical store is enabled. However, you can set manually which region you want your users to access Cosmos DB analytical store. Select **Additional connection properties** and then **New**. Under **Property Name**, write ***PreferredRegions*** and set the **Value** to the region you want (example: WestUS2, there is no space between words and numbers)
8. Select ***Create***

Azure Cosmos DB databases are visible under the tab **Linked** in the Azure Cosmos DB section. You can

differentiate an HTAP enabled Azure Cosmos DB container from an OLTP only container with the following icons:

Synapse container:

OLTP only container:

Quickly interact with code-generated actions

When you right-click into a container, you'll have a list of gestures that will trigger a Spark or SQL run-time. Writing into a container will happen through the Transactional Store of Azure Cosmos DB and will consume Request Units.

Next steps

- Learn what is supported between Synapse and Azure Cosmos DB
- Learn how to query an analytical store with Apache Spark 3 for Azure Synapse Analytics
- Learn how to query an analytical store with Apache Spark 2 for Azure Synapse Analytics

What is Azure Cosmos DB analytical store?

8/19/2022 • 28 minutes to read • Edit Online

APPLIES TO: SQL API Azure Cosmos DB API for MongoDB

Azure Cosmos DB analytical store is a fully isolated column store for enabling large-scale analytics against operational data in your Azure Cosmos DB, without any impact to your transactional workloads.

Azure Cosmos DB transactional store is schema-agnostic, and it allows you to iterate on your transactional applications without having to deal with schema or index management. In contrast to this, Azure Cosmos DB analytical store is schematized to optimize for analytical query performance. This article describes in detailed about analytical storage.

Challenges with large-scale analytics on operational data

The multi-model operational data in an Azure Cosmos DB container is internally stored in an indexed row-based "transactional store". Row store format is designed to allow fast transactional reads and writes in the order-of-milliseconds response times, and operational queries. If your dataset grows large, complex analytical queries can be expensive in terms of provisioned throughput on the data stored in this format. High consumption of provisioned throughput in turn, impacts the performance of transactional workloads that are used by your real-time applications and services.

Traditionally, to analyze large amounts of data, operational data is extracted from Azure Cosmos DB's transactional store and stored in a separate data layer. For example, the data is stored in a data warehouse or data lake in a suitable format. This data is later used for large-scale analytics and analyzed using compute engine such as the Apache Spark clusters. This separation of analytical storage and compute layers from operational data results in additional latency, because the ETL(Extract, Transform, Load) pipelines are run less frequently to minimize the potential impact on your transactional workloads.

The ETL pipelines also become complex when handling updates to the operational data when compared to handling only newly ingested operational data.

Column-oriented analytical store

Azure Cosmos DB analytical store addresses the complexity and latency challenges that occur with the traditional ETL pipelines. Azure Cosmos DB analytical store can automatically sync your operational data into a separate column store. Column store format is suitable for large-scale analytical queries to be performed in an optimized manner, resulting in improving the latency of such queries.

Using Azure Synapse Link, you can now build no-ETL HTAP solutions by directly linking to Azure Cosmos DB analytical store from Azure Synapse Analytics. It enables you to run near real-time large-scale analytics on your operational data.

Features of analytical store

When you enable analytical store on an Azure Cosmos DB container, a new column-store is internally created based on the operational data in your container. This column store is persisted separately from the row-oriented transactional store for that container. The inserts, updates, and deletes to your operational data are automatically synced to analytical store. You don't need the Change Feed or ETL to sync the data.

Column store for analytical workloads on operational data

Analytical workloads typically involve aggregations and sequential scans of selected fields. By storing the data in a column-major order, the analytical store allows a group of values for each field to be serialized together. This format reduces the IOPS required to scan or compute statistics over specific fields. It dramatically improves the query response times for scans over large data sets.

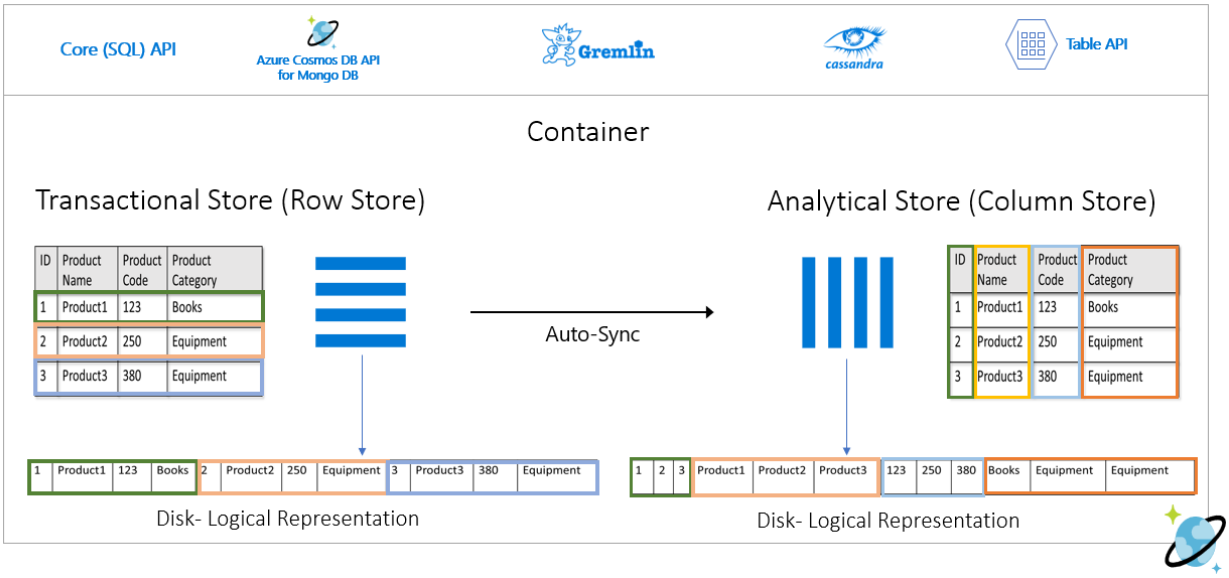
For example, if your operational tables are in the following format:

ID	Product Name	Product Code	Product Category
1	Product1	123	Books
2	Product2	250	Equipment
3	Product3	380	Equipment

The row store persists the above data in a serialized format, per row, on the disk. This format allows for faster transactional reads, writes, and operational queries, such as, "Return information about Product1". However, as the dataset grows large and if you want to run complex analytical queries on the data it can be expensive. For example, if you want to get "the sales trends for a product under the category named 'Equipment' across different business units and months", you need to run a complex query. Large scans on this dataset can get expensive in terms of provisioned throughput and can also impact the performance of the transactional workloads powering your real-time applications and services.

Analytical store, which is a column store, is better suited for such queries because it serializes similar fields of data together and reduces the disk IOPS.

The following image shows transactional row store vs. analytical column store in Azure Cosmos DB:



Decoupled performance for analytical workloads

There's no impact on the performance of your transactional workloads due to analytical queries, as the analytical store is separate from the transactional store. Analytical store doesn't need separate request units (RUs) to be allocated.

Auto-Sync

Auto-Sync refers to the fully managed capability of Azure Cosmos DB where the inserts, updates, deletes to operational data are automatically synced from transactional store to analytical store in near real time. Auto-

sync latency is usually within 2 minutes. In cases of shared throughput database with a large number of containers, auto-sync latency of individual containers could be higher and take up to 5 minutes. We would like to learn more how this latency fits your scenarios. For that, please reach out to the Azure Cosmos DB Team.

At the end of each execution of the automatic sync process, your transactional data will be immediately available for Azure Synapse Analytics runtimes:

- Azure Synapse Analytics Spark pools can read all data, including the most recent updates, through Spark tables, which are updated automatically, or via the `spark.read` command, that always reads the last state of the data.
- Azure Synapse Analytics SQL Serverless pools can read all data, including the most recent updates, through views, which are updated automatically, or via `SELECT` together with the `OPENROWSET` commands, which always reads the latest status of the data.

> **NOTE**
> Your transactional data will be synchronized to analytical store even if your transactional time-to-live (TTL) is smaller than 2 minutes.

> **NOTE**
> Please note that if you delete your container, analytical store is also deleted.

Scalability & elasticity

By using horizontal partitioning, Azure Cosmos DB transactional store can elastically scale the storage and throughput without any downtime. Horizontal partitioning in the transactional store provides scalability & elasticity in auto-sync to ensure data is synced to the analytical store in near real time. The data sync happens regardless of the transactional traffic throughput, whether it's 1000 operations/sec or 1 million operations/sec, and it doesn't impact the provisioned throughput in the transactional store.

Automatically handle schema updates

Azure Cosmos DB transactional store is schema-agnostic, and it allows you to iterate on your transactional applications without having to deal with schema or index management. In contrast to this, Azure Cosmos DB analytical store is schematized to optimize for analytical query performance. With the auto-sync capability, Azure Cosmos DB manages the schema inference over the latest updates from the transactional store. It also manages the schema representation in the analytical store out-of-the-box which, includes handling nested data types.

As your schema evolves, and new properties are added over time, the analytical store automatically presents a unionized schema across all historical schemas in the transactional store.

> **NOTE**
> In the context of analytical store, we consider the following structures as property:
>
> - JSON "elements" or "string-value pairs separated by a `:` ".
> - JSON objects, delimited by `{` and `}`.
> - JSON arrays, delimited by `[` and `]`.

Schema constraints

The following constraints are applicable on the operational data in Azure Cosmos DB when you enable analytical store to automatically infer and represent the schema correctly:

- You can have a maximum of 1000 properties across all nested levels in the document schema and a maximum nesting depth of 127.
  - Only the first 1000 properties are represented in the analytical store.
  - Only the first 127 nested levels are represented in the analytical store.
  - The first level of a JSON document is its `/` root level.
  - Properties in the first level of the document will be represented as columns.
- Sample scenarios:
  - If your document's first level has 2000 properties, only the first 1000 will be represented.
  - If your documents have five levels with 200 properties in each one, all properties will be represented.
  - If your documents have 10 levels with 400 properties in each one, only the two first levels will be fully represented in analytical store. Half of the third level will also be represented.
- The hypothetical document below contains four properties and three levels.
  - The levels are `root`, `myArray`, and the nested structure within the `myArray`.
  - The properties are `id`, `myArray`, `myArray.nested1`, and `myArray.nested2`.
  - The analytical store representation will have two columns, `id`, and `myArray`. You can use Spark or T-SQL functions to also expose the nested structures as columns.

```
{
  "id": "1",
  "myArray": [
    "string1",
    "string2",
    {
      "nested1": "abc",
      "nested2": "cde"
    }
  ]
}
```

- While JSON documents (and Cosmos DB collections/containers) are case-sensitive from the uniqueness perspective, analytical store isn't.
  - **In the same document:** Properties names in the same level should be unique when compared case-insensitively. For example, the following JSON document has "Name" and "name" in the same level. While it's a valid JSON document, it doesn't satisfy the uniqueness constraint and hence won't be fully represented in the analytical store. In this example, "Name" and "name" are the same when compared in a case-insensitive manner. Only `"Name": "fred"` will be represented in analytical store, because it's the first occurrence. And `"name": "john"` won't be represented at all.

    ```
    {"id": 1, "Name": "fred", "name": "john"}
    ```

  - **In different documents:** Properties in the same level and with the same name, but in different cases, will be represented within the same column, using the name format of the first occurrence. For example, the following JSON documents have `"Name"` and `"name"` in the same level. Since the first document format is `"Name"`, this is what will be used to represent the property name in analytical store. In other words, the column name in analytical store will be `"Name"`. Both `"fred"` and `"john"` will be represented, in the `"Name"` column.

```
{"id": 1, "Name": "fred"}
{"id": 2, "name": "john"}
```

- The first document of the collection defines the initial analytical store schema.
  - Documents with more properties than the initial schema will generate new columns in analytical store.
  - Columns can't be removed.
  - The deletion of all documents in a collection doesn't reset the analytical store schema.
  - There's not schema versioning. The last version inferred from transactional store is what you'll see in analytical store.
- Currently Azure Synapse Spark can't read properties that contain some special characters in their names, listed below. Azure Synapse SQL serverless isn't affected.
  - :
  - `
  - ,
  - ;
  - {}
  - ()
  - \n
  - \t
  - =
  - "

> **NOTE**
> White spaces are also listed in the Spark error message returned when you reach this limitation. But we have added a special treatment for white spaces, please check out more details in the items below.

- If you have properties names using the characters listed above, the alternatives are:
  - Change your data model in advance to avoid these characters.
  - Since currently we don't support schema reset, you can change your application to add a redundant property with a similar name, avoiding these characters.
  - Use Change Feed to create a materialized view of your container without these characters in properties names.
  - Use the `dropColumn` Spark option to ignore the affected columns and load all other columns into a DataFrame. The syntax is:

```
# Removing one column:
df = spark.read\
     .format("cosmos.olap")\
     .option("spark.synapse.linkedService","<your-linked-service-name>")\
     .option("spark.synapse.container","<your-container-name>")\
     .option("spark.synapse.dropColumn","FirstName,LastName")\
     .load()

# Removing multiple columns:
df = spark.read\
     .format("cosmos.olap")\
     .option("spark.synapse.linkedService","<your-linked-service-name>")\
     .option("spark.synapse.container","<your-container-name>")\
     .option("spark.synapse.dropColumn","FirstName,LastName;StreetName,StreetNumber")\
     .option("spark.cosmos.dropMultiColumnSeparator", ";")\
     .load()
```

- Azure Synapse Spark now supports properties with white spaces in their names. For that, you need to use the `allowWhiteSpaceInFieldNames` Spark option to load the affected columns into a DataFrame, keeping the original name. The syntax is:

```
df = spark.read\
     .format("cosmos.olap")\
     .option("spark.synapse.linkedService","<your-linked-service-name>")\
     .option("spark.synapse.container","<your-container-name>")\
     .option("spark.cosmos.allowWhiteSpaceInFieldNames", "true")\
    .load()
```

- The following BSON datatypes aren't supported and won't be represented in analytical store:
  - Decimal128
  - Regular Expression
  - DB Pointer
  - JavaScript
  - Symbol
  - MinKey/MaxKey
- When using DateTime strings that follow the ISO 8601 UTC standard, expect the following behavior:
  - Spark pools in Azure Synapse will represent these columns as `string`.
  - SQL serverless pools in Azure Synapse will represent these columns as `varchar(8000)`.
- Properties with `UNIQUEIDENTIFIER (guid)` types are represented as `string` in analytical store and should be converted to `VARCHAR` in **SQL** or to `string` in **Spark** for correct visualization.
- SQL serverless pools in Azure Synapse support result sets with up to 1000 columns, and exposing nested columns also counts towards that limit. Please consider this information when designing your data architecture and modeling your transactional data.
- If you rename a property, in one or many documents, it will be considered a new column. If you execute the same rename in all documents in the collection, all data will be migrated to the new column and the old column will be represented with `NULL` values.

Schema representation

There are two types of schema representation in the analytical store. These types define the schema representation method for all containers in the database account and have tradeoffs between the simplicity of query experience versus the convenience of a more inclusive columnar representation for polymorphic schemas.

- Well-defined schema representation, default option for SQL (CORE) API accounts.
- Full fidelity schema representation, default option for Azure Cosmos DB API for MongoDB accounts.

Well-defined schema representation

The well-defined schema representation creates a simple tabular representation of the schema-agnostic data in the transactional store. The well-defined schema representation has the following considerations:

- The first document defines the base schema and property must always have the same type across all documents. The only exceptions are:
  - From `NULL` to any other data type. The first non-null occurrence defines the column data type. Any document not following the first non-null datatype won't be represented in analytical store.
  - From `float` to `integer`. All documents will be represented in analytical store.
  - From `integer` to `float`. All documents will be represented in analytical store. However, to read this data with Azure Synapse SQL serverless pools, you must use a WITH clause to convert the column to `varchar`. And after this initial conversion, it's possible to convert it again to a number. Please check the example below, where **num** initial value was an integer and the second one was a float.

```
SELECT CAST (num as float) as num
FROM OPENROWSET(PROVIDER = 'CosmosDB',
                CONNECTION = '<your-connection',
                OBJECT = 'IntToFloat',
                SERVER_CREDENTIAL = 'your-credential'
)
WITH (num varchar(100)) AS [IntToFloat]
```

- Properties that don't follow the base schema data type won't be represented in analytical store. For example, consider the documents below: the first one defined the analytical store base schema. The second document, where `id` is `"2"`, **doesn't** have a well-defined schema since property `"code"` is a string and the first document has `"code"` as a number. In this case, the analytical store registers the data type of `"code"` as `integer` for lifetime of the container. The second document will still be included in analytical store, but its `"code"` property won't.
  - `{"id": "1", "code":123}`
  - `{"id": "2", "code": "123"}`

> **NOTE**
> The condition above doesn't apply for `NULL` properties. For example, `{"a":123} and {"a":NULL}` is still well-defined.

> **NOTE**
> The condition above doesn't change if you update `"code"` of document `"1"` to a string in your transactional store. In analytical store, `"code"` will be kept as `integer` since currently we don't support schema reset.

- Array types must contain a single repeated type. For example, `{"a": ["str",12]}` isn't a well-defined schema because the array contains a mix of integer and string types.

> **NOTE**
> If the Azure Cosmos DB analytical store follows the well-defined schema representation and the specification above is violated by certain items, those items won't be included in the analytical store.

- Expect different behavior in regard to different types in well-defined schema:
  - Spark pools in Azure Synapse will represent these values as `undefined`.
  - SQL serverless pools in Azure Synapse will represent these values as `NULL`.
- Expect different behavior in regard to explicit `NULL` values:
  - Spark pools in Azure Synapse will read these values as `0` (zero). And it will change to `undefined` as soon as the column has a non-null value.
  - SQL serverless pools in Azure Synapse will read these values as `NULL`.
- Expect different behavior in regard to missing columns:
  - Spark pools in Azure Synapse will represent these columns as `undefined`.
  - SQL serverless pools in Azure Synapse will represent these columns as `NULL`.

Full fidelity schema representation

The full fidelity schema representation is designed to handle the full breadth of polymorphic schemas in the schema-agnostic operational data. In this schema representation, no items are dropped from the analytical store even if the well-defined schema constraints (that is no mixed data type fields nor mixed data type arrays) are violated.

This is achieved by translating the leaf properties of the operational data into the analytical store with distinct columns based on the data type of values in the property. The leaf property names are extended with data types as a suffix in the analytical store schema such that they can be queries without ambiguity.

In the full fidelity schema representation, each datatype of each property will generate a column for that datatype. Each of them count as one of the 1000 maximum properties.

For example, let's take the following sample document in the transactional store:

```
{
name: "John Doe",
age: 32,
profession: "Doctor",
address: {
  streetNo: 15850,
  streetName: "NE 40th St.",
  zip: 98052
},
salary: 1000000
}
```

The leaf property `streetNo` within the nested object `address` will be represented in the analytical store schema as a column `address.object.streetNo.int32`. The datatype is added as a suffix to the column. This way, if another document is added to the transactional store where the value of leaf property `streetNo` is "123" (note it's a string), the schema of the analytical store automatically evolves without altering the type of a previously written column. A new column added to the analytical store as `address.object.streetNo.string` where this value of "123" is stored.

Data type to suffix map for full fidelity schema

Here's a map of all the property data types and their suffix representations in the analytical store in full fidelity schema representation:

ORIGINAL DATA TYPE	SUFFIX	EXAMPLE
Double	".float64"	24.99

ORIGINAL DATA TYPE	SUFFIX	EXAMPLE
Array	".array"	["a", "b"]
Binary	".binary"	0
Boolean	".bool"	True
Int32	".int32"	123
Int64	".int64"	255486129307
NULL	".NULL"	NULL
String	".string"	"ABC"
Timestamp	".timestamp"	Timestamp(0, 0)
ObjectId	".objectId"	ObjectId("5f3f7b59330ec25c132623a2")
Document	".object"	{"a": "a"}

- Expect different behavior in regard to explicit `NULL` values:
  - Spark pools in Azure Synapse will read these values as `0` (zero).
  - SQL serverless pools in Azure Synapse will read these values as `NULL`.
- Expect different behavior in regard to missing columns:
  - Spark pools in Azure Synapse will represent these columns as `undefined`.
  - SQL serverless pools in Azure Synapse will represent these columns as `NULL`.

Working with the MongoDB `_id` field

the MongoDB `_id` field is fundamental to every collection in MongoDB and originally has a hexadecimal representation. As you can see in the table above, `Full Fidelity Schema` will preserve its characteristics, creating a challenge for its vizualiation in Azure Synapse Analytics. For correct visualization, you must convert the `_id` datatype as below:

Spark

```
import org.apache.spark.sql.types._
val simpleSchema = StructType(Array(
    StructField("_id", StructType(Array(StructField("objectId",BinaryType,true)) ),true),
    StructField("id", StringType, true)
  ))

df = spark.read.format("cosmos.olap")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .schema(simpleSchema)
    .load()

df.select("id", "_id.objectId").show()
```

SQL

```
SELECT TOP 100 id=CAST(_id as VARBINARY(1000))
FROM OPENROWSET('CosmosDB',
                'Your-account;Database=your-database;Key=your-key',
                HTAP) WITH (_id VARCHAR(1000)) as HTAP
```

Full fidelity schema for SQL API accounts

It's possible to use full fidelity Schema for SQL (Core) API accounts, instead of the default option, by setting the schema type when enabling Synapse Link on a Cosmos DB account for the first time. Here are the considerations about changing the default schema representation type:

- This option is only valid for accounts that **don't** have Synapse Link already enabled.
- It isn't possible to reset the schema representation type, from well-defined to full fidelity or vice-versa.
- Currently Azure Cosmos DB API for MongoDB isn't compatible with this possibility of changing the schema representation. All MongoDB accounts will always have full fidelity schema representation type.
- Currently this change can't be made through the Azure portal. All database accounts that have Synapse Link enabled by the Azure portal will have the default schema representation type, well-defined schema.

The schema representation type decision must be made at the same time that Synapse Link is enabled on the account, using Azure CLI or PowerShell.

With the Azure CLI:

```
az cosmosdb create --name MyCosmosDBDatabaseAccount --resource-group MyResourceGroup --subscription
MySubscription --analytical-storage-schema-type "FullFidelity" --enable-analytical-storage true
```

> **NOTE**
> In the command above, replace `create` with `update` for existing accounts.

With the PowerShell:

```
 New-AzCosmosDBAccount -ResourceGroupName MyResourceGroup -Name MyCosmosDBDatabaseAccount  -
EnableAnalyticalStorage true -AnalyticalStorageSchemaType "FullFidelity"
```

> **NOTE**
> In the command above, replace `New-AzCosmosDBAccount` with `Update-AzCosmosDBAccount` for existing accounts.

Analytical Time-to-Live (TTL)

Analytical TTL (ATTL) indicates how long data should be retained in your analytical store, for a container.

Analytical store is enabled when ATTL is set with a value other than `NULL` and `0`. When enabled, inserts, updates, deletes to operational data are automatically synced from transactional store to analytical store, irrespective of the transactional TTL (TTTL) configuration. The retention of this transactional data in analytical store can be controlled at container level by the `AnalyticalStoreTimeToLiveInSeconds` property.

The possible ATTL configurations are:

- If the value is set to `0` or set to `NULL`: the analytical store is disabled and no data is replicated from transactional store to analytical store
- If the value is set to `-1`: the analytical store retains all historical data, irrespective of the retention of the

data in the transactional store. This setting indicates that the analytical store has infinite retention of your operational data

- If the value is set to any positive integer `n` number: items will expire from the analytical store `n` seconds after their last modified time in the transactional store. This setting can be leveraged if you want to retain your operational data for a limited period of time in the analytical store, irrespective of the retention of the data in the transactional store

Some points to consider:

- After the analytical store is enabled with an ATTL value, it can be updated to a different valid value later.
- While TTTL can be set at the container or item level, ATTL can only be set at the container level currently.
- You can achieve longer retention of your operational data in the analytical store by setting ATTL >= TTTL at the container level.
- The analytical store can be made to mirror the transactional store by setting ATTL = TTTL.
- If you have ATTL bigger than TTTL, at some point in time you'll have data that only exists in analytical store. This data is read only.

How to enable analytical store on a container:

- From the Azure portal, the ATTL option, when turned on, is set to the default value of -1. You can change this value to 'n' seconds, by navigating to container settings under Data Explorer.
- From the Azure Management SDK, Azure Cosmos DB SDKs, PowerShell, or Azure CLI, the ATTL option can be enabled by setting it to either -1 or 'n' seconds.

To learn more, see how to configure analytical TTL on a container.

Cost-effective analytics on historical data

Data tiering refers to the separation of data between storage infrastructures optimized for different scenarios. Thereby improving the overall performance and cost-effectiveness of the end-to-end data stack. With analytical store, Azure Cosmos DB now supports automatic tiering of data from the transactional store to analytical store with different data layouts. With analytical store optimized in terms of storage cost compared to the transactional store, allows you to retain much longer horizons of operational data for historical analysis.

After the analytical store is enabled, based on the data retention needs of the transactional workloads, you can configure `transactional TTL` property to have records automatically deleted from the transactional store after a certain time period. Similarly, the `analytical TTL` allows you to manage the lifecycle of data retained in the analytical store, independent from the transactional store. By enabling analytical store and configuring transactional and analytical `TTL` properties, you can seamlessly tier and define the data retention period for the two stores.

> **NOTE**
>
> When `analytical TTL` is bigger than `transactional TTL`, your container will have data that only exists in analytical store. This data is read only and currently we don't support document level `TTL` in analytical store. If your container data may need an update or a delete at some point in time in the future, don't use `analytical TTL` bigger than `transactional TTL`. This capability is recommended for data that won't need updates or deletes in the future.

> **NOTE**
> If your scenario doesn't demand physical deletes, you can adopt a logical delete/update approach. Insert in transactional store another version of the same document that only exists in analytical store but needs a logical delete/update. Maybe with a flag indicating that it's a delete or an update of an expired document. Both versions of the same document will co-exist in analytical store, and your application should only consider the last one.

Resilience

Analytical store relies on Azure Storage and offers the following protection against physical failure:

- Single region Azure Cosmos DB database accounts allocate analytical store in Locally Redundant Storage (LRS) Azure Storage accounts.
- If any geo-region replication is configured for the Azure Cosmos DB database account, analytical store is allocated in Zone-Redundant Storage (ZRS) Azure storage accounts.

Backup

Although analytical store has built-in protection against physical failures, backup can be necessary for accidental deletes or updates in transactional store. In those cases, you can restore a container and use the restored container to backfill the data in the original container, or fully rebuild analytical store if necessary.

> **NOTE**
> Currently analytical store isn't backuped and can't be restored, and your backup policy can't be planned relying on that.

Synapse Link, and analytical store by consequence, has different compatibility levels with Azure Cosmos DB backup modes:

- Periodic backup mode is fully compatible with Synapse Link and these 2 features can be used in the same database account.
- Continuous backup mode isn't fully supported yet:
  - Database accounts with Synapse Link enabled currently can't use continuous backup mode.
  - Database accounts with continuous backup mode enabled can enable Synapse Link through a support case. This capability is in preview now.
  - Database accounts that have neither continuous backup nor Synapse Link enabled can use these two features together through a support case. This capability is in preview now.

Backup Polices

There two possible backup polices and to understand how to use them, two details about Cosmos DB backups are very important:

- The original container is restored without analytical store in both backup modes.
- Cosmos DB doesn't support containers overwrite from a restore.

Now let's see how to use backup and restores from the analytical store perspective.

Restoring a container with TTTL >= ATTL

When `transactional TTL` is equal or bigger than `analytical TTL`, all data in analytical store still exists in transactional store. In case of a restore, you have two possible situations:

- To use the restored container as a replacement for the original container. To rebuild analytical store, just enable Synapse Link at account level and container level.
- To use the restored container as a data source to backfill or update the data in the original container. In this

case, analytical store will automatically reflect the data operations.

Restoring a container with TTTL < ATTL

When `transactional TTL` is smaller than `analytical TTL`, some data only exists in analytical store and won't be in the restored container. Again your have two possible situations:

- To use the restored container as a replacement for the original container. In this case, when you enable Synapse Link at container level, only the data that was in transactional store will be included in the new analytical store. But please note that the analytical store of the original container remains available for queries as long as the original container exists. You may want to change your application to query both.
- To use the restored container as a data source to backfill or update the data in the original container:
- Analytical store will automatically reflect the data operations for the data that is in transactional store.
- If you re-insert data that was previously removed from transactional store due to `transactional TTL`, this data will be duplicated in analytical store.

Example:

- Container `OnlineOrders` has TTTL set to one month and ATTL set for one year.
- When you restore it to `OnlineOrdersNew` and turn on analytical store to rebuild it, there will be only one month of data in both transactional and analytical store.
- Original container `OnlineOrders` isn't deleted and its analytical store is still available.
- New data is only ingested into `OnlineOrdersNew`.
- Analytical queries will do a UNION ALL from analytical stores while the original data is still relevant.

If you want to delete the original container but don't want to lose its analytical store data, you can persist the analytical store of the original container in another Azure data service. Synapse Analytics has the capability to perform joins between data stored in different locations. An example: A Synapse Analytics query joins analytical store data with external tables located in Azure Blob Storage, Azure Data Lake Store, etc.

It's important to note that the data in the analytical store has a different schema than what exists in the transactional store. While you can generate snapshots of your analytical store data, and export it to any Azure Data service, at no RUs costs, we can't guarantee the use of this snapshot to back feed the transactional store. This process isn't supported.

Global Distribution

If you have a globally distributed Azure Cosmos DB account, after you enable analytical store for a container, it will be available in all regions of that account. Any changes to operational data are globally replicated in all regions. You can run analytical queries effectively against the nearest regional copy of your data in Azure Cosmos DB.

Partitioning

Analytical store partitioning iscompletelyindependent of partitioning inthetransactional store. By default, data in analytical store isn't partitioned. If your analytical queries have frequently used filters, you have the option to partition based on these fields for better query performance. To learn more, see introduction to custom partitioning and how to configure custom partitioning.

Security

- **Authentication with the analytical store** is the same as the transactional store for a given database. You can use primary, secondary, or read-only keys for authentication. You can leverage linked service in Synapse Studio to prevent pasting the Azure Cosmos DB keys in the Spark notebooks. For Azure Synapse SQL serverless, you can use SQL credentials to also prevent pasting the Azure Cosmos DB keys in the

SQL notebooks. The Access to these Linked Services or to these SQL credentials are available to anyone who has access to the workspace. Please note that the Cosmos DB read only key can also be used.

- **Network isolation using private endpoints** - You can control network access to the data in the transactional and analytical stores independently. Network isolation is done using separate managed private endpoints for each store, within managed virtual networks in Azure Synapse workspaces. To learn more, see how to Configure private endpoints for analytical store article.
- **Data encryption at rest** - Your analytical store encryption is enabled by default.
- **Data encryption with customer-managed keys** - You can seamlessly encrypt the data across transactional and analytical stores using the same customer-managed keys in an automatic and transparent manner. Azure Synapse Link only supports configuring customer-managed keys using your Azure Cosmos DB account's managed identity. You must configure your account's managed identity in your Azure Key Vault access policy before enabling Azure Synapse Link on your account. To learn more, see how to Configure customer-managed keys using Azure Cosmos DB accounts' managed identities article.

Support for multiple Azure Synapse Analytics runtimes

The analytical store is optimized to provide scalability, elasticity, and performance for analytical workloads without any dependency on the compute run-times. The storage technology is self-managed to optimize your analytics workloads without manual efforts.

By decoupling the analytical storage system from the analytical compute system, data in Azure Cosmos DB analytical store can be queried simultaneously from the different analytics runtimes supported by Azure Synapse Analytics. As of today, Azure Synapse Analytics supports Apache Spark and serverless SQL pool with Azure Cosmos DB analytical store.

> **NOTE**
> You can only read from analytical store using Azure Synapse Analytics runtimes. And the opposite is also true, Azure Synapse Analytics runtimes can only read from analytical store. Only the auto-sync process can change data in analytical store. You can write data back to Cosmos DB transactional store using Azure Synapse Analytics Spark pool, using the built-in Azure Cosmos DB OLTP SDK.

Pricing

Analytical store follows a consumption-based pricing model where you're charged for:

- Storage: the volume of the data retained in the analytical store every month including historical data as defined by analytical TTL.
- Analytical write operations: the fully managed synchronization of operational data updates to the analytical store from the transactional store (auto-sync)
- Analytical read operations: the read operations performed against the analytical store from Azure Synapse Analytics Spark pool and serverless SQL pool run times.

Analytical store pricing is separate from the transaction store pricing model. There's no concept of provisioned RUs in the analytical store. See Azure Cosmos DB pricing page for full details on the pricing model for analytical store.

Data in the analytics store can only be accessed through Azure Synapse Link, which is done in the Azure Synapse Analytics runtimes: Azure Synapse Apache Spark pools and Azure Synapse serverless SQL pools. See Azure Synapse Analytics pricing page for full details on the pricing model to access data in analytical store.

In order to get a high-level cost estimate to enable analytical store on an Azure Cosmos DB container, from the analytical store perspective, you can use the Azure Cosmos DB Capacity planner and get an estimate of your analytical storage and write operations costs. Analytical read operations costs depends on the analytics workload characteristics but as a high-level estimate, scan of 1 TB of data in analytical store typically results in 130,000 analytical read operations, and results in a cost of $0.065.

> **NOTE**
>
> Analytical store read operations estimates aren't included in the Cosmos DB cost calculator since they are a function of your analytical workload. While the above estimate is for scanning 1TB of data in analytical store, applying filters reduces the volume of data scanned and this determines the exact number of analytical read operations given the consumption pricing model. A proof-of-concept around the analytical workload would provide a more finer estimate of analytical read operations. This estimate doesn't include the cost of Azure Synapse Analytics.

Next steps

To learn more, see the following docs:

- Azure Synapse Link for Azure Cosmos DB
- Check out the learn module on how to Design hybrid transactional and analytical processing using Azure Synapse Analytics
- Get started with Azure Synapse Link for Azure Cosmos DB
- Frequently asked questions about Synapse Link for Azure Cosmos DB
- Azure Synapse Link for Azure Cosmos DB Use cases

Azure Synapse Link for Azure Cosmos DB supported features

8/19/2022 • 2 minutes to read • Edit Online

This article describes the functionalities that are currently supported in Azure Synapse Link for Azure Cosmos DB.

Azure Synapse support

There are two types of containers in Azure Cosmos DB:

- HTAP container - A container with Synapse Link enabled. This container has both transactional store and analytical store.
- OLTP container - A container with Synaspe Link not enabled. This container has only transactional store and no analytical store.

You can connect to an Azure Cosmos DB container without enabling Synapse Link. In this scenario, you can only read/write to the transactional store. What follows is a list of the currently supported features within Synapse Link for Azure Cosmos DB.

CATEGORY	DESCRIPTION	APACHE SPARK POOL	SERVERLESS SQL POOL
Run-time Support	Supported Azure Synapse runtime to access Azure Cosmos DB	✓	✓
Azure Cosmos DB API support	Supported Azure Cosmos DB API kind	SQL / MongoDB	SQL / MongoDB
Object	Objects such as a table that can be created, pointing directly to Azure Cosmos DB container	Dataframe, View, Table	View
Read	Type of Azure Cosmos DB container that can be read	OLTP / HTAP	HTAP
Write	Can the Azure Synapse runtime be used to write data to an Azure Cosmos DB container	Yes	No

- If you write data into an Azure Cosmos DB container from Spark, this process happens through the transactional store of Azure Cosmos DB. It will impact the transactional performance of Azure Cosmos DB by consuming Request Units.
- Dedicated SQL pool integration through external tables is currently not supported.

Supported code-generated actions for Spark

GESTURE	DESCRIPTION	OLTP	HTAP
Load to DataFrame	Load and read data into a Spark DataFrame	✓	✓
Create Spark table	Create a table pointing to an Azure Cosmos DB container	✓	✓
Write DataFrame to container	Write data into a container	✓	✓
Load streaming DataFrame from container	Stream data using Azure Cosmos DB change feed	✓	✓
Write streaming DataFrame to container	Stream data using Azure Cosmos DB change feed	✓	✓

Supported code-generated actions for serverless SQL pool

GESTURE	DESCRIPTION	OLTP	HTAP
Explore data	Explore data from a container with familiar T-SQL syntax and automatic schema inference	X	✓
Create views and build BI reports	Create a SQL view to have direct access to a container for BI through serverless SQL pool	X	✓
Join disparate data sources along with Cosmos DB data	Store results of query reading data from Cosmos DB containers along with data in Azure Blob Storage or Azure Data Lake Storage using CETAS	X	✓

Next steps

- See how to connect to Synapse Link for Azure Cosmos DB
- Learn how to query the Cosmos DB Analytical Store with Spark 3
- Learn how to query the Cosmos DB Analytical Store with Spark 2

Azure Synapse Link for Azure Cosmos DB: Near real-time analytics use cases

8/19/2022 • 8 minutes to read • Edit Online

APPLIES TO: SQL API Azure Cosmos DB API for MongoDB

Azure Synapse Link for Azure Cosmos DB is a cloud native hybrid transactional and analytical processing (HTAP) capability that enables you to run near real-time analytics over operational data. Synapse Link creates a tight seamless integration between Azure Cosmos DB and Azure Synapse Analytics.

You might be curious to understand what industry use cases can leverage this cloud native HTAP capability for near real-time analytics over operational data. Here are three common use cases for Azure Synapse Link for Azure Cosmos DB:

- Supply chain analytics, forecasting & reporting
- Real-time personalization
- Predictive maintenance, anomaly detection in IOT scenarios

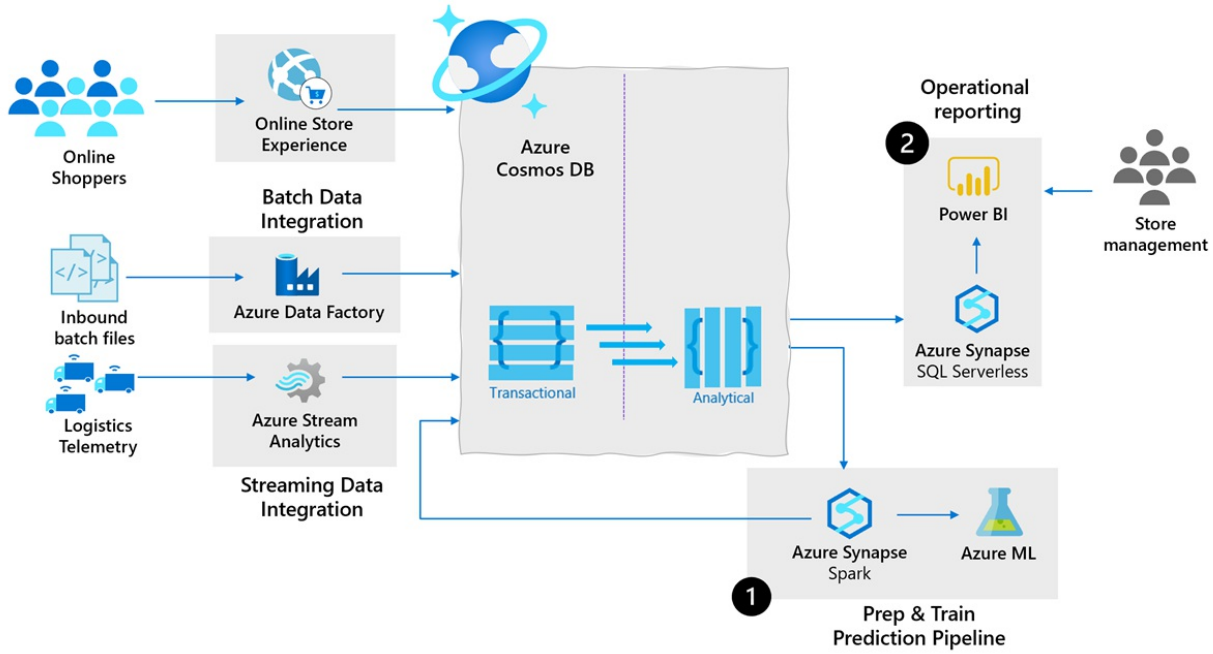
> **NOTE**
> Azure Synapse Link for Azure Cosmos DB targets the scenario where enterprise teams are looking to run near real-time analytics. These analytics are run without ETL over operational data generated across transactional applications built on Azure Cosmos DB. This does not replace the need for a separate data warehouse when there are traditional data warehouse requirements such as workload management, high concurrency, persistence aggregates across multiple data sources.

Supply chain analytics, forecasting & reporting

Research studies show that embedding big data analytics in supply chain operations leads to improvements in order-to-cycle delivery times and supply chain efficiency.

Manufacturers are onboarding to cloud-native technologies to break out of constraints of legacy Enterprise Resource Planning (ERP) and Supply Chain Management (SCM) systems. With supply chains generating increasing volumes of operational data every minute (order, shipment, transaction data), manufacturers need an operational database. This operational database should scale to handle the data volumes as well as an analytical platform to get to a level of real-time contextual intelligence to stay ahead of the curve.

The following architecture shows the power of leveraging Azure Cosmos DB as the cloud-native operational database and Synapse Link in supply chain analytics:



Based on previous architecture, you can achieve the following use cases with Synapse Link for Azure Cosmos DB:

- **Prepare & train predictive pipeline:** Generate insights over the operational data across the supply chain using machine learning translates. This way you can lower inventory, operations costs, and reduce the order-to-delivery times for customers.

  Synapse Link allows you to analyze the changing operational data in Azure Cosmos DB without any manual ETL processes. It saves you from additional cost, latency, and operational complexity. Synapse Link enables data engineers and data scientists to build robust predictive pipelines:

  - Query operational data from Azure Cosmos DB analytical store by leveraging native integration with Apache Spark pools in Azure Synapse Analytics. You can query the data in an interactive notebook or scheduled remote jobs without complex data engineering.
  - Build Machine Learning (ML) models with Spark ML algorithms and Azure ML integration in Azure Synapse Analytics.
  - Write back the results after model inference into Azure Cosmos DB for operational near-real-time scoring.

- **Operational reporting:** Supply chain teams need flexible and custom reports over real-time, accurate operational data. These reports are required to obtain a snapshot view of supply chain effectiveness, profitability, and productivity. It allows data analysts and other key stakeholders to constantly reevaluate the business and identify areas to tweak to reduce operational costs.

  Synapse Link for Azure Cosmos DB enables rich business intelligence (BI)/reporting scenarios:

  - Query operational data from Azure Cosmos DB analytical store by using native integration with serverless SQL pool and full expressiveness of T-SQL language.
  - Model and publish auto refreshing BI dashboards over Azure Cosmos DB through serverless SQL pool support for familiar BI tools. For example, Azure Analysis Services, Power BI Premium, etc.

The following is some guidance for data integration for batch & streaming data into Azure Cosmos DB:

- **Batch data integration & orchestration:** With supply chains getting more complex, supply chain data platforms need to integrate with variety of data sources and formats. Azure Synapse comes built-in with the same data integration engine and experiences as Azure Data Factory. This integration allows data

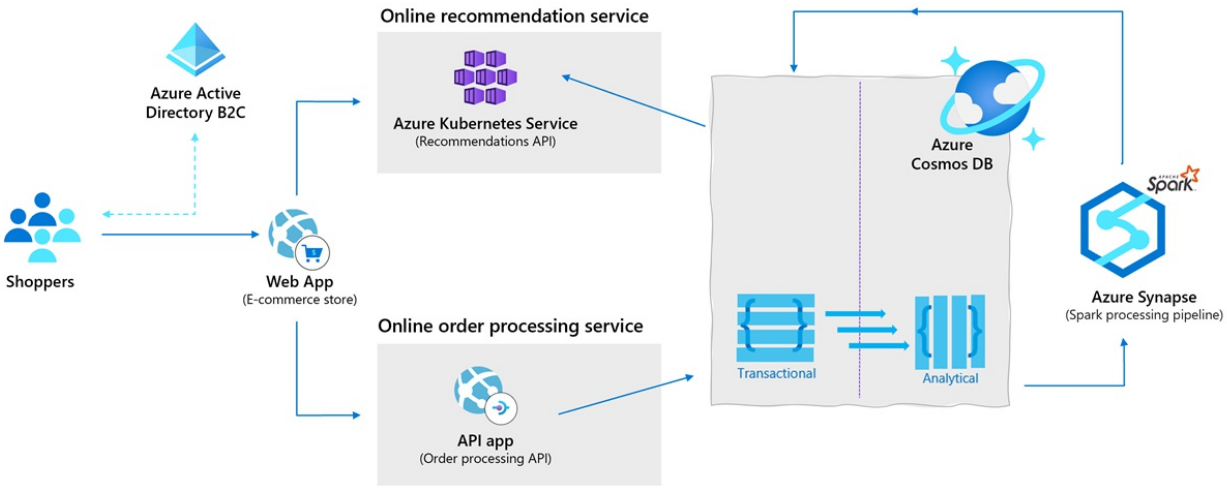
engineers to create rich data pipelines without a separate orchestration engine:

  - Move data from 85+ supported data sources to Azure Cosmos DB with Azure Data Factory.
  - Write code-free ETL pipelines to Azure Cosmos DB including relational-to-hierarchical and hierarchical-to-hierarchical mappings with mapping data flows.

- **Streaming data integration & processing:** With the growth of Industrial IoT (sensors tracking assets from 'floor-to-store', connected logistics fleets, etc.), there is an explosion of real-time data being generated in a streaming fashion that needs to be integrated with traditional slow moving data for generating insights. Azure Stream Analytics is a recommended service for streaming ETL and processing on Azure with a wide range of scenarios. Azure Stream Analytics supports Azure Cosmos DB as a native data sink.

Real-time personalization

Retailers today must build secure and scalable e-commerce solutions that meet the demands of both customers and business. These e-commerce solutions need to engage customers through customized products and offers, process transactions quickly and securely, and focus on fulfillment and customer service. Azure Cosmos DB along with the latest Synapse Link for Azure Cosmos DB allows retailers to generate personalized recommendations for customers in real time. They use low-latency and tunable consistency settings for immediate insights as shown in the following architecture:



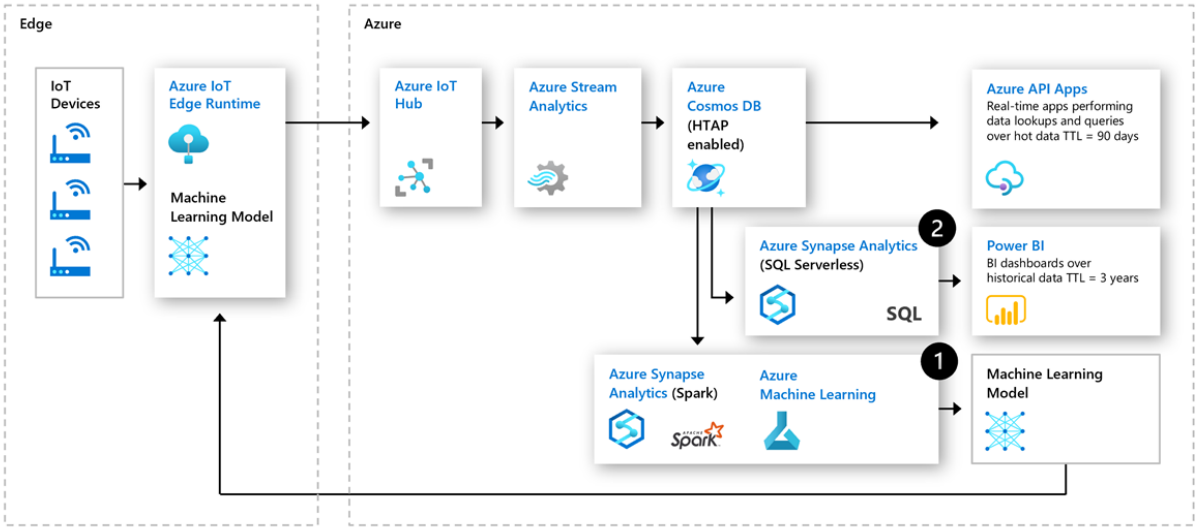
Synapse Link for Azure Cosmos DB use case:

- **Prepare & train predictive pipeline:** You can generate insights over the operational data across your business units or customer segments using Synapse Spark and machine learning models. This translates to personalized delivery to target customer segments, predictive end-user experiences and targeted marketing to fit your end-user requirements.

IOT predictive maintenance

Industrial IOT innovations have drastically reduced downtimes of machinery and increased overall efficiency across all fields of industry. One of such innovations is predictive maintenance analytics for machinery at the edge of the cloud.

The following is an architecture leveraging the cloud native HTAP capabilities of Azure Synapse Link for Azure Cosmos DB in IoT predictive maintenance:



Synapse Link for Azure Cosmos DB use cases:

- **Prepare & train predictive pipeline:** The historical operational data from IoT device sensors could be used to train predictive models such as anomaly detectors. These anomaly detectors are then deployed back to the edge for real-time monitoring. Such a virtuous loop allows for continuous retraining of the predictive models.
- **Operational reporting:** With the growth of digital twin initiatives, companies are collecting vast amounts of operational data from large number of sensors to build a digital copy of each machine. This data powers BI needs to understand trends over historical data in addition to real-time applications over recent hot data.

Sample scenario: HTAP for Azure Cosmos DB

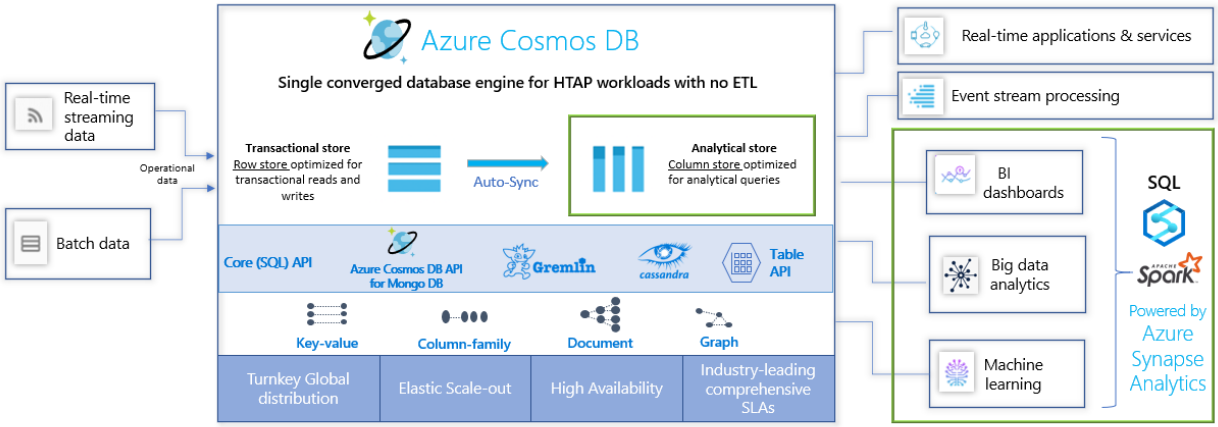
For nearly a decade, Azure Cosmos DB has been used by thousands of customers for mission critical applications that require elastic scale, turnkey global distribution, multi-region write replication for low latency and high availability of both reads & writes in their transactional workloads.

The following list shows an overview of the various workload patterns that are supported with operational data using Azure Cosmos DB:

- Real-time apps & services
- Event stream processing
- BI dashboards
- Big data analytics
- Machine learning

Azure Synapse Link enables Azure Cosmos DB to not just power transactional workloads but also perform near real-time analytical workloads over historical operational data. It happens with no ETL requirements and guaranteed performance isolation from the transactional workloads.

The following image shows workload patterns using Azure Cosmos DB:



Let us take the example of an e-commerce company CompanyXYZ with global operations across 20 countries/regions to illustrate the benefits of choosing Azure Cosmos DB as the single real-time database powering both transactional and analytical requirements of an inventory management platform.

- CompanyXYZ's core business depends on the inventory management system – hence availability & reliability are core pillar requirements. Benefits of using Azure Cosmos DB:
  - By virtue of deep integration with Azure infrastructure, and transparent multi-region writes, global replication, Azure Cosmos DB provides industry-leading 99.999% high availability against regional outages.
- CompanyXYZ's supply chain partners may be in separate geographic locations but they may have to see a single view of the product inventory across the globe to support their local operations. This includes the need to be able to read updates made by other supply chain partners in real time. As well as being able to make updates without worrying about conflicts with other partners at high throughput. Benefits of using Azure Cosmos DB:
  - With its unique multi-region writes replication protocol and latch-free, write-optimized transactional store, Azure Cosmos DB guarantees less than 10-ms latencies for both indexed reads and writes at the 99th percentile globally.
  - High throughput ingestion of both batch & streaming data feeds with real-time indexing in transactional store.
  - Azure Cosmos DB transactional store provides three more options than the two extremes of strong and eventual consistency levels to achieve the availability vs performance tradeoffs closest to the business need.
- CompanyXYZ's supply chain partners have highly fluctuating traffic patterns ranging from hundreds to millions of requests/s and thus the inventory management platform needs to deal with unexpected burstiness in traffic. Benefits of using Azure Cosmos DB:
  - Azure Cosmos DB's transactional store supports elastic scalability of storage and throughput using horizontal partitioning. Containers and databases configured in Autopilot mode can automatically and instantly scale the provisioned throughput based on the application needs without impacting the availability, latency, throughput, or performance of the workload globally.
- CompanyXYZ needs to establish a secure analytics platform to house system-wide historical inventory data to enable analytics and insights across supply chain partner, business units and functions. The analytics platform needs to enable collaboration across the system, traditional BI/reporting use cases, advanced analytics use cases and predictive intelligent solutions over the operational inventory data.

Benefits of using Synapse Link for Azure Cosmos DB:

- By using Azure Cosmos DB analytical store, a fully isolated column store, Synapse Link enables no Extract-Transform-Load (ETL) analytics in Azure Synapse Analytics against globally distributed operational data at scale. Business analysts, data engineers and data scientists can now use Synapse Spark or Synapse SQL in an interoperable manner to run near real-time business intelligence, analytics, and machine learning pipelines without impacting the performance of their transactional workloads on Azure Cosmos DB. See the benefits of Synapse Link in Azure Cosmos DB for more details.

Next steps

To learn more, see the following docs:

- Azure Synapse Link for Azure Cosmos DB
- Azure Cosmos DB Analytical Store
- Working with Azure Synapse Link for Azure Cosmos DB
- Frequently asked questions about Azure Synapse Link for Azure Cosmos DB
- Apache Spark in Azure Synapse Analytics
- Serverless SQL pool runtime support in Azure Synapse Analytics

Azure Synapse Link for Azure SQL Database (Preview)

8/19/2022 • 3 minutes to read • Edit Online

This article helps you to understand the functions of Azure Synapse Link for Azure SQL Database. You can use the Azure Synapse Link for SQL functionality to replicate your operational data into an Azure Synapse Analytics dedicated SQL pool from Azure SQL Database.

> **IMPORTANT**
>
> Azure Synapse Link for SQL is currently in PREVIEW. See the Supplemental Terms of Use for Microsoft Azure Previews for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

Link connection

A link connection identifies a mapping relationship between an Azure SQL database and an Azure Synapse Analytics dedicated SQL pool. You can create, manage, monitor and delete link connections in your Synapse workspace. When creating a link connection, you can select both source database and a destination Synapse dedicated SQL pool so that the operational data from your source database will be automatically replicated to the specified destination Synapse dedicated SQL pool. You can also add or remove one or more tables from your source database to be replicated.

You can start or stop a link connection. When started, a link connection will start from a full initial load from your source database followed by incremental change feeds via the change feed feature in Azure SQL database. When you stop a link connection, the updates made to the operational data won't be synchronized to your Synapse dedicated SQL pool. For more information, see Azure Synapse Link change feed for SQL Server 2022 and Azure SQL Database.

You need to select compute core counts for each link connection to replicate your data. The core counts represent the compute power and it impacts your data replication latency and cost.

You also have the chance to make a trade-off between cost and latency by selecting the continuous or batch mode to replicate the data. When you select continuous mode, the runtime will be running continuously so that any changes applied to your SQL DB or SQL Server will be replicated to Synapse with low latency. When you select batch mode with specified interval, the changes applied to your SQL DB or SQL Server will be accumulated and replicated to Synapse in a batch mode with specified interval. By doing so, you can save cost as you are only charged for the time when the runtime is required to replicate your data. After each batch of data is replicated, the runtime will be shut down automatically.

Monitoring

You can monitor Azure Synapse Link for SQL at the link and table levels. For each link connection, you'll see the following status:

- **Initial:** a link connection is created but not started. You will not be charged in initial state.
- **Starting:** a link connection is setting up compute engines to replicate data.
- **Running:** a link connection is replicating data.
- **Stopping:** a link connection is shutting down the compute engines.
- **Stopped:** a link connection is stopped. You will not be charged in stopped state.

For each table, you'll see the following status:

- **Snapshotting:** a source table is initially loaded to the destination with full snapshot.
- **Replicating:** any updates on source table are replicated to the destination.
- **Failed:** the data on source table can't be replicated to destination due to a fatal error. If you want to retry after fixing the error, remove the table from link connection and add it back.
- **Suspended:** replication is suspended for this table due to an error. It will be resumed after the error is resolved.

Transactional consistency across tables

You can enable transactional consistency across tables for each link connection. However, it limits overall replication throughput.

Known limitations

A consolidated list of known limitations and issues can be found at Known limitations and issues with Azure Synapse Link for SQL (Preview).

Next steps

- To learn more, see how to Configure Synapse Link for Azure SQL Database (Preview).

Azure Synapse Link for SQL Server 2022 (Preview)

8/19/2022 • 4 minutes to read • Edit Online

This article helps you to understand the functions of Azure Synapse Link for SQL Server 2022. You can use the Azure Synapse Link for SQL functionality to replicate your operational data into an Azure Synapse Analytics dedicated SQL pool from SQL Server 2022.

> **IMPORTANT**
>
> Azure Synapse Link for SQL is currently in PREVIEW. See the Supplemental Terms of Use for Microsoft Azure Previews for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

Link connection

A link connection identifies a mapping relationship between an SQL Server 2022 and an Azure Synapse Analytics dedicated SQL pool. You can create, manage, monitor and delete link connections in your Synapse workspace. When creating a link connection, you can select both source database and destination Synapse dedicated SQL pool so that the operational data from your source database will be automatically replicated to the specified destination Synapse dedicated SQL pool. You can also add or remove one or more tables from your source database to be replicated.

You can start or stop a link connection. When started, a link connection will start from a full initial load from your source database followed by incremental change feeds via change feed feature in SQL Server 2022. When you stop a link connection, the updates made to the operational data won't be synchronized to your Synapse dedicated SQL pool. For more information, see Azure Synapse Link change feed for SQL Server 2022 and Azure SQL Database.

You need to select compute core counts for each link connection to replicate your data. The core counts represent the compute power and it impacts your data replication latency and cost.

You also have the chance to make a trade-off between cost and latency by selecting the continuous or batch mode to replicate the data. When you select continuous mode, the runtime will be running continuously so that any changes applied to your SQL DB or SQL Server will be replicated to Synapse with low latency. When you select batch mode with specified interval, the changes applied to your SQL DB or SQL Server will be accumulated and replicated to Synapse in a batch mode with specified interval. By doing so, you can save cost as you are only charged for the time when the runtime is required to replicate your data. After each batch of data is replicated, the runtime will be shut down automatically.

Landing zone

The landing zone is an interim staging store required for Azure Synapse Link for SQL Server 2022. First, the operational data is loaded from the SQL Server 2022 to the landing zone. Next, the data is copied from the landing zone to the Synapse dedicated SQL pool. You need to provide your own Azure Data Lake Storage Gen2 account to be used as a landing zone. It is not supported to use this landing zone for anything other than Azure Synapse Link for SQL.

The shared access signature (SAS) token from your Azure Data Lake Storage Gen2 account is required for a link connection to get access to the landing zone. Be aware that the SAS token has an expiration date. Make sure to rotate the SAS token before the expiration date to ensure the SAS token is valid. Otherwise, Azure Synapse Link for SQL will fail to replicate the data from SQL Server 2022.

Self-hosted integration runtime

Self-hosted integration runtime is a software agent that you can download and install on an on-premises machine or a virtual machine. It is required for Azure Synapse Link for SQL Server 2022 to get access the data on SQL Server 2022 on premises that is behind the firewall. Currently, the self-hosted IR is only supported on a Windows operating system. For more information, see Create a self-hosted integration runtime

Monitoring

You can monitor Azure Synapse Link for SQL at the link and table levels. For each link connection, you'll see the following status:

- **Initial:** a link connection is created but not started. You will not be charged in initial state.
- **Starting:** a link connection is setting up compute engines to replicate data.
- **Running:** a link connection is replicating data.
- **Stopping:** a link connection is shutting down the compute engines.
- **Stopped:** a link connection is stopped. You will not be charged in stopped state.

For each table, you'll see the following status:

- **Snapshotting:** a source table is initially loaded to the destination with full snapshot.
- **Replicating:** any updates on source table are replicated to the destination.
- **Failed:** the data on source table can't be replicated to destination. If you want to retry after fixing the error, remove the table from link connection and add it back.
- **Suspended:** replication is suspended for this table due to an error. It will be resumed after the error is resolved.

For more information, see Manage Synapse Link for SQL change feed.

Transactional consistency across tables

You can enable transactional consistency across table for each link connection. However, it limits overall replication throughput.

Known limitations

A consolidated list of known limitations and issues can be found at Known limitations and issues with Azure Synapse Link for SQL (Preview).

Next steps

- To learn more, see how to Configure Synapse Link for SQL Server 2022 (Preview).

Configure and use Azure Synapse Link for Azure Cosmos DB

8/19/2022 • 13 minutes to read • Edit Online

APPLIES TO: SQL API Azure Cosmos DB API for MongoDB

Azure Synapse Link for Azure Cosmos DB is a cloud-native hybrid transactional and analytical processing (HTAP) capability that enables you to run near real-time analytics over operational data in Azure Cosmos DB. Synapse Link creates a tight seamless integration between Azure Cosmos DB and Azure Synapse Analytics.

Azure Synapse Link is available for Azure Cosmos DB SQL API or for Azure Cosmos DB API for Mongo DB accounts. Use the following steps to run analytical queries with the Azure Synapse Link for Azure Cosmos DB:

- Enable Azure Synapse Link for your Azure Cosmos DB accounts
- Create an analytical store enabled container
- Enable analytical store in an existing container
- Optional - Update analytical store ttl for a container
- Optional - Disable analytical store in a container
- Connect your Azure Cosmos database to an Azure Synapse workspace
- Query the analytical store using Azure Synapse Spark Pool
- Query the analytical store using Azure Synapse serverless SQL pool
- Use Azure Synapse serverless SQL pool to analyze and visualize data in Power BI

You can also checkout the learn module on how to configure Azure Synapse Link for Azure Cosmos DB.

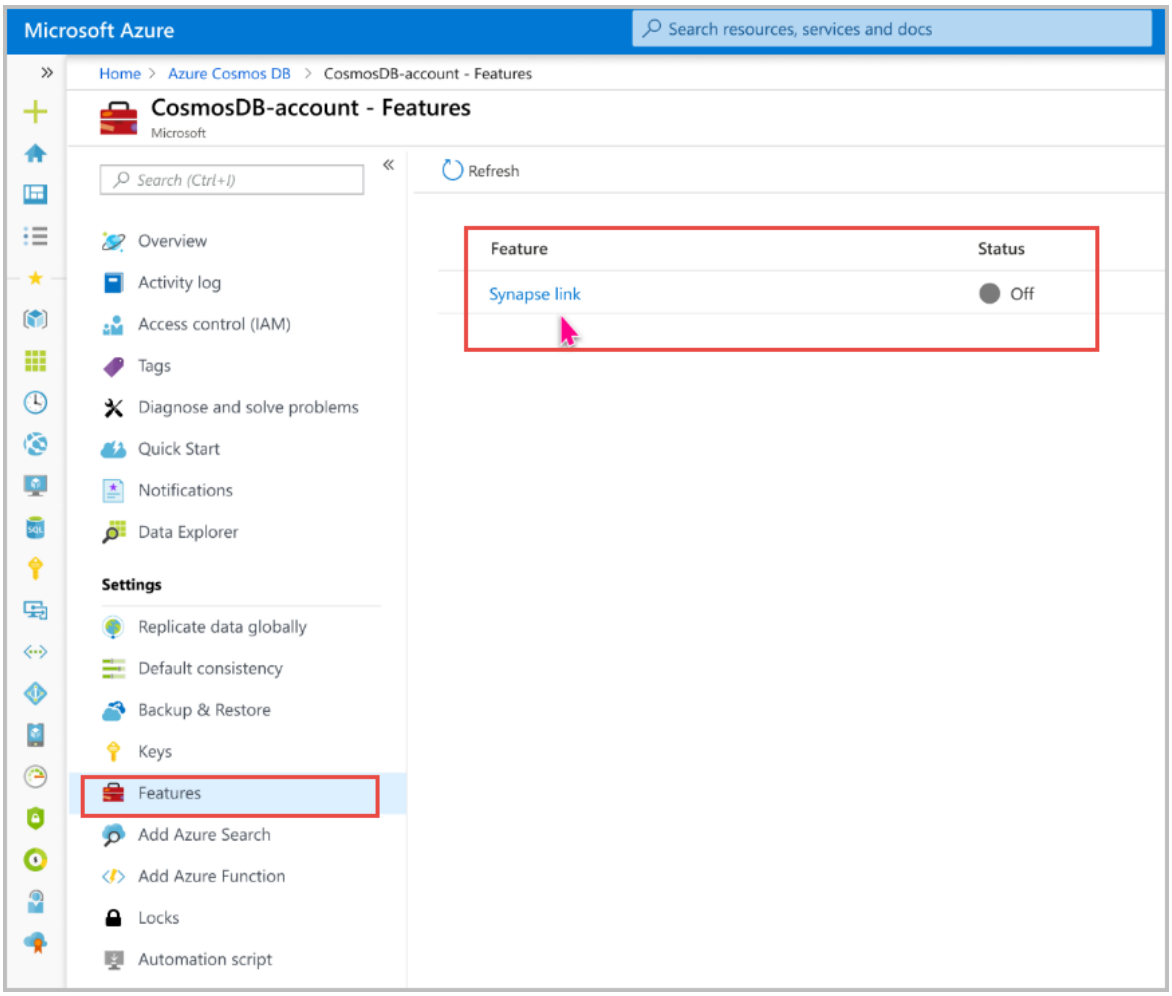
Enable Azure Synapse Link for Azure Cosmos DB accounts

> **NOTE**
> If you want to use customer-managed keys with Azure Synapse Link, you must configure your account's managed identity in your Azure Key Vault access policy before enabling Synapse Link on your account. To learn more, see how to Configure customer-managed keys using Azure Cosmos DB accounts' managed identities article.

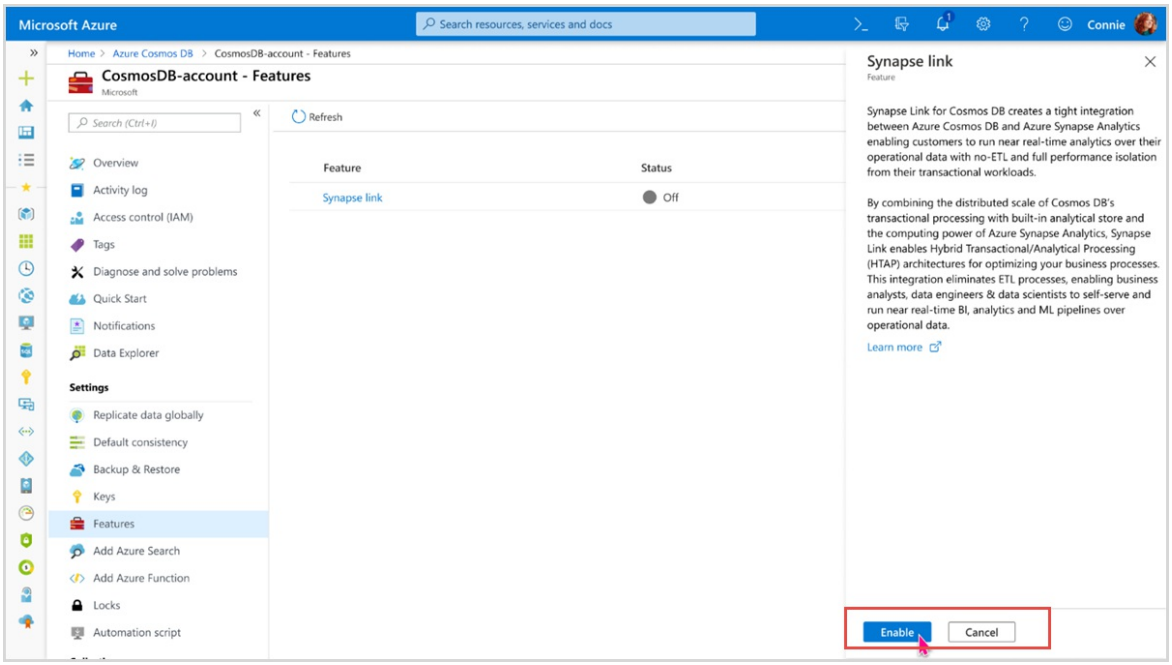
> **NOTE**
> If you want to use Full Fidelity Schema for SQL (CORE) API accounts, you can't use the Azure portal to enable Synapse Link. This option can't be changed after Synapse Link is enabled in your account and to set it you must use Azure CLI or PowerShell. For more information, check analytical store schema representation documentation.

Azure portal

1. Sign into the Azure portal.
2. Create a new Azure account, or select an existing Azure Cosmos DB account.
3. Navigate to your Azure Cosmos DB account and open the **Features** pane.
4. Select **Synapse Link** from the features list.



5. Next it prompts you to enable synapse link on your account. Select **Enable**. This process can take 1 to 5 minutes to complete.



6. Your account is now enabled to use Synapse Link. Next see how to create analytical store enabled containers to automatically start replicating your operational data from the transactional store to the analytical store.

> **NOTE**
>
> Turning on Synapse Link does not turn on the analytical store automatically. Once you enable Synapse Link on the Cosmos DB account, enable analytical store on containers to start using Synapse Link.

Command-Line Tools

Enable Synapse Link in your Cosmos DB SQL API or MongoDB API account using Azure CLI or PowerShell.

Azure CLI

Use `--enable-analytical-storage true` for both **create** or **update** operations. You also need to choose the representation schema type. For SQL API accounts you can use `--analytical-storage-schema-type` with the values `FullFidelity` or `WellDefined`. For MongoDB API accounts, always use `--analytical-storage-schema-type FullFidelity`.

- Create a new Azure Cosmos DB account with Synapse Link enabled
- Update an existing Azure Cosmos DB account to enable Synapse Link

PowerShell

Use `EnableAnalyticalStorage true` for both **create** or **update** operations. You also need to choose the representation schema type. For SQL API accounts you can use `--analytical-storage-schema-type` with the values `FullFidelity` or `WellDefined`. For MongoDB API accounts, always use `-AnalyticalStorageSchemaType FullFidelity`.

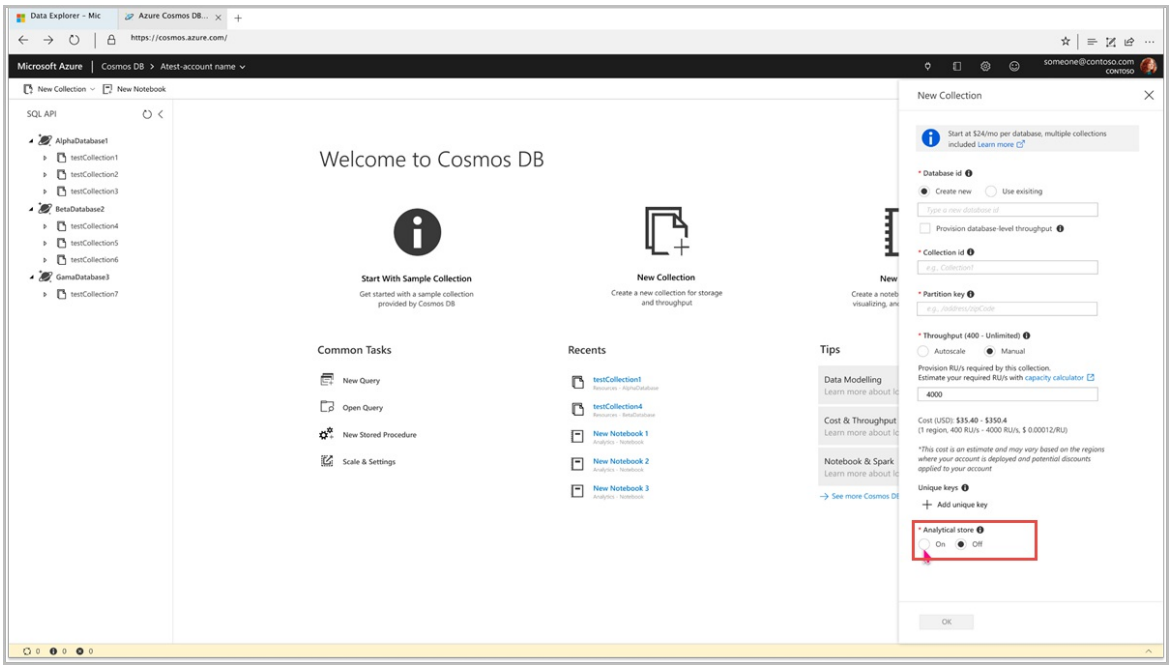
- Create a new Azure Cosmos DB account with Synapse Link enabled
- Update an existing Azure Cosmos DB account to enable Synapse Link

Create an analytical store enabled container

You can turn on analytical store when creating an Azure Cosmos DB container by using one of the following options.

Azure portal

1. Sign in to the Azure portal or the Azure Cosmos DB Explorer.
2. Navigate to your Azure Cosmos DB account and open the **Data Explorer** tab.
3. Select **New Container** and enter a name for your database, container, partition key and throughput details. Turn on the **Analytical store** option. After you enable the analytical store, it creates a container with `analytical TTL` property set to the default value of -1 (infinite retention). This analytical store that retains all the historical versions of records and can be changed later.



4. If you have previously not enabled Synapse Link on this account, it will prompt you to do so because it's a pre-requisite to create an analytical store enabled container. If prompted, select **Enable Synapse Link**. This process can take 1 to 5 minutes to complete.

5. Select **OK**, to create an analytical store enabled Azure Cosmos DB container.

6. After the container is created, verify that analytical store has been enabled by clicking **Settings**, right below Documents in Data Explorer, and check if the **Analytical Store Time to Live** option is turned on.

Azure Cosmos DB SDKs

Set the `analytical TTL` property to the required value to create an analytical store enabled container. For the list of allowed values, see the analytical TTL supported values article.

.NET SDK

The following code creates a container with analytical store by using the .NET SDK. Set the `AnalyticalStoreTimeToLiveInSeconds` property to the required value in seconds or use `-1` for infinite retention. This setting can be changed later.

```
// Create a container with a partition key, and analytical TTL configured to -1 (infinite retention)
ContainerProperties properties = new ContainerProperties()
{
    Id = "myContainerId",
    PartitionKeyPath = "/id",
    AnalyticalStoreTimeToLiveInSeconds = -1,
};
CosmosClient cosmosClient = new CosmosClient("myConnectionString");
await cosmosClient.GetDatabase("myDatabase").CreateContainerAsync(properties);
```

Java V4 SDK

The following code creates a container with analytical store by using the Java V4 SDK. Set the `AnalyticalStoreTimeToLiveInSeconds` property to the required value in seconds or use `-1` for infinite retention. This setting can be changed later.

```
// Create a container with a partition key and  analytical TTL configured to  -1 (infinite retention)
CosmosContainerProperties containerProperties = new CosmosContainerProperties("myContainer",
"/myPartitionKey");

containerProperties.setAnalyticalStoreTimeToLiveInSeconds(-1);

container = database.createContainerIfNotExists(containerProperties, 400).block().getContainer();
```

Python V4 SDK

The following code creates a container with analytical store by using the Python V4 SDK. Set the `analytical_storage_ttl` property to the required value in seconds or use `-1` for infinite retention. This setting can be changed later.

```
# Azure Cosmos DB Python SDK, for SQL API only.
# Creating an analytical store enabled container.

import azure.cosmos as cosmos
import azure.cosmos.cosmos_client as cosmos_client
import azure.cosmos.exceptions as exceptions
from azure.cosmos.partition_key import PartitionKey

HOST = 'your-cosmos-db-account-URI'
KEY = 'your-cosmos-db-account-key'
DATABASE = 'your-cosmos-db-database-name'
CONTAINER = 'your-cosmos-db-container-name'

# Client
client = cosmos_client.CosmosClient(HOST,  KEY )

# Database client
try:
    db = client.create_database(DATABASE)

except exceptions.CosmosResourceExistsError:
    db = client.get_database_client(DATABASE)

# Creating the container with analytical store enabled
try:
    container = db.create_container(
        id=CONTAINER,
        partition_key=PartitionKey(path='/id', kind='Hash'),analytical_storage_ttl=-1
    )
    properties = container.read()
    print('Container with id \'{0}\' created'.format(container.id))
    print('Partition Key - \'{0}\''.format(properties['partitionKey']))

except exceptions.CosmosResourceExistsError:
    print('A container with already exists')
```

Command-Line Tools

Set the `analytical TTL` property to the required value to create an analytical store enabled container. For the list of allowed values, see the analytical TTL supported values article.

Azure CLI

The following options create a container with analytical store by using Azure CLI. Set the `--analytical-storage-ttl` property to the required value in seconds or use `-1` for infinite retention. This setting can be changed later.

- Create an Azure Cosmos DB MongoDB collection
- Create an Azure Cosmos DB SQL API container

PowerShell

The following options create a container with analytical store by using PowerShell. Set the `-AnalyticalStorageTtl` property to the required value in seconds or use `-1` for infinite retention. This setting can be changed later.

- Create an Azure Cosmos DB MongoDB collection
- Create an Azure Cosmos DB SQL API container

Enable analytical store in an existing container

> **NOTE**
> You can turn on analytical store on existing Azure Cosmos DB SQL API containers. This capability is general available and can be used for production workloads.

Please note the following details when enabling Azure Synapse Link on your existing containers:

- The same performance isolation of the analytical store auto-sync process applies to the initial sync and there is no performance impact on your OLTP workload.
- A container's initial sync with analytical store total time will vary depending on the data volume and on the documents complexity. This process can take anywhere from a few seconds to multiple days. Please use the Azure portal to monitor the migration progress.
- The throughput of your container, or database account, also influences the total initial sync time. Although RU/s are not used in this migration, the total RU/s available influences the performance of the process. You can temporarily increase your environment's available RUs to speed up the process.
- You won't be able to query analytical store of an existing container while Synapse Link is being enabled on that container. Your OLTP workload isn't impacted and you can keep on reading data normally. Data ingested after the start of the initial sync will be merged into analytical store by the regular analytical store auto-sync process.
- Currently existing MongoDB API collections are not supported. The alternative is to migrate the data into a new collection, created with analytical store turned on.

Azure portal

1. Sign in to the Azure portal.
2. Navigate to your Azure Cosmos DB account and open the **Azure Synapse Link"** tab in the **Integrations** left navigation section. In this tab you can enable Synapse Link in your database account and you can enable Synapse Link on your existing containers.
3. After you click the blue **Enable Synapse Link on your container(s)** button, you will start to see the progress of your containers initial sync progress.
4. Optionally, you can go to the **Power BI** tab, also in the **Integrations** section, to create Power BI dashboards on your Synapse Link enabled containers.

Command-Line Tools

Set the `analytical TTL` property to `-1` for infinite retention or use a positive integer to specify the number of seconds that the data will be retain in analytical store. For more information, see the analytical TTL supported values article.

Azure CLI

- Use az cosmosdb sql container update to update `--analytical-storage-ttl`.
- Check the migration status in the Azure portal.

PowerShell

- Use Update Analytical ttl to update `-AnalyticalStorageTtl`.
- Check the migration status in the Azure portal.

Optional - Update the analytical store time to live

After the analytical store is enabled with a particular TTL value, you may want to update it to a different valid value. You can update the value by using the Azure portal, Azure CLI, PowerShell, or Cosmos DB SDKs. For information on the various Analytical TTL config options, see the analytical TTL supported values article.

Azure portal

If you created an analytical store enabled container through the Azure portal, it contains a default `analytical TTL` of `-1`. Use the following steps to update this value:

1. Sign in to the Azure portal or the Azure Cosmos DB Explorer.
2. Navigate to your Azure Cosmos DB account and open the **Data Explorer** tab.
3. Select an existing container that has analytical store enabled. Expand it and modify the following values:
   a. Open the **Scale & Settings** window.
   b. Under **Setting** find, **Analytical Storage Time to Live**.
   c. Select **On (no default)** or select **On** and set a TTL value.
   d. Click **Save** to save the changes.

.NET SDK

The following code shows how to update the TTL for analytical store by using the .NET SDK:

```
// Get the container, update AnalyticalStorageTimeToLiveInSeconds
ContainerResponse containerResponse = await client.GetContainer("database",
"container").ReadContainerAsync();
// Update analytical store TTL
containerResponse.Resource. AnalyticalStorageTimeToLiveInSeconds = 60 * 60 * 24 * 180  // Expire analytical
store data in 6 months;
await client.GetContainer("database", "container").ReplaceContainerAsync(containerResponse.Resource);
```

Java V4 SDK

The following code shows how to update the TTL for analytical store by using the Java V4 SDK:

```
CosmosContainerProperties containerProperties = new CosmosContainerProperties("myContainer",
"/myPartitionKey");

// Update analytical store TTL to expire analytical store data in 6 months;
containerProperties.setAnalyticalStoreTimeToLiveInSeconds (60 * 60 * 24 * 180 );

// Update container settings
container.replace(containerProperties).block();
```

Python V4 SDK

Currently not supported.

Azure CLI

The following links show how to update containers analytical TTL by using Azure CLI:

- Azure Cosmos DB API for Mongo DB
- Azure Cosmos DB SQL API

PowerShell

The following links show how to update containers analytical TTL by using PowerShell:

- Azure Cosmos DB API for Mongo DB
- Azure Cosmos DB SQL API

Optional - Disable analytical store in a SQL API container

Analytical store can be disabled in SQL API containers using Azure CLI or PowerShell.

> **NOTE**
> Please note that currently this action can't be undone. If analytical store is disabled in a container, it can never be re-enabled.

> **NOTE**
> Please note that disabling analitical store is not available for MongoDB API collections.

Azure CLI

Set `--analytical-storage-ttl` parameter to 0 using the `az cosmosdb sql container update` Azure CLI command.

PowerShell

Set `-AnalyticalStorageTtl` paramenter to 0 using the `Update-AzCosmosDBSqlContainer` PowerShell command.

Connect to a Synapse workspace

Use the instructions in Connect to Azure Synapse Link on how to access an Azure Cosmos DB database from Azure Synapse Analytics Studio with Azure Synapse Link.

Query analytical store using Apache Spark for Azure Synapse Analytics

Use the instructions in the Query Azure Cosmos DB analytical store using Spark 3 article on how to query with Synapse Spark 3. That article gives some examples on how you can interact with the analytical store from Synapse gestures. Those gestures are visible when you right-click on a container. With gestures, you can quickly generate code and tweak it to your needs. They are also perfect for discovering data with a single click.

For Spark 2 integration use the instruction in the Query Azure Cosmos DB analytical store using Spark 2 article.

Query the analytical store using serverless SQL pool in Azure Synapse Analytics

Serverless SQL pool allows you to query and analyze data in your Azure Cosmos DB containers that are enabled with Azure Synapse Link. You can analyze data in near real-time without impacting the performance of your transactional workloads. It offers a familiar T-SQL syntax to query data from the analytical store and integrated connectivity to a wide range of BI and ad-hoc querying tools via the T-SQL interface. To learn more, see the Query analytical store using serverless SQL pool article.

Use serverless SQL pool to analyze and visualize data in Power BI

You can use integrated BI experience in Azure Cosmos DB portal, to build BI dashboards using Synapse Link with just a few clicks. To learn more, see how to build BI dashboards using Synapse Link. This integrated experience

will create simple T-SQL views in Synapse serverless SQL pools, for your Cosmos DB containers. You can build BI dashboards over these views, which will query your Azure Cosmos DB containers in real-time, using Direct Query, reflecting latest changes to your data. There is no performance or cost impact to your transactional workloads, and no complexity of managing ETL pipelines.

If you want to use advance T-SQL views with joins across your containers or build BI dashboards in import] (/power-bi/connect-data/service-dataset-modes-understand#import-mode) mode, see how to use Serverless SQL pool to analyze Azure Cosmos DB data with Synapse Link article.

Configure custom partitioning

Custom partitioning enables you to partition analytical store dataonfields that are commonly used as filters inanalyticalqueriesresulting in improved query performance.To learn more, see the introduction to custom partitioning and how to configure custom partitioning articles.

Azure Resource Manager template

The Azure Resource Manager template creates a Synapse Link enabled Azure Cosmos DB account for SQL API. This template creates a Core (SQL) API account in one region with a container configured with analytical TTL enabled, and an option to use manual or autoscale throughput. To deploy this template, click on **Deploy to Azure** on the readme page.

Getting started with Azure Synapse Link - Samples

You can find samples to get started with Azure Synapse Link on GitHub. These showcase end-to-end solutions with IoT and retail scenarios. You can also find the samples corresponding to Azure Cosmos DB API for MongoDB in the same repo under the MongoDB folder.

Next steps

To learn more, see the following docs:

- Checkout the learn module on how to configure Azure Synapse Link for Azure Cosmos DB.
- Azure Cosmos DB analytical store overview.
- Frequently asked questions about Synapse Link for Azure Cosmos DB.
- Apache Spark in Azure Synapse Analytics.
- Serverless SQL pool runtime support in Azure Synapse Analytics.

Connect to Azure Synapse Link for Azure Cosmos DB

8/19/2022 • 2 minutes to read • Edit Online

This article describes how to access an Azure Cosmos DB database from Azure Synapse Analytics Studio with Azure Synapse Link.

Prerequisites

Before you connect an Azure Cosmos DB database to your workspace, you'll need an:

- Existing Azure Cosmos DB database, or create a new account by following the steps in Quickstart: Manage an Azure Cosmos DB account.
- Existing Azure Synapse workspace, or create a new workspace by following the steps in Quickstart: Create a Synapse workspace.

Enable Synapse Link on an Azure Cosmos DB database account

To run large-scale analytics into Azure Cosmos DB without affecting your operational performance, we recommend enabling Synapse Link for Azure Cosmos DB. Synapse Link brings HTAP capability to a container and built-in support in Azure Synapse.

Go to Synapse Studio

From your Azure Synapse workspace, select **Launch Synapse Studio**. On the Synapse Studio home page, select **Data**, which takes you to the Data Object Explorer.

Connect an Azure Cosmos DB database to an Azure Synapse workspace

Connecting an Azure Cosmos DB database is done as a linked service. With an Azure Cosmos DB linked service, you can browse and explore data, read, and write from Apache Spark for Azure Synapse Analytics or SQL into Azure Cosmos DB.

From the Data Object Explorer, you can directly connect to an Azure Cosmos DB database by following these steps:

1. Select the **+** icon near **Data**.
2. Select **Connect to external data**.
3. Select the API that you want to connect to, for example, **SQL API** or **API for MongoDB**.
4. Select **Continue**.
5. Use a friendly name to name the linked service. The name will appear in the Data Object Explorer and is used by Azure Synapse runtimes to connect to the database and containers.
6. Select the **Azure Cosmos DB account name** and the **database name**.
7. (Optional) If no region is specified, Azure Synapse runtime operations will be routed toward the nearest region where the analytical store is enabled. You can also manually set the region you want your users to use to access the Azure Cosmos DB analytical store. Select **Additional connection properties**, and then select **New**. Under **Property Name**, enter **PreferredRegions**. Set the **Value** to the region you want, for example, **WestUS2**. (There are no spaces between the words and the number.)

8. Select **Create**.

Azure Cosmos DB databases appear on the **Linked** tab under the **Azure Cosmos DB** section. With Azure Cosmos DB, you can differentiate an HTAP-enabled container from an OLTP-only container through the following icons:

OLTP-only container:

HTAP-enabled container:

Quickly interact with code-generated actions

By right-clicking into a container, you have a list of gestures that will trigger a Spark or SQL runtime. Writing into a container will happen through the Transactional Store of Azure Cosmos DB and will consume Request Units.

Next steps

- Learn what is supported between Azure Synapse and Azure Cosmos DB
- Configure private endpoints for Azure Cosmos DB analytical store
- Learn how to query the analytical store with Spark 3
- Learn how to query the analytical store with Spark 2

Integrated Power BI experience in Azure Cosmos DB portal for Synapse Link enabled accounts

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: SQL API

With the integrated Power BI experience, you can visualize your Azure Cosmos DB data in near real time in just a few clicks. It uses the built-in Power BI integration feature in the Azure portal along with Azure Synapse Link.

Synapse Link enables you to build Power BI dashboards with no performance or cost impact to your transactional workloads, and no ETL pipelines. With DirectQuery, you can build dashboards using live data from your Azure Cosmos DB accounts, without importing or copying the data into Power BI.

Build a Power BI report

Use the following steps to build a Power BI report from Azure Cosmos DB data in DirectQuery mode:

1. Sign into the Azure portal and navigate to your Azure Cosmos DB account.
2. From the **Integrations** section, open the **Power BI** pane and select **Get started**.

   > **NOTE**
   > Currently, this option is only available for SQL API accounts.

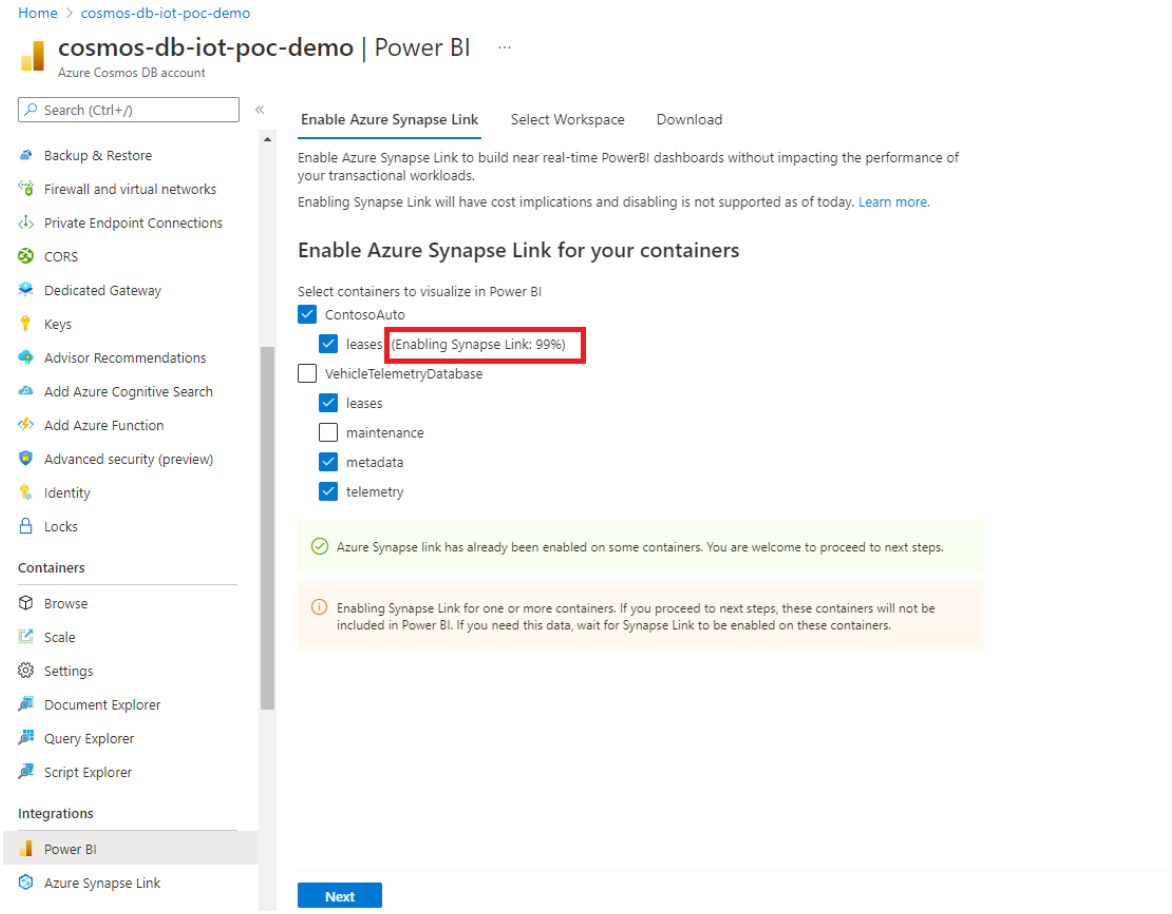
3. From the **Enable Azure Synapse Link** tab, if your account is not already enabled with Synapse Link, you can enable it from **Enable Azure Synapse link for this account** section. If Synapse Link is already enabled for your account, you will not see this tab.

   > **NOTE**
   > Once enabled, you cannot disable Azure Synapse Link. Enabling Azure Synapse Link has cost implications. See Azure Synapse Link pricing section for more details.

4. Next from the **Enable Azure Synapse Link for your containers** section, choose the required containers to enable Synapse Link.

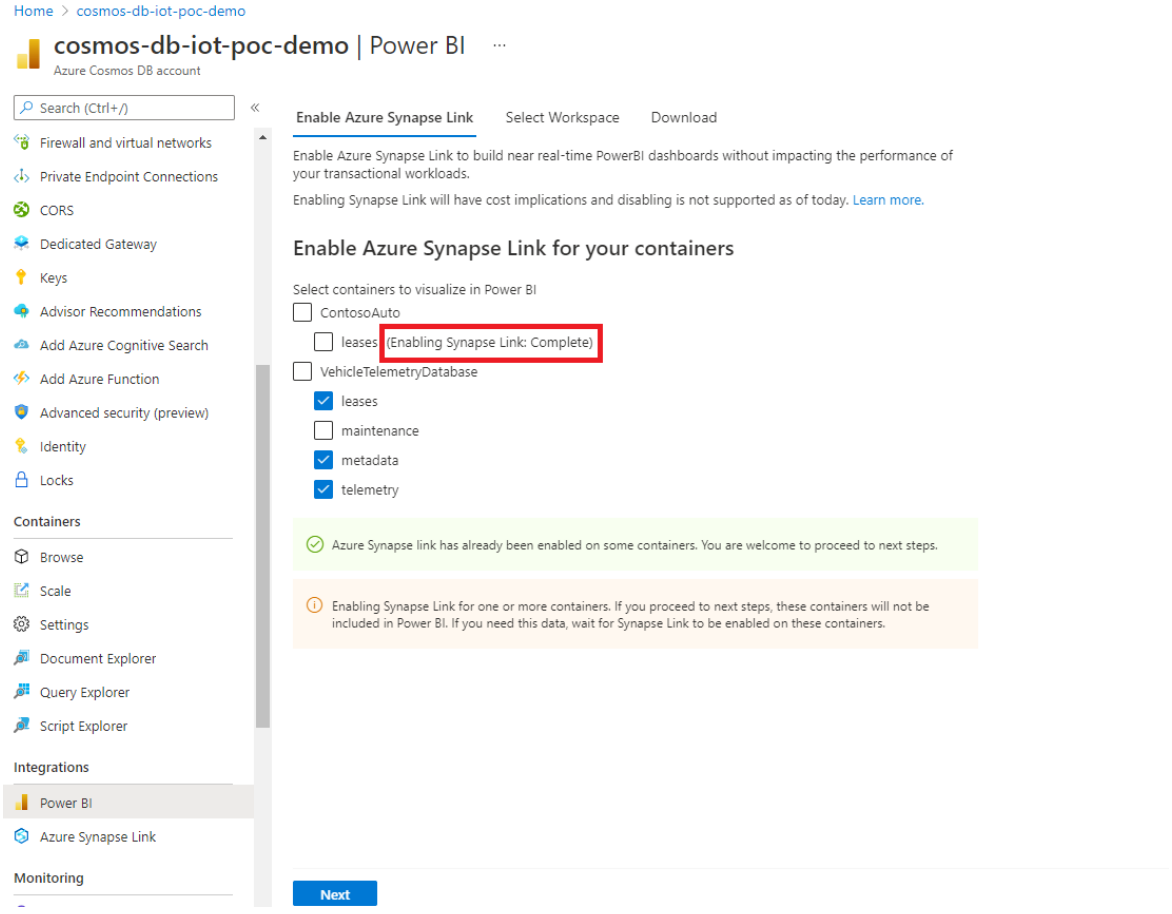
   - If you already enabled Synapse Link on some containers, you will see the checkbox next to the container name is selected. You may optionally deselect them, based on the data you'd like to visualize in Power BI.
   - If Synapse Link is not enabled, you can enable this on your existing containers.

     If enabling Synapse Link is in progress on any of the containers, the data from those containers will not be included. You should come back to this tab later and import data when the containers are enabled.



5. Depending on the amount of data in your containers, it may take a while to enable Synapse Link. To learn more, see enable Synapse Link on existing containers article.

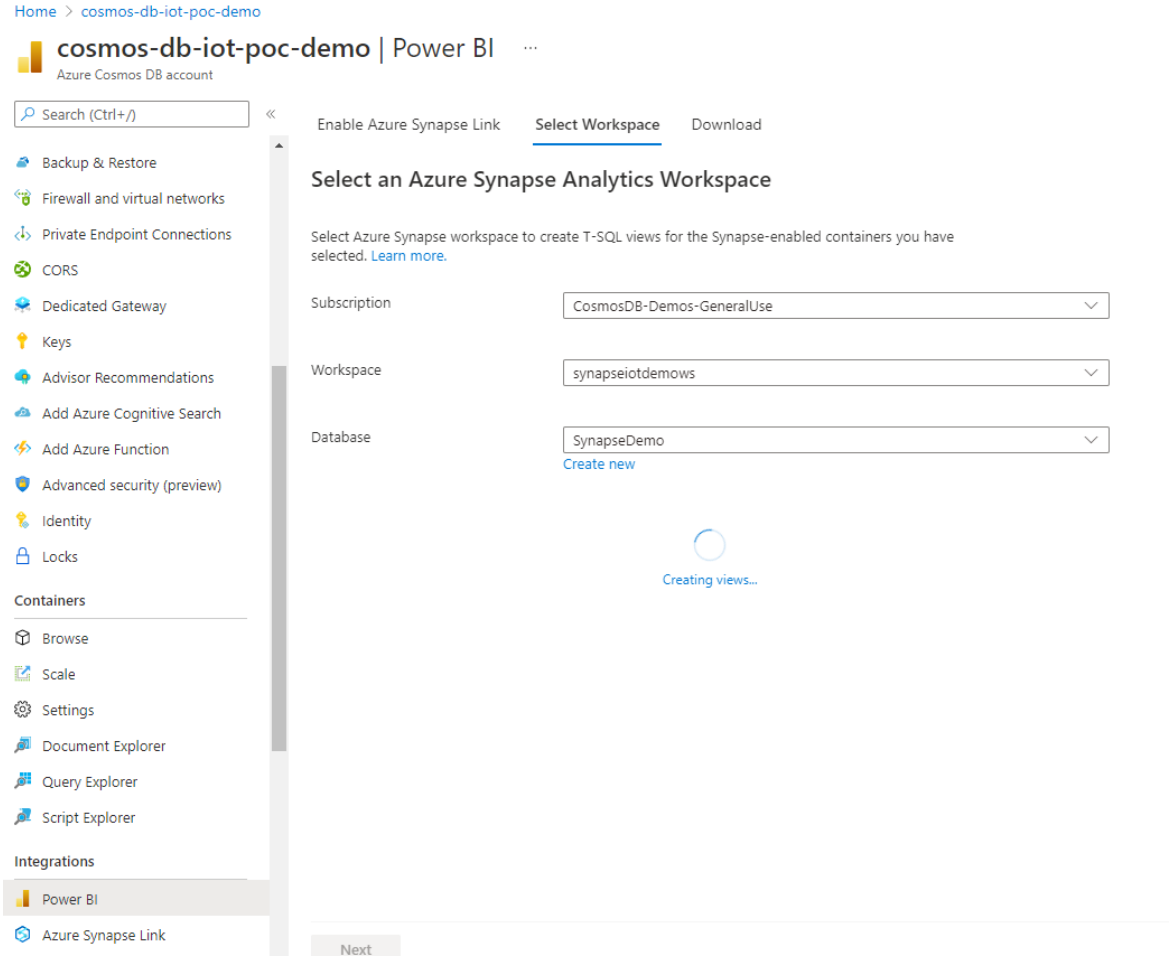
   You can check the progress in the portal as shown in the following screen. Containers are enabled with Synapse Link when the progress reaches 100%.



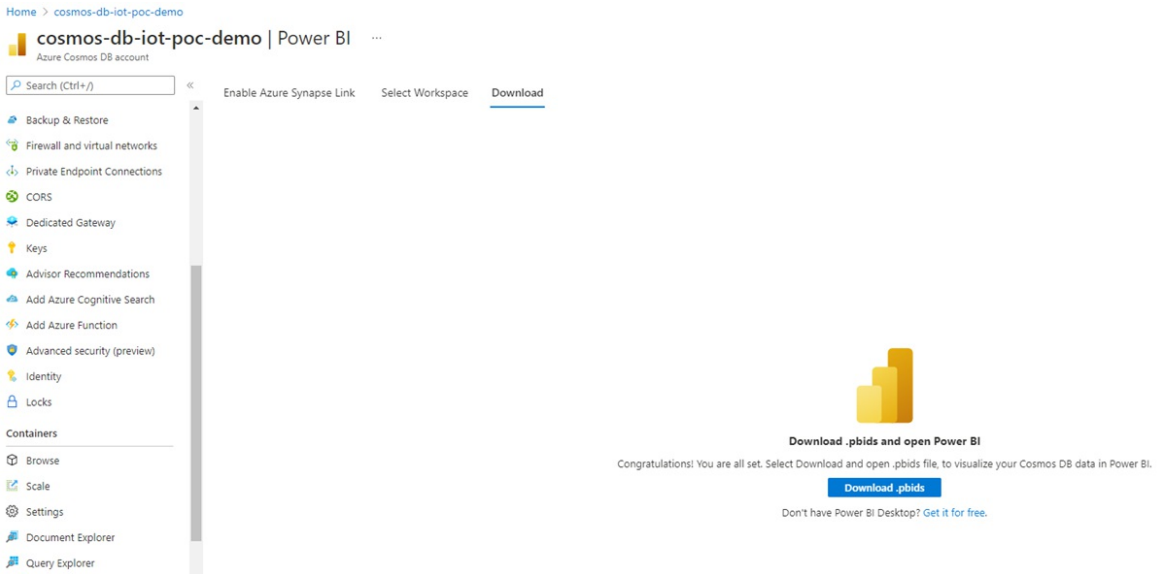
6. From the **Select workspace** tab, choose the Azure Synapse Analytics workspace and select **Next**. This will automatically create T-SQL views in Synapse Analytics, for the containers selected earlier. For more information on T-SQL views required to connect your Cosmos DB to Power BI, see Prepare views article.

> **NOTE**
>
> Your Cosmos DB container proprieties will be represented as columns in T-SQL views, including deep nested JSON data. This is a quick start for your BI dashboards. These views will be available in your Synapse workspace/database; you can also use these exact same views in Synapse Workspace for data exploration, data science, data engineering, etc. Please note that advanced scenarios may demand more complex views or fine tuning of these views, for better performance. For more information. see best practices for Synapse Link when using Synapse serverless SQL pools article.

7. You can either choose an existing workspace or create a new one. To select an existing workspace, provide the **Subscription**, **Workspace**, and the **Database** details. Azure portal will use your Azure AD credentials to automatically connect to your Synapse workspace and create T-SQL views. Make sure you have "Synapse administrator" permissions to this workspace.

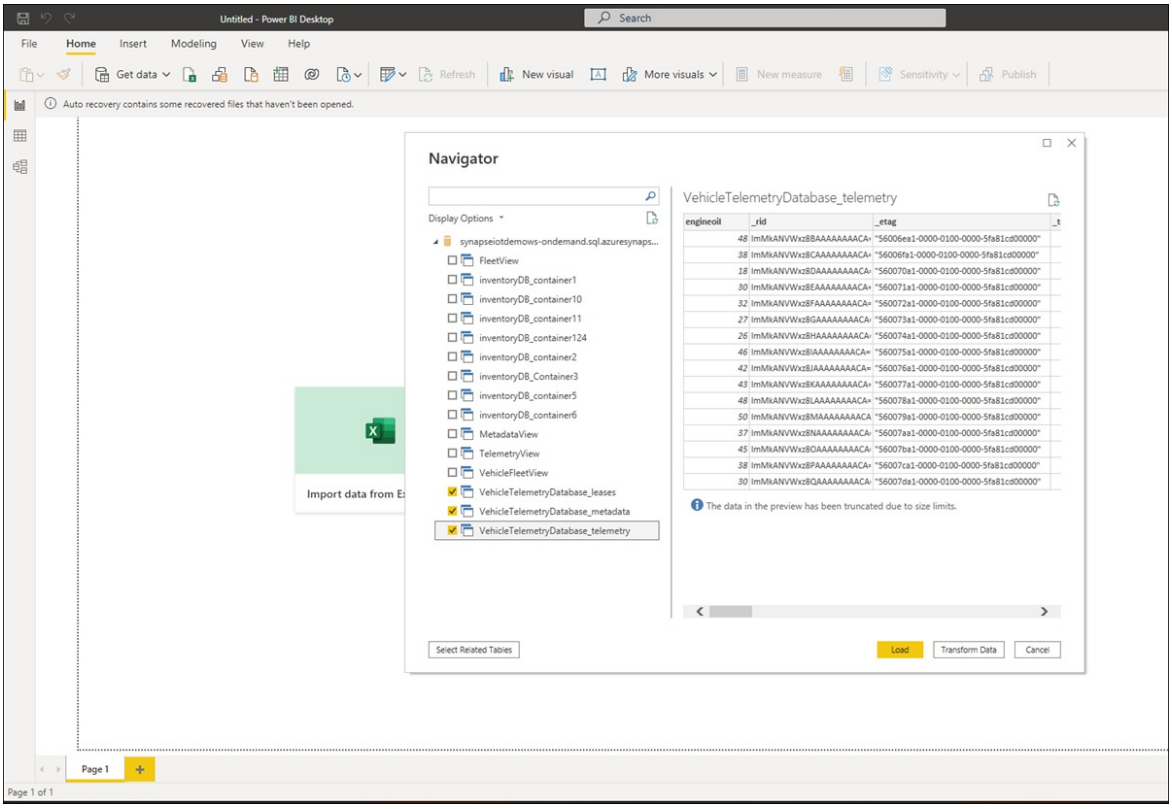


8. Next, select **Download .pbids** to download the Power BI data source file. Open the downloaded file. It contains the required connection information and opens a Power BI dashboard.



9. You can now connect to Azure Cosmos DB data from Power BI, using Synapse Link. A list of T-SQL views corresponding to the data in each container are displayed.

   For example, the following screen shows vehicle fleet data. You can load this data for further analysis or transform it before loading.



10. You can now start building the dashboard using Azure Cosmos DB's analytical data. Any changes to your data will be reflected in the dashboard, as soon as the data is replicated to analytical store, which typically happens in a couple of minutes.

Next steps

- Connect serverless SQL pool to Power BI Desktop & create report
- Use Power BI and serverless Synapse SQL pool to analyze Azure Cosmos DB data with Synapse Link

Use Power BI and serverless Synapse SQL pool to analyze Azure Cosmos DB data with Synapse Link

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: SQL API Azure Cosmos DB API for MongoDB

In this article, you learn how to build a serverless SQL pool database and views over Synapse Link for Azure Cosmos DB. You will query the Azure Cosmos DB containers and then build a model with Power BI over those views to reflect that query.

With Azure Synapse Link, you can build near real-time dashboards in Power BI to analyze your Azure Cosmos DB data. There is no performance or cost impact to your transactional workloads, and no complexity of managing ETL pipelines. You can use either DirectQuery or import modes.

> **NOTE**
> You can build Power BI dashboards with just a few clicks using Azure Cosmos DB portal. For more information, see Integrated Power BI experience in Azure Cosmos DB portal for Synapse Link enabled accounts. This will automatically create T-SQL views in Synapse serverless SQL pools on your Cosmos DB containers. You can simply download the .pbids file that connects to these T-SQL views to start building your BI dashboards.

In this scenario, you will use dummy data about Surface product sales in a partner retail store. You will analyze the revenue per store based on the proximity to large households and the impact of advertising for a specific week. In this article, you create two views named **RetailSales** and **StoreDemographics** and a query between them. You can get the sample product data from this GitHub repo.

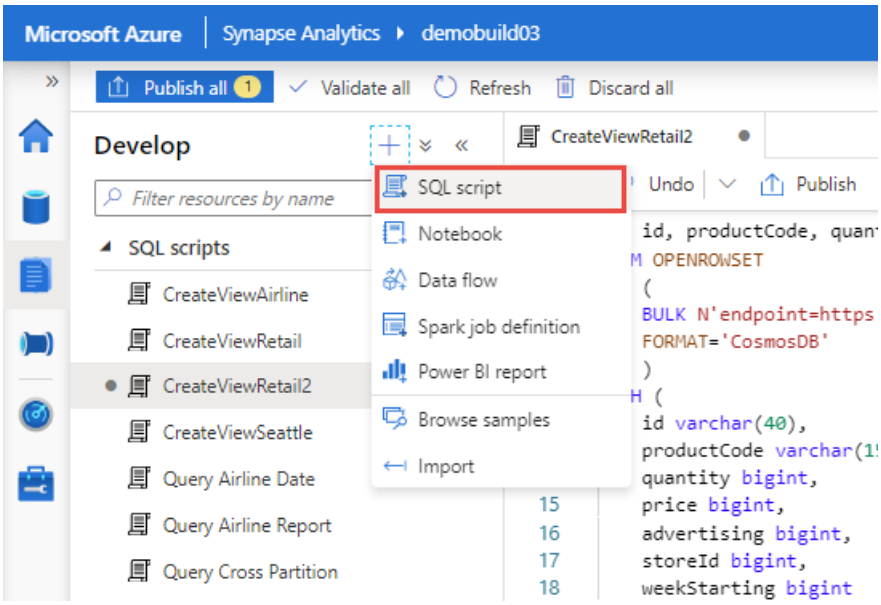
Prerequisites

Make sure to create the following resources before you start:

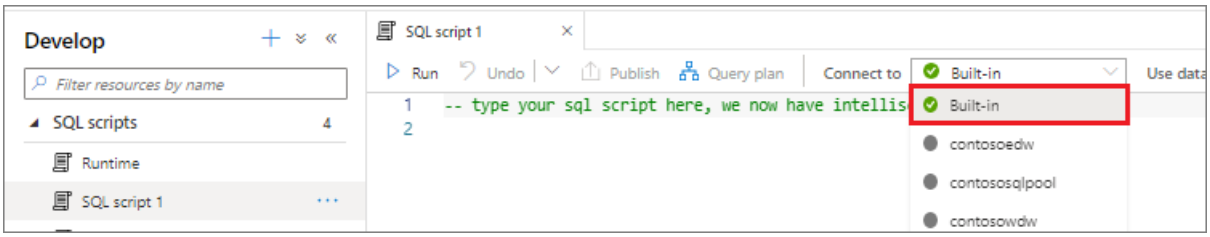
- Create an Azure Cosmos DB account of kind SQL(core) or MongoDB.
- Enable Azure Synapse Link for your Azure Cosmos account
- Create a database within the Azure Cosmos account and two containers that have analytical store enabled.
- Load products data into the Azure Cosmos containers as described in this batch data ingestion notebook.
- Create a Synapse workspace named **SynapseLinkBI**.
- Connect the Azure Cosmos database to the Synapse workspace.

Create a database and views

From the Synapse workspace go the **Develop** tab, select the **+** icon, and select **SQL Script**.



Every workspace comes with a serverless SQL endpoint. After creating a SQL script, from the tool bar on the top connect to **Built-in**.



Creating views in the **master** or **default** databases is not recommended or supported. Create a new database, named **RetailCosmosDB**, and a SQL view over the Synapse Link enabled containers. The following command shows how to create a database:

```
--Createdatabase
CreatedatabaseRetailCosmosDB
```

Next, create multiple views across different Synapse Link enabled Azure Cosmos containers. Views will allow you to use T-SQL to join and query Azure Cosmos DB data sitting in different containers. Make sure to select the **RetailCosmosDB** database when creating the views.

The following scripts show how to create views on each container. For simplicity, let's use the automatic schema inference feature of serverless SQL pool over Synapse Link enabled containers:

RetailSales view:

```
--Create view for RetailSales container
CREATEVIEW RetailSales
AS
SELECT  *
FROM OPENROWSET (
    'CosmosDB', N'account=<Your Azure Cosmos account name>;database=<Your Azure Cosmos database
name>;region=<Your Azure Cosmos DB Region>;key=<Your Azure Cosmos DB key here>',RetailSales)
AS q1
```

Make sure to insert your Azure Cosmos DB region and the primary key in the previous SQL script. All the characters in the region name should be in lower case without spaces. Unlike the other parameters of the `OPENROWSET` command, the container name parameter should be specified without quotes around it.

StoreDemographics view:

```
--Create view for StoreDemographics container
CREATEVIEWStoreDemographics
AS
SELECT  *
FROM OPENROWSET (
    'CosmosDB', N'account=<Your Azure Cosmos account name>;database=<Your Azure Cosmos database
name>;region=<Your Azure Cosmos DB Region>;key=<Your Azure Cosmos DB key here>', StoreDemographics)
AS q1
```

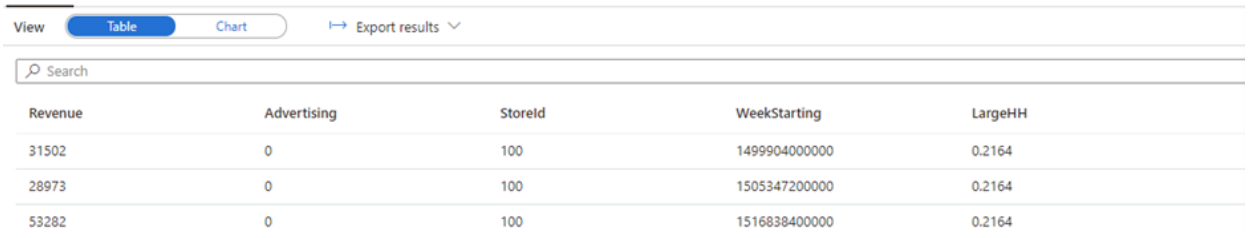
Now run the SQL script by selecting the **Run** command.

Query the views

Now that the two views are created, let's define the query to join those two views as follows:

```
SELECT
sum(p.[revenue]) as revenue
,p.[advertising]
,p.[storeId]
,p.[weekStarting]
,q.[largeHH]
 FROM [dbo].[RetailSales] as p
INNER JOIN [dbo].[StoreDemographics] as q ON q.[storeId] = p.[storeId]
GROUP BY p.[advertising], p.[storeId], p.[weekStarting], q.[largeHH]
```

Select **Run** that gives the following table as result:



Revenue	Advertising	StoreId	WeekStarting	LargeHH
31502	0	100	1499904000000	0.2164
28973	0	100	1505347200000	0.2164
53282	0	100	1516838400000	0.2164

Model views over containers with Power BI

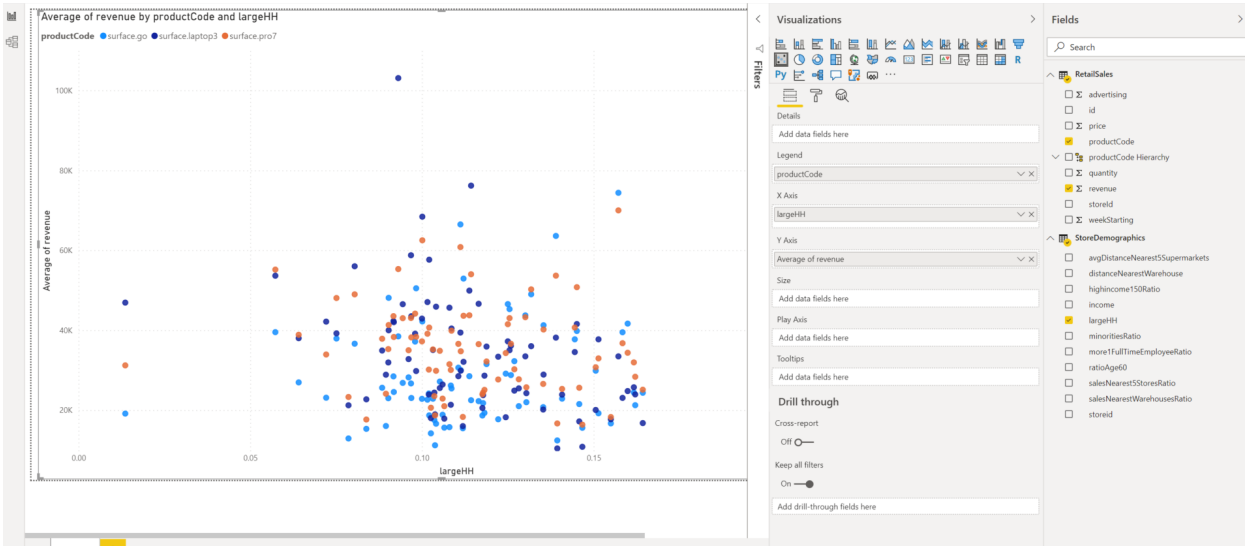
Next open the Power BI desktop and connect to the serverless SQL endpoint by using the following steps:

1. Open the Power BI Desktop application. Select **Get data** and select **more**.
2. Choose **Azure Synapse Analytics (SQL DW)** from the list of connection options.
3. Enter the name of the SQL endpoint where the database is located. Enter `SynapseLinkBI-ondemand.sql.azuresynapse.net` within the **Server** field. In this example, **SynapseLinkBI** is name of the workspace. Replace it if you have given a different name to your workspace. Select **Direct Query** for data connectivity mode and then **OK**.
4. Select the preferred authentication method such as Azure AD.
5. Select the **RetailCosmosDB** database and the **RetailSales**, **StoreDemographics** views.
6. Select **Load** to load the two views into the direct query mode.
7. Select **Model** to create a relationship between the two views through the **storeId** column.
8. Drag the **StoreId** column from the **RetailSales** view towards the **StoreId** column in the **StoreDemographics** view.
9. Select the Many to one (*:1) relationship because there are multiple rows with the same store ID in the

RetailSales view. **StoreDemographics** has only one store ID row (it is a dimension table).

Now navigate to the **report** window and create a report to compare the relative importance of household size to the average revenue per store based on the scattered representation of revenue and LargeHH index:

1. Select **Scatter chart**.
2. Drag and drop **LargeHH** from the **StoreDemographics** view into the X-axis.
3. Drag and drop **Revenue** from **RetailSales** view into the Y-axis. Select **Average** to get the average sales per product per store and per week.
4. Drag and drop the **productCode** from **RetailSales** view into the legend to select a specific product line. After you choose these options, you should see a graph like the following screenshot:



Next steps

Integrated Power BI experience in Azure Cosmos DB portal for Synapse Link enabled accounts

Use T-SQL to query Azure Cosmos DB data using Azure Synapse Link

Use serverless SQL pool to analyze Azure Open Datasets and visualize the results in Azure Synapse Studio

Interact with Azure Cosmos DB using Apache Spark 3 in Azure Synapse Link

8/19/2022 • 7 minutes to read • Edit Online

In this article, you'll learn how to interact with Azure Cosmos DB using Synapse Apache Spark 3. With its full support for Scala, Python, SparkSQL, and C#, Synapse Apache Spark 3 is central to analytics, data engineering, data science, and data exploration scenarios in Azure Synapse Link for Azure Cosmos DB.

The following capabilities are supported while interacting with Azure Cosmos DB:

- Synapse Apache Spark 3 allows you to analyze data in your Azure Cosmos DB containers that are enabled with Azure Synapse Link in near real-time without impacting the performance of your transactional workloads. The following two options are available to query the Azure Cosmos DB analytical store from Spark:
  - Load to Spark DataFrame
  - Create Spark table
- Synapse Apache Spark also allows you to ingest data into Azure Cosmos DB. It is important to note that data is always ingested into Azure Cosmos DB containers through the transactional store. When Synapse Link is enabled, any new inserts, updates, and deletes are then automatically synced to the analytical store.
- Synapse Apache Spark also supports Spark structured streaming with Azure Cosmos DB as a source as well as a sink.

The following sections walk you through the syntax of above capabilities. You can also checkout the learn module on how to Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics. Gestures in Azure Synapse Analytics workspace are designed to provide an easy out-of-the-box experience to get started. Gestures are visible when you right-click on an Azure Cosmos DB container in the **Data** tab of the Synapse workspace. With gestures, you can quickly generate code and tailor it to your needs. Gestures are also perfect for discovering data with a single click.

> **IMPORTANT**
>
> You should be aware of some constraints in the analytical schema that could lead to the unexpected behavior in data loading operations. As an example, only first 1000 properties from transactional schema are available in the analytical schema, properties with spaces are not available, etc. If you are experiencing some unexpected results, check the analytical store schema constraints for more details.

Query Azure Cosmos DB analytical store

Before you learn about the two possible options to query Azure Cosmos DB analytical store, loading to Spark DataFrame and creating Spark table, it is worth exploring the differences in experience so you can choose the option that works for your needs.

The difference in experience is around whether underlying data changes in the Azure Cosmos DB container should be automatically reflected in the analysis performed in Spark. When either a Spark DataFrame is registered or a Spark table is created against a container's analytical store, metadata around the current snapshot of data in the analytical store is fetched to Spark for efficient pushdown of subsequent analysis. It is important to note that since Spark follows a lazy evaluation policy, unless an action is invoked on the Spark DataFrame or a SparkSQL query is executed against the Spark table, actual data is not fetched from the underlying container's analytical store.

In the case of **loading to Spark DataFrame**, the fetched metadata is cached through the lifetime of the Spark session and hence subsequent actions invoked on the DataFrame are evaluated against the snapshot of the analytical store at the time of DataFrame creation.

On the other hand, in the case of **creating a Spark table**, the metadata of the analytical store state is not cached in Spark and is reloaded on every SparkSQL query execution against the Spark table.

Thus, you can choose between loading to Spark DataFrame and creating a Spark table based on whether you want your Spark analysis to be evaluated against a fixed snapshot of the analytical store or against the latest snapshot of the analytical store respectively.

> **NOTE**
>
> To query the Azure Cosmos DB API of Mongo DB accounts, learn more about the full fidelity schema representation in the analytical store and the extended property names to be used.

> **NOTE**
>
> Please note that all `options` in the commands below are case sensitive.

Load to Spark DataFrame

In this example, you'll create a Spark DataFrame that points to the Azure Cosmos DB analytical store. You can then perform additional analysis by invoking Spark actions against the DataFrame. This operation doesn't impact the transactional store.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

df = spark.read.format("cosmos.olap")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .load()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

val df_olap = spark.read.format("cosmos.olap").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    load()
```

Create Spark table

In this example, you'll create a Spark table that points the Azure Cosmos DB analytical store. You can then perform additional analysis by invoking SparkSQL queries against the table. This operation neither impacts the transactional store nor does it incur any data movement. If you decide to delete this Spark table, the underlying Azure Cosmos DB container and the corresponding analytical store will not be affected.

This scenario is convenient to reuse Spark tables through third-party tools and provide accessibility to the underlying data for the run-time.

The syntax to create a Spark table is as follows:

```
%%sql
-- To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
spark.cosmos.preferredRegions '<Region1>,<Region2>' in the config options

create table call_center using cosmos.olap options (
    spark.synapse.linkedService '<enter linked service name>',
    spark.cosmos.container '<enter container name>'
)
```

NOTE

If you have scenarios where the schema of the underlying Azure Cosmos DB container changes over time; and if you want the updated schema to automatically reflect in the queries against the Spark table, you can achieve this by setting the `spark.cosmos.autoSchemaMerge` option to `true` in the Spark table options.

Write Spark DataFrame to Azure Cosmos DB container

In this example, you'll write a Spark DataFrame into an Azure Cosmos DB container. This operation will impact the performance of transactional workloads and consume request units provisioned on the Azure Cosmos DB container or the shared database.

The syntax in **Python** would be the following:

```
# Write a Spark DataFrame into an Azure Cosmos DB container
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

YOURDATAFRAME.write.format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .mode('append')\
    .save()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

import org.apache.spark.sql.SaveMode

df.write.format("cosmos.oltp").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    mode(SaveMode.Append).
    save()
```

Load streaming DataFrame from container

In this gesture, you'll use Spark Streaming capability to load data from a container into a dataframe. The data will be stored in the primary data lake account (and file system) you connected to the workspace.

> **NOTE**
>
> If you are looking to reference external libraries in Synapse Apache Spark, learn more here. For instance, if you are looking to ingest a Spark DataFrame to a container of Cosmos DB API for Mongo DB, you can leverage the Mongo DB connector for Spark here.

Load streaming DataFrame from Azure Cosmos DB container

In this example, you'll use Spark's structured streaming capability to load data from an Azure Cosmos DB container into a Spark streaming DataFrame using the change feed functionality in Azure Cosmos DB. The checkpoint data used by Spark will be stored in the primary data lake account (and file system) that you connected to the workspace.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

dfStream = spark.readStream\
    .format("cosmos.oltp.changeFeed")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .option("spark.cosmos.changeFeed.startFrom", "Beginning")\
    .option("spark.cosmos.changeFeed.mode", "Incremental")\
    .load()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

val dfStream = spark.readStream.
    format("cosmos.oltp.changeFeed").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    option("spark.cosmos.changeFeed.startFrom", "Beginning").
    option("spark.cosmos.changeFeed.mode", "Incremental").
    load()
```

Write streaming DataFrame to Azure Cosmos DB container

In this example, you'll write a streaming DataFrame into an Azure Cosmos DB container. This operation will impact the performance of transactional workloads and consume Request Units provisioned on the Azure Cosmos DB container or shared database. If the folder */localWriteCheckpointFolder* isn't created (in the example below), it will be automatically created.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

streamQuery = dfStream\
    .writeStream\
    .format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .option("checkpointLocation", "/tmp/myRunId/")\
    .outputMode("append")\
    .start()

streamQuery.awaitTermination()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

val query = dfStream.
            writeStream.
            format("cosmos.oltp").
            outputMode("append").
            option("spark.synapse.linkedService", "<enter linked service name>").
            option("spark.cosmos.container", "<enter container name>").
            option("checkpointLocation", "/tmp/myRunId/").
            start()

query.awaitTermination()
```

Next steps

- Samples to get started with Azure Synapse Link on GitHub
- Learn what is supported in Azure Synapse Link for Azure Cosmos DB
- Connect to Synapse Link for Azure Cosmos DB
- Checkout the learn module on how to Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics.

Interact with Azure Cosmos DB using Apache Spark 2 in Azure Synapse Link

8/19/2022 • 9 minutes to read • Edit Online

> **NOTE**
> For Synapse Link for Cosmos DB using Spark 3, refer to this article Azure Synapse Link for Azure Cosmos DB on Spark 3

In this article, you'll learn how to interact with Azure Cosmos DB using Synapse Apache Spark 2. With its full support for Scala, Python, SparkSQL, and C#, Synapse Apache Spark is central to analytics, data engineering, data science, and data exploration scenarios in Azure Synapse Link for Azure Cosmos DB.

The following capabilities are supported while interacting with Azure Cosmos DB:

- Synapse Apache Spark allows you to analyze data in your Azure Cosmos DB containers that are enabled with Azure Synapse Link in near real-time without impacting the performance of your transactional workloads. The following two options are available to query the Azure Cosmos DB analytical store from Spark:
  - Load to Spark DataFrame
  - Create Spark table
- Synapse Apache Spark also allows you to ingest data into Azure Cosmos DB. It is important to note that data is always ingested into Azure Cosmos DB containers through the transactional store. When Synapse Link is enabled, any new inserts, updates, and deletes are then automatically synced to the analytical store.
- Synapse Apache Spark also supports Spark structured streaming with Azure Cosmos DB as a source as well as a sink.

The following sections walk you through the syntax of above capabilities. You can also checkout the learn module on how to Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics. Gestures in Azure Synapse Analytics workspace are designed to provide an easy out-of-the-box experience to get started. Gestures are visible when you right-click on an Azure Cosmos DB container in the **Data** tab of the Synapse workspace. With gestures, you can quickly generate code and tailor it to your needs. Gestures are also perfect for discovering data with a single click.

> **IMPORTANT**
> You should be aware of some constraints in the analytical schema that could lead to the unexpected behavior in data loading operations. As an example, only first 1000 properties from transactional schema are available in the analytical schema, properties with spaces are not available, etc. If you are experiencing some unexpected results, check the analytical store schema constraints for more details.

Query Azure Cosmos DB analytical store

Before you learn about the two possible options to query Azure Cosmos DB analytical store, loading to Spark DataFrame and creating Spark table, it is worth exploring the differences in experience so you can choose the option that works for your needs.

The difference in experience is around whether underlying data changes in the Azure Cosmos DB container should be automatically reflected in the analysis performed in Spark. When either a Spark DataFrame is registered or a Spark table is created against a container's analytical store, metadata around the current

snapshot of data in the analytical store is fetched to Spark for efficient pushdown of subsequent analysis. It is important to note that since Spark follows a lazy evaluation policy, unless an action is invoked on the Spark DataFrame or a SparkSQL query is executed against the Spark table, actual data is not fetched from the underlying container's analytical store.

In the case of **loading to Spark DataFrame**, the fetched metadata is cached through the lifetime of the Spark session and hence subsequent actions invoked on the DataFrame are evaluated against the snapshot of the analytical store at the time of DataFrame creation.

On the other hand, in the case of **creating a Spark table**, the metadata of the analytical store state is not cached in Spark and is reloaded on every SparkSQL query execution against the Spark table.

Thus, you can choose between loading to Spark DataFrame and creating a Spark table based on whether you want your Spark analysis to be evaluated against a fixed snapshot of the analytical store or against the latest snapshot of the analytical store respectively.

If your analytical queries have frequently used filters, you have the option to partition based on these fields for better query performance. You can periodically execute partitioning job from an Azure Synapse Spark notebook, to trigger partitioning on analytical store. This partitioned store points to the ADLS Gen2 primary storage account that is linked to your Azure Synapse workspace. To learn more, see the introduction to custom partitioning and how to configure custom partitioning articles.

> **NOTE**
> To query the Azure Cosmos DB API of Mongo DB accounts, learn more about the full fidelity schema representation in the analytical store and the extended property names to be used.

> **NOTE**
> Please note that all `options` in the commands below are case sensitive. For example, you must use `Gateway` while `gateway` will return an error.

Load to Spark DataFrame

In this example, you'll create a Spark DataFrame that points to the Azure Cosmos DB analytical store. You can then perform additional analysis by invoking Spark actions against the DataFrame. This operation doesn't impact the transactional store.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

df = spark.read.format("cosmos.olap")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .load()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

val df_olap = spark.read.format("cosmos.olap").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    load()
```

Create Spark table

In this example, you'll create a Spark table that points the Azure Cosmos DB analytical store. You can then perform additional analysis by invoking SparkSQL queries against the table. This operation neither impacts the transactional store nor does it incur any data movement. If you decide to delete this Spark table, the underlying Azure Cosmos DB container and the corresponding analytical store will not be affected.

This scenario is convenient to reuse Spark tables through third-party tools and provide accessibility to the underlying data for the run-time.

The syntax to create a Spark table is as follows:

```
%%sql
-- To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
spark.cosmos.preferredRegions '<Region1>,<Region2>' in the config options

create table call_center using cosmos.olap options (
    spark.synapse.linkedService '<enter linked service name>',
    spark.cosmos.container '<enter container name>'
)
```

> **NOTE**
>
> If you have scenarios where the schema of the underlying Azure Cosmos DB container changes over time; and if you want the updated schema to automatically reflect in the queries against the Spark table, you can achieve this by setting the `spark.cosmos.autoSchemaMerge` option to `true` in the Spark table options.

Write Spark DataFrame to Azure Cosmos DB container

In this example, you'll write a Spark DataFrame into an Azure Cosmos DB container. This operation will impact the performance of transactional workloads and consume request units provisioned on the Azure Cosmos DB container or the shared database.

The syntax in **Python** would be the following:

```
# Write a Spark DataFrame into an Azure Cosmos DB container
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

YOURDATAFRAME.write.format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .option("spark.cosmos.write.upsertEnabled", "true")\
    .mode('append')\
    .save()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

import org.apache.spark.sql.SaveMode

df.write.format("cosmos.oltp").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    option("spark.cosmos.write.upsertEnabled", "true").
    mode(SaveMode.Overwrite).
    save()
```

Load streaming DataFrame from container

In this gesture, you'll use Spark Streaming capability to load data from a container into a dataframe. The data will be stored in the primary data lake account (and file system) you connected to the workspace.

> **NOTE**
> If you are looking to reference external libraries in Synapse Apache Spark, learn more here. For instance, if you are looking to ingest a Spark DataFrame to a container of Cosmos DB API for Mongo DB, you can leverage the Mongo DB connector for Spark here.

Load streaming DataFrame from Azure Cosmos DB container

In this example, you'll use Spark's structured streaming capability to load data from an Azure Cosmos DB container into a Spark streaming DataFrame using the change feed functionality in Azure Cosmos DB. The checkpoint data used by Spark will be stored in the primary data lake account (and file system) that you connected to the workspace.

If the folder */localReadCheckpointFolder* isn't created (in the example below), it will be automatically created. This operation will impact the performance of transactional workloads and consume Request Units provisioned on the Azure Cosmos DB container or shared database.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

dfStream = spark.readStream\
    .format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .option("spark.cosmos.changeFeed.readEnabled", "true")\
    .option("spark.cosmos.changeFeed.startFromTheBeginning", "true")\
    .option("spark.cosmos.changeFeed.checkpointLocation", "/localReadCheckpointFolder")\
    .option("spark.cosmos.changeFeed.queryName", "streamQuery")\
    .load()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

val dfStream = spark.readStream.
    format("cosmos.oltp").
    option("spark.synapse.linkedService", "<enter linked service name>").
    option("spark.cosmos.container", "<enter container name>").
    option("spark.cosmos.changeFeed.readEnabled", "true").
    option("spark.cosmos.changeFeed.startFromTheBeginning", "true").
    option("spark.cosmos.changeFeed.checkpointLocation", "/localReadCheckpointFolder").
    option("spark.cosmos.changeFeed.queryName", "streamQuery").
    load()
```

Write streaming DataFrame to Azure Cosmos DB container

In this example, you'll write a streaming DataFrame into an Azure Cosmos DB container. This operation will impact the performance of transactional workloads and consume Request Units provisioned on the Azure Cosmos DB container or shared database. If the folder */localWriteCheckpointFolder* isn't created (in the example below), it will be automatically created.

The syntax in **Python** would be the following:

```
# To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

# If you are using managed private endpoints for Azure Cosmos DB analytical store and using batch
writes/reads and/or streaming writes/reads to transactional store you should set connectionMode to Gateway.

def writeBatchToCosmos(batchDF, batchId):
  batchDF.persist()
  print("--> BatchId: {}, Document count: {} : {}".format(batchId, batchDF.count(),
datetime.datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S.%f")))
  batchDF.write.format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "<enter linked service name>")\
    .option("spark.cosmos.container", "<enter container name>")\
    .option("spark.cosmos.write.upsertEnabled", "true")\
    .mode('append')\
    .save()
  print("<-- BatchId: {}, Document count: {} : {}".format(batchId, batchDF.count(),
datetime.datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S.%f")))
  batchDF.unpersist()

streamQuery = dfStream\
        .writeStream\
        .foreachBatch(writeBatchToCosmos) \
        .option("checkpointLocation", "/localWriteCheckpointFolder")\
        .start()

streamQuery.awaitTermination()
```

The equivalent syntax in **Scala** would be the following:

```
// To select a preferred list of regions in a multi-region Azure Cosmos DB account, add
.option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

// If you are using managed private endpoints for Azure Cosmos DB analytical store and using batch
writes/reads and/or streaming writes/reads to transactional store you should set connectionMode to Gateway.

val query = dfStream.
            writeStream.
            foreachBatch { (batchDF: DataFrame, batchId: Long) =>
              batchDF.persist()
              batchDF.write.format("cosmos.oltp").
                option("spark.synapse.linkedService", "<enter linked service name>").
                option("spark.cosmos.container", "<enter container name>").
                option("spark.cosmos.write.upsertEnabled", "true").
                mode(SaveMode.Overwrite).
                save()
              println(s"BatchId: $batchId, Document count: ${batchDF.count()}")
              batchDF.unpersist()
              ()
            }.
            option("checkpointLocation", "/localWriteCheckpointFolder").
            start()

query.awaitTermination()
```

Next steps

- Samples to get started with Azure Synapse Link on GitHub
- Learn what is supported in Azure Synapse Link for Azure Cosmos DB
- Connect to Synapse Link for Azure Cosmos DB
- Checkout the learn module on how to Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics.

Copy data from Azure Cosmos DB into a dedicated SQL pool using Apache Spark

8/19/2022 • 2 minutes to read • Edit Online

Azure Synapse Link for Azure Cosmos DB enables users to run near real-time analytics over operational data in Azure Cosmos DB. However, there are times when some data needs to be aggregated and enriched to serve data warehouse users. Curating and exporting Azure Synapse Link data can be done with just a few cells in a notebook.

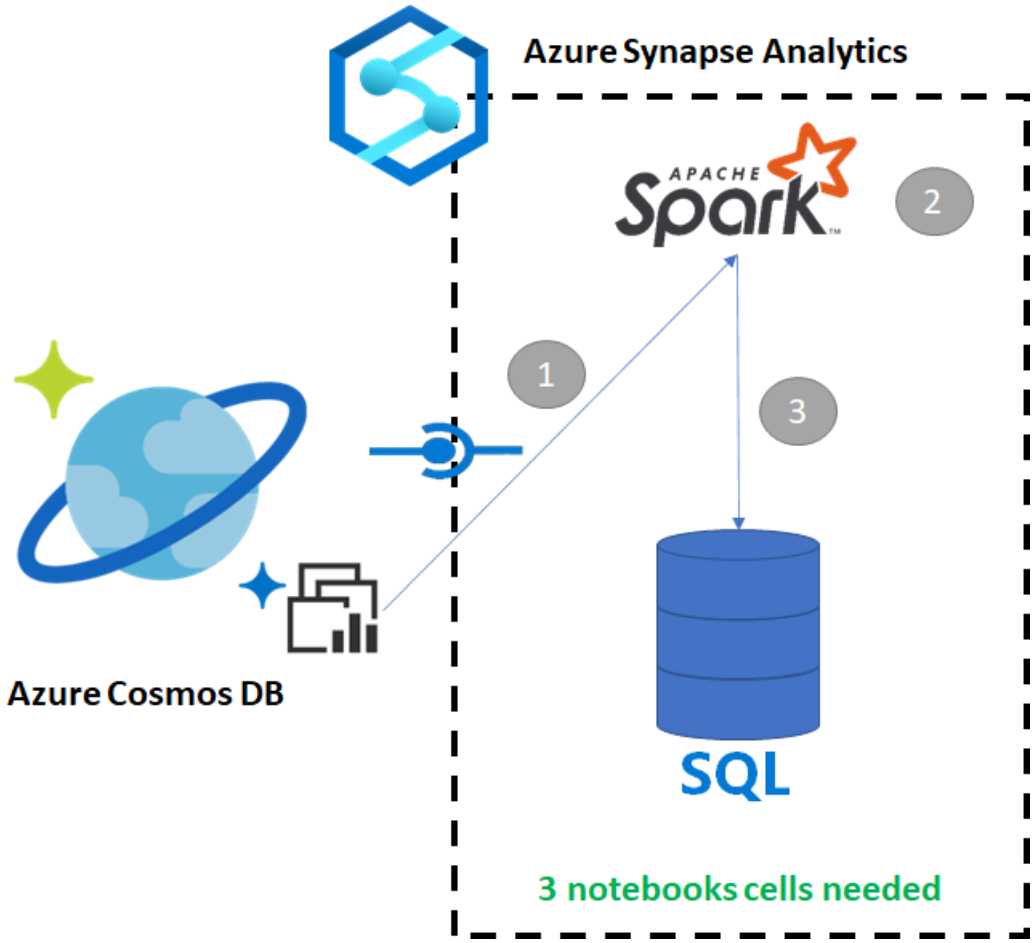
Prerequisites

- Provision a Synapse workspace with:
  - Serverless Apache Spark pool
  - dedicated SQL pool
- Provision a Cosmos DB account with a HTAP container with data
- Connect the Azure Cosmos DB HTAP container to the workspace
- Have the right setup to import data into a dedicated SQL pool from Spark

Steps

In this tutorial, you'll connect to the analytical store so there's no impact on the transactional store (it won't consume any Request Units). We'll go through the following steps:

1. Read the Cosmos DB HTAP container into a Spark dataframe
2. Aggregate the results in a new dataframe
3. Ingest the data into a dedicated SQL pool



Data

In that example, we use an HTAP container called **RetailSales**. It's part of a linked service called **ConnectedData**, and has the following schema:

- _rid: string (nullable = true)
- _ts: long (nullable = true)
- logQuantity: double (nullable = true)
- productCode: string (nullable = true)
- quantity: long (nullable = true)
- price: long (nullable = true)
- id: string (nullable = true)
- advertising: long (nullable = true)
- storeId: long (nullable = true)
- weekStarting: long (nullable = true)
- _etag: string (nullable = true)

We'll aggregate the sales (*quantity*, *revenue* (price x quantity) by *productCode* and *weekStarting* for reporting purposes. Finally, we'll export that data into a dedicated SQL pool table called `dbo.productsales`.

Configure a Spark Notebook

Create a Spark notebook with Scala as Spark (Scala) as the main language. We use the notebook's default setting for the session.

Read the data in Spark

Read the Cosmos DB HTAP container with Spark into a dataframe in the first cell.

```
val df_olap = spark.read.format("cosmos.olap").
    option("spark.synapse.linkedService", "ConnectedData").
    option("spark.cosmos.container", "RetailSales").
    load()
```

Aggregate the results in a new dataframe

In the second cell, we run the transformation and aggregates needed for the new dataframe before loading it into a dedicated SQL pool database.

```
// Select relevant columns and create revenue
val df_olap_step1 =
df_olap.select("productCode","weekStarting","quantity","price").withColumn("revenue",col("quantity")*col("pr
ice"))
//Aggregate revenue, quantity sold and avg. price by week and product ID
val df_olap_aggr = df_olap_step1.groupBy("productCode","weekStarting").agg(sum("quantity") as
"Sum_quantity",sum("revenue") as "Sum_revenue").
    withColumn("AvgPrice",col("Sum_revenue")/col("Sum_quantity"))
```

Load the results into a dedicated SQL pool

In the third cell, we load the data into a dedicated SQL pool. It will automatically create a temporary external table, external data source, and external file format that will be deleted once the job is done.

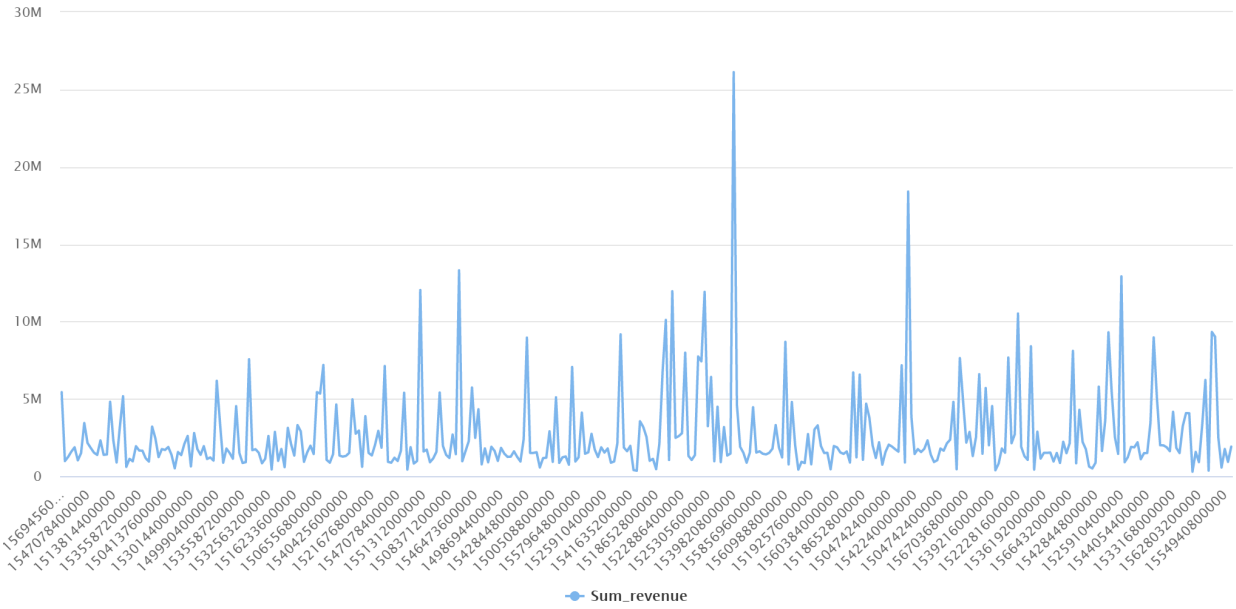
```
df_olap_aggr.write.sqlanalytics("userpool.dbo.productsales", Constants.INTERNAL)
```

Query the results with SQL

You can query the result using a simple SQL query such as the following SQL script:

```
SELECT  [productCode]
,[weekStarting]
,[Sum_quantity]
,[Sum_revenue]
,[AvgPrice]
 FROM [dbo].[productsales]
```

Your query will present the following results in a chart mode:



Next steps

- Query Azure Cosmos DB Analytical Store with Apache Spark 3
- Query Azure Cosmos DB Analytical Store with Apache Spark 2

Query Azure Cosmos DB data with a serverless SQL pool in Azure Synapse Link

8/19/2022 • 15 minutes to read • Edit Online

A serverless SQL pool allows you to analyze data in your Azure Cosmos DB containers that are enabled with Azure Synapse Link in near real time without affecting the performance of your transactional workloads. It offers a familiar T-SQL syntax to query data from the analytical store and integrated connectivity to a wide range of business intelligence (BI) and ad-hoc querying tools via the T-SQL interface.

For querying Azure Cosmos DB, the full SELECT surface area is supported through the OPENROWSET function, which includes the majority of SQL functions and operators. You can also store results of the query that reads data from Azure Cosmos DB along with data in Azure Blob Storage or Azure Data Lake Storage by using create external table as select (CETAS). You can't currently store serverless SQL pool query results to Azure Cosmos DB by using CETAS.

In this article, you'll learn how to write a query with a serverless SQL pool that will query data from Azure Cosmos DB containers that are enabled with Azure Synapse Link. You can then learn more about building serverless SQL pool views over Azure Cosmos DB containers and connecting them to Power BI models in this tutorial. This tutorial uses a container with an Azure Cosmos DB well-defined schema. You can also check out the learn module on how to Query Azure Cosmos DB with SQL Serverless for Azure Synapse Analytics

Prerequisites

- Make sure that you have prepared Analytical store:
  - Enable analytical store on your Cosmos DB containers.
  - Get the connection string with a read-only key that you will use to query analytical store.
  - Get the read-only key that will be used to access Cosmos DB container
- Make sure that you have applied all best practices, such as:
  - Ensure that your Cosmos DB analytical storage is in the same region as serverless SQL pool.
  - Ensure that the client application (Power BI, Analysis service) is in the same region as serverless SQL pool.
  - If you are returning a large amount of data (bigger than 80GB), consider using caching layer such as Analysis services and load the partitions smaller than 80GB in the Analysis services model.
  - If you are filtering data using string columns, make sure that you are using the `OPENROWSET` function with the explicit `WITH` clause that has the smallest possible types (for example, don't use VARCHAR(1000) if you know that the property has up to 5 characters).

Overview

Serverless SQL pool enables you to query Azure Cosmos DB analytical storage using `OPENROWSET` function.

- `OPENROWSET` with inline key. This syntax can be used to query Azure Cosmos DB collections without need to prepare credentials.
- `OPENROWSET` that referenced credential that contains the Cosmos DB account key. This syntax can be used to create views on Azure Cosmos DB collections.

- OPENROWSET with key
- OPENROWSET with credential

To support querying and analyzing data in an Azure Cosmos DB analytical store, a serverless SQL pool is used. The serverless SQL pool uses the `OPENROWSET` SQL syntax, so you must first convert your Azure Cosmos DB connection string to this format:

```
OPENROWSET(
       'CosmosDB',
       '<SQL connection string for Azure Cosmos DB>',
       <Container name>
    )  [ < with clause > ] AS alias
```

The SQL connection string for Azure Cosmos DB specifies the Azure Cosmos DB account name, database name, database account master key, and an optional region name to the `OPENROWSET` function. Some of this information can be taken from the standard Azure Cosmos DB connection string.

Converting from the standard Azure Cosmos DB connection string format:

```
AccountEndpoint=https://<database account name>.documents.azure.com:443/;AccountKey=<database account master key>;
```

The SQL connection string has the following format:

```
'account=<database account name>;database=<database name>;region=<region name>;key=<database account master key>'
```

The region is optional. If omitted, the container's primary region is used.

The Azure Cosmos DB container name is specified without quotation marks in the `OPENROWSET` syntax. If the container name has any special characters, for example, a dash (-), the name should be wrapped within square brackets (`[]`) in the `OPENROWSET` syntax.

IMPORTANT

Make sure that you're using some UTF-8 database collation, for example, `Latin1_General_100_CI_AS_SC_UTF8`, because string values in an Azure Cosmos DB analytical store are encoded as UTF-8 text. A mismatch between text encoding in the file and collation might cause unexpected text conversion errors. You can easily change default collation of the current database by using the T-SQL statement `alter database current collate Latin1_General_100_CI_AI_SC_UTF8`.

NOTE

A serverless SQL pool doesn't support querying an Azure Cosmos DB transactional store.

Sample dataset

The examples in this article are based on data from the European Centre for Disease Prevention and Control (ECDC) COVID-19 Cases and COVID-19 Open Research Dataset (CORD-19), doi:10.5281/zenodo.3715505.

You can see the license and the structure of data on these pages. You can also download sample data for the ECDC and CORD-19 datasets.

To follow along with this article showcasing how to query Azure Cosmos DB data with a serverless SQL pool, make sure that you create the following resources:

- An Azure Cosmos DB database account that's Azure Synapse Link enabled.
- An Azure Cosmos DB database named `covid`.
- Two Azure Cosmos DB containers named `Ecdc` and `Cord19` loaded with the preceding sample datasets.

You can use the following connection string for testing purpose:
`Account=synapselink-cosmosdb-sqlsample;Database=covid;Key=s5zarR2pT0JWH9k8roipnWxUYBegOuFGjJpSjGlR36y86cW0GQ6RaaG8kGjsRAQoWMw1QKTkkX8HQtFpJjC8Hg==`
. Note that this connection will not guarantee performance because this account might be located in remote region compared to your Synapse SQL endpoint.

Explore Azure Cosmos DB data with automatic schema inference

The easiest way to explore data in Azure Cosmos DB is by using the automatic schema inference capability. By omitting the `WITH` clause from the `OPENROWSET` statement, you can instruct the serverless SQL pool to

autodetect (infer) the schema of the analytical store of the Azure Cosmos DB container.

- OPENROWSET with key
- OPENROWSET with credential

```
SELECT TOP 10 *
FROM OPENROWSET(
       'CosmosDB',
       'Account=synapselink-cosmosdb-
sqlsample;Database=covid;Key=s5zarR2pT0JWH9k8roipnWxUYBegOuFGjJpSjGlR36y86cW0GQ6RaaG8kGjsRAQoWMw1QKTkkX8HQtF
pJjC8Hg==',
       Ecdc) as documents
```

In the preceding example, we instructed the serverless SQL pool to connect to the `covid` database in the Azure Cosmos DB account `MyCosmosDbAccount` authenticated by using the Azure Cosmos DB key (the dummy in the preceding example). We then accessed the `Ecdc` container's analytical store in the `West US 2` region. Since there's no projection of specific properties, the `OPENROWSET` function will return all properties from the Azure Cosmos DB items.

Assuming that the items in the Azure Cosmos DB container have `date_rep`, `cases`, and `geo_id` properties, the results of this query are shown in the following table:

DATE_REP	CASES	GEO_ID
2020-08-13	254	RS
2020-08-12	235	RS
2020-08-11	163	RS

If you need to explore data from the other container in the same Azure Cosmos DB database, you can use the same connection string and reference the required container as the third parameter:

```
SELECT TOP 10 *
FROM OPENROWSET(
       'CosmosDB',
       'Account=synapselink-cosmosdb-
sqlsample;Database=covid;Key=s5zarR2pT0JWH9k8roipnWxUYBegOuFGjJpSjGlR36y86cW0GQ6RaaG8kGjsRAQoWMw1QKTkkX8HQtF
pJjC8Hg==',
       Cord19) as cord19
```

Explicitly specify schema

While automatic schema inference capability in `OPENROWSET` provides a simple, easy-to-use querience, your business scenarios might require you to explicitly specify the schema to read-only relevant properties from the Azure Cosmos DB data.

The `OPENROWSET` function enables you to explicitly specify what properties you want to read from the data in the container and to specify their data types.

Let's imagine that we've imported some data from the ECDC COVID dataset with the following structure into Azure Cosmos DB:

```
{"date_rep":"2020-08-13","cases":254,"countries_and_territories":"Serbia","geo_id":"RS"}
{"date_rep":"2020-08-12","cases":235,"countries_and_territories":"Serbia","geo_id":"RS"}
{"date_rep":"2020-08-11","cases":163,"countries_and_territories":"Serbia","geo_id":"RS"}
```

These flat JSON documents in Azure Cosmos DB can be represented as a set of rows and columns in Synapse SQL. The `OPENROWSET` function enables you to specify a subset of properties that you want to read and the exact column types in the `WITH` clause:

- OPENROWSET with key
- OPENROWSET with credential

```
SELECT TOP 10 *
FROM OPENROWSET(
      'CosmosDB',
      'Account=synapselink-cosmosdb-
sqlsample;Database=covid;Key=s5zarR2pT0JWH9k8roipnWxUYBegOuFGjJpSjGlR36y86cW0GQ6RaaG8kGjsRAQoWMw1QKTkkX8HQtF
pJjC8Hg==',
       Ecdc
    ) with ( date_rep varchar(20), cases bigint, geo_id varchar(6) ) as rows
```

The result of this query might look like the following table:

DATE_REP	CASES	GEO_ID
2020-08-13	254	RS
2020-08-12	235	RS
2020-08-11	163	RS

For more information about the SQL types that should be used for Azure Cosmos DB values, see the rules for SQL type mappings at the end of the article.

Create view

Creating views in the `master` or default databases is not recommended or supported. So you need to create a user database for your views.

Once you identify the schema, you can prepare a view on top of your Azure Cosmos DB data. You should place your Azure Cosmos DB account key in a separate credential and reference this credential from `OPENROWSET` function. Do not keep your account key in the view definition.

```
CREATE CREDENTIAL MyCosmosDbAccountCredential
WITH IDENTITY = 'SHARED ACCESS SIGNATURE', SECRET =
's5zarR2pT0JWH9k8roipnWxUYBegOuFGjJpSjGlR36y86cW0GQ6RaaG8kGjsRAQoWMw1QKTkkX8HQtFpJjC8Hg==';
GO
CREATE OR ALTER VIEW Ecdc
AS SELECT *
FROM OPENROWSET(
      PROVIDER = 'CosmosDB',
      CONNECTION = 'Account=synapselink-cosmosdb-sqlsample;Database=covid',
      OBJECT = 'Ecdc',
      SERVER_CREDENTIAL = 'MyCosmosDbAccountCredential'
    ) with ( date_rep varchar(20), cases bigint, geo_id varchar(6) ) as rows
```

Do not use `OPENROWSET` without explicitly defined schema because it might impact your performance. Make sure that you use the smallest possible sizes for your columns (for example VARCHAR(100) instead of default VARCHAR(8000)). You should use some UTF-8 collation as default database collation or set it as explicit column collation to avoid UTF-8 conversion issue. Collation `Latin1_General_100_BIN2_UTF8` provides best performance when you filter data using some string columns.

Query nested objects

With Azure Cosmos DB, you can represent more complex data models by composing them as nested objects or arrays. The autosync capability of Azure Synapse Link for Azure Cosmos DB manages the schema representation in the analytical store out of the box, which includes handling nested data types that allow for rich querying from the serverless SQL pool.

For example, the CORD-19 dataset has JSON documents that follow this structure:

```
{
    "paper_id": <str>,                  # 40-character sha1 of the PDF
    "metadata": {
        "title": <str>,
        "authors": <array of objects>    # list of author dicts, in order
        ...
     }
     ...
}
```

The nested objects and arrays in Azure Cosmos DB are represented as JSON strings in the query result when the `OPENROWSET` function reads them. You can specify the paths to nested values in the objects when you use the `WITH` clause:

```
SELECT TOP 10 *
FROM OPENROWSET(
       'CosmosDB',
       'Account=synapselink-cosmosdb-
sqlsample;Database=covid;Key=s5zarR2pT0JWH9k8roipnWxUYBegOuFGjJpSjGlR36y86cW0GQ6RaaG8kGjsRAQoWMw1QKTkkX8HQtF
pJjC8Hg==',
       Cord19)
WITH (  paper_id    varchar(8000),
        title        varchar(1000) '$.metadata.title',
        metadata     varchar(max),
        authors      varchar(max) '$.metadata.authors'
) AS docs;
```

The result of this query might look like the following table:

PAPER_ID	TITLE	METADATA	AUTHORS
bb11206963e831f...	Supplementary Information An eco-epidemi...	`{"title":"Supplementary Informati…`	`[{"first":"Julien","last":"Mélade","suffix"`
bb1206963e831f1...	The Use of Convalescent Sera in Immune-E...	`{"title":"The Use of Convalescent…`	`[{"first":"Antonio","last":"Lavazza","suff…`
bb378eca9aac649...	Tylosema esculentum (Marama) Tuber and B...	`{"title":"Tylosema esculentum (Ma…`	`[{"first":"Walter","last":"Chingwaru","suf…`

Learn more about analyzing complex data types like Parquet files and containers in Azure Synapse Link for Azure Cosmos DB or nested structures in a serverless SQL pool.

IMPORTANT

If you see unexpected characters in your text like `MÃƒÂ©lade` instead of `Mélade`, then your database collation isn't set to UTF-8 collation. Change collation of the database to UTF-8 collation by using a SQL statement like `ALTER DATABASE MyLdw COLLATE LATIN1_GENERAL_100_CI_AS_SC_UTF8`.

Flatten nested arrays

Azure Cosmos DB data might have nested subarrays like the author's array from a CORD-19 dataset:

```
{
    "paper_id": <str>,                    # 40-character sha1 of the PDF
    "metadata": {
        "title": <str>,
        "authors": [                      # list of author dicts, in order
            {
                "first": <str>,
                "middle": <list of str>,
                "last": <str>,
                "suffix": <str>,
                "affiliation": <dict>,
                "email": <str>
            },
            ...
        ],
        ...
}
```

In some cases, you might need to "join" the properties from the top item (metadata) with all elements of the array (authors). A serverless SQL pool enables you to flatten nested structures by applying the `OPENJSON` function on the nested array:

```
SELECT
    *
FROM
    OPENROWSET(
      'CosmosDB',
      'Account=synapselink-cosmosdb-
sqlsample;Database=covid;Key=s5zarR2pT0JWH9k8roipnWxUYBegOuFGjJpSjGlR36y86cW0GQ6RaaG8kGjsRAQoWMw1QKTkkX8HQtF
pJjC8Hg==',
       Cord19
    ) WITH ( title varchar(1000) '$.metadata.title',
             authors varchar(max) '$.metadata.authors' ) AS docs
      CROSS APPLY OPENJSON ( authors )
                  WITH (
                       first varchar(50),
                       last varchar(50),
                       affiliation nvarchar(max) as json
                  ) AS a
```

The result of this query might look like the following table:

TITLE	AUTHORS	FIRST	LAST	AFFILIATION
Supplementary Information An eco-epidemi…	[{"first":"Julien","last":"Mélade","suffix":"","affiliation":{"laboratory":"Centre de Recher…	Julien	Mélade	{"laboratory":"Centre de Recher…
Supplementary Information An eco-epidemi…	[{"first":"Nicolas","last":"##4#","suffix":"","affiliation":{"laboratory":"","institution":"U…	Nicolas	##4#	{"laboratory":"","institution":"U…
Supplementary Information An eco-epidemi…	[{"first":"Beza","last":"Ramazindrazana","suffix":"","affiliation":{"laboratory":"Centre de Recher…	Beza	Ramazindrazana	{"laboratory":"Centre de Recher…
Supplementary Information An eco-epidemi…	[{"first":"Olivier","last":"Flores","suffix":"","affiliation":{"laboratory":"UMR C53 CIRAD, …	Olivier	Flores	{"laboratory":"UMR C53 CIRAD, …

IMPORTANT

If you see unexpected characters in your text like `MÃƒÃ©lade` instead of `Mélade`, then your database collation isn't set to UTF-8 collation. Change collation of the database to UTF-8 collation by using a SQL statement like `ALTER DATABASE MyLdw COLLATE LATIN1_GENERAL_100_CI_AS_SC_UTF8`.

Azure Cosmos DB to SQL type mappings

Although Azure Cosmos DB transactional store is schema-agnostic, the analytical store is schematized to optimize for analytical query performance. With the autosync capability of Azure Synapse Link, Azure Cosmos DB manages the schema representation in the analytical store out of the box, which includes handling nested data types. Since a serverless SQL pool queries the analytical store, it's important to understand how to map Azure Cosmos DB input data types to SQL data types.

Azure Cosmos DB accounts of SQL (Core) API support JSON property types of number, string, Boolean, null, nested object, or array. You would need to choose SQL types that match these JSON types if you're using the `WITH` clause in `OPENROWSET`. The following table shows the SQL column types that should be used for different property types in Azure Cosmos DB.

AZURE COSMOS DB PROPERTY TYPE	SQL COLUMN TYPE
Boolean	bit
Integer	bigint
Decimal	float
String	varchar (UTF-8 database collation)
Date time (ISO-formatted string)	varchar(30)
Date time (UNIX timestamp)	bigint
Null	`any SQL type`
Nested object or array	varchar(max) (UTF-8 database collation), serialized as JSON text

Full fidelity schema

Azure Cosmos DB full fidelity schema records both values and their best match types for every property in a container. The `OPENROWSET` function on a container with full fidelity schema provides both the type and the actual value in each cell. Let's assume that the following query reads the items from a container with full fidelity schema:

```
SELECT *
FROM OPENROWSET(
      'CosmosDB',
      'account=MyCosmosDbAccount;database=covid;region=westus2;key=C0Sm0sDbKey==',
       Ecdc
    ) as rows
```

The result of this query will return types and values formatted as JSON text:

DATE_REP	CASES	GEO_ID
{"date":"2020-08-13"}	{"int32":"254"}	{"string":"RS"}
{"date":"2020-08-12"}	{"int32":"235"}	{"string":"RS"}
{"date":"2020-08-11"}	{"int32":"316"}	{"string":"RS"}
{"date":"2020-08-10"}	{"int32":"281"}	{"string":"RS"}
{"date":"2020-08-09"}	{"int32":"295"}	{"string":"RS"}
{"string":"2020/08/08"}	{"int32":"312"}	{"string":"RS"}
{"date":"2020-08-07"}	{"float64":"339.0"}	{"string":"RS"}

For every value, you can see the type identified in an Azure Cosmos DB container item. Most of the values for the `date_rep` property contain `date` values, but some of them are incorrectly stored as strings in Azure Cosmos DB. Full fidelity schema will return both correctly typed `date` values and incorrectly formatted `string` values. The number of cases is information stored as an `int32` value, but there's one value that's entered as a decimal number. This value has the `float64` type. If there are some values that exceed the largest `int32` number, they would be stored as the `int64` type. All `geo_id` values in this example are stored as `string` types.

> **IMPORTANT**
>
> The `OPENROWSET` function without a `WITH` clause exposes both values with expected types and the values with incorrectly entered types. This function is designed for data exploration and not for reporting. Don't parse JSON values returned from this function to build reports. Use an explicit WITH clause to create your reports. You should clean up the values that have incorrect types in the Azure Cosmos DB container to apply corrections in the full fidelity analytical store.

If you need to query Azure Cosmos DB accounts of the Mongo DB API kind, you can learn more about the full fidelity schema representation in the analytical store and the extended property names to be used in What is Azure Cosmos DB Analytical Store?.

Query items with full fidelity schema

While querying full fidelity schema, you need to explicitly specify the SQL type and the expected Azure Cosmos DB property type in the `WITH` clause.

In the following example, we'll assume that `string` is the correct type for the `geo_id` property and `int32` is the correct type for the `cases` property:

```
SELECT geo_id, cases = SUM(cases)
FROM OPENROWSET(
      'CosmosDB'
      'account=MyCosmosDbAccount;database=covid;region=westus2;key=C0Sm0sDbKey==',
       Ecdc
    ) WITH ( geo_id VARCHAR(50) '$.geo_id.string',
             cases INT '$.cases.int32'
    ) as rows
GROUP BY geo_id
```

Values for `geo_id` and `cases` that have other types will be returned as `NULL` values. This query will reference only the `cases` with the specified type in the expression (`cases.int32`).

If you have values with other types (`cases.int64`, `cases.float64`) that can't be cleaned in an Azure Cosmos DB container, you would need to explicitly reference them in a `WITH` clause and combine the results. The following query aggregates both `int32`, `int64`, and `float64` stored in the `cases` column:

```
SELECT geo_id, cases = SUM(cases_int) + SUM(cases_bigint) + SUM(cases_float)
FROM OPENROWSET(
      'CosmosDB',
      'account=MyCosmosDbAccount;database=covid;region=westus2;key=C0Sm0sDbKey==',
       Ecdc
    ) WITH ( geo_id VARCHAR(50) '$.geo_id.string',
             cases_int INT '$.cases.int32',
             cases_bigint BIGINT '$.cases.int64',
             cases_float FLOAT '$.cases.float64'
    ) as rows
GROUP BY geo_id
```

In this example, the number of cases is stored either as `int32`, `int64`, or `float64` values. All values must be extracted to calculate the number of cases per country.

Troubleshooting

Review the self-help page to find the known issues or troubleshooting steps that can help you to resolve potential problems with Cosmos DB queries.

Next steps

For more information, see the following articles:

- Use Power BI and serverless SQL pool with Azure Synapse Link
- Create and use views in a serverless SQL pool
- Tutorial on building serverless SQL pool views over Azure Cosmos DB and connecting them to Power BI models via DirectQuery
- Visit the Azure Synapse link for Cosmos DB self-help page if you are getting some errors or experiencing performance issues.
- Checkout the learn module on how to Query Azure Cosmos DB with SQL Serverless for Azure Synapse Analytics.

Get started with Azure Synapse Link for Azure SQL Database (Preview)

8/19/2022 • 6 minutes to read • Edit Online

This article provides a step-by-step guide for getting started with Azure Synapse Link for Azure SQL Database. For more information, see Synapse Link for Azure SQL Database (Preview).

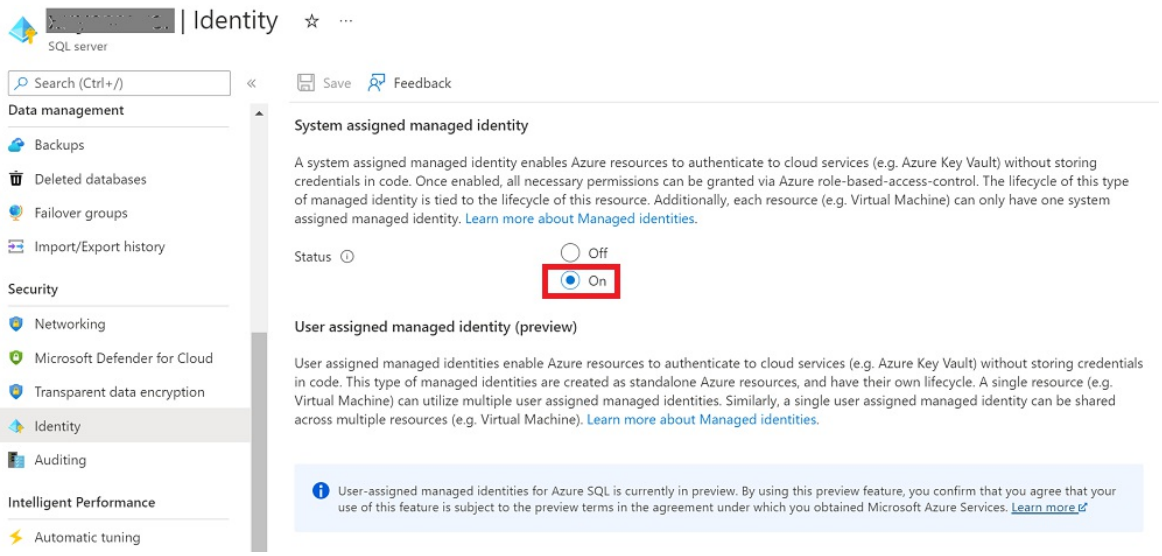
> **IMPORTANT**
>
> Azure Synapse Link for SQL is currently in PREVIEW. See the Supplemental Terms of Use for Microsoft Azure Previews for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

Prerequisites

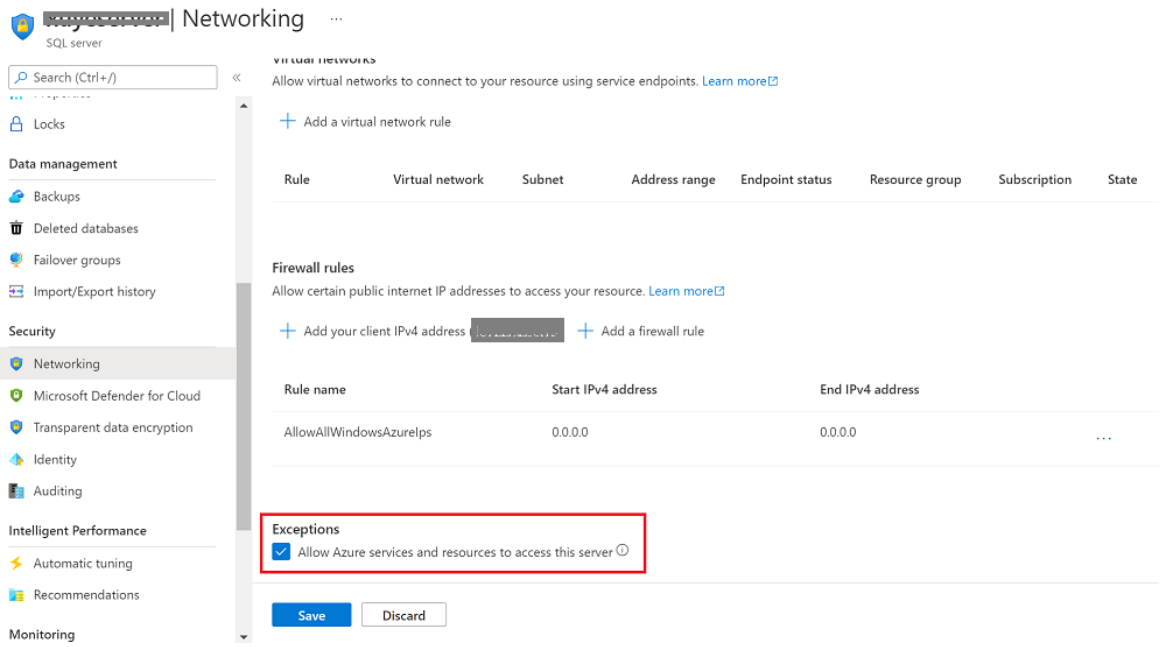
- Create a new Synapse workspace to get Azure Synapse Link for SQL. Make sure to check "Disable Managed virtual network" and "Allow connections from all IP address" when creating Synapse workspace.
- For DTU-based provisioning, make sure your Azure SQL Database service is at least Standard tier with a minimum of 100 DTUs. Free, Basic, or Standard tiers with fewer than 100 DTUs provisioned are not supported.

Configure your source Azure SQL Database

1. Go to Azure portal, navigate to your Azure SQL Server, select **Identity**, and then set **System assigned managed identity** to **On**.

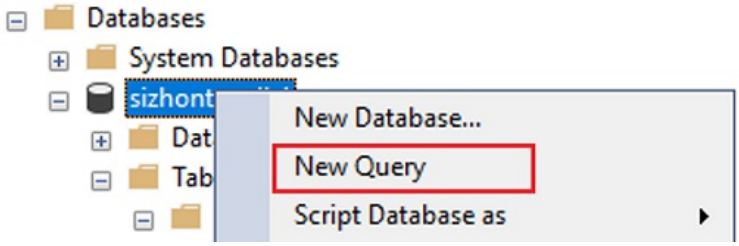


2. Navigate to **Networking**, then check **Allow Azure services and resources to access this server**.



3. Using Microsoft SQL Server Management Studio (SSMS) or Azure Data Studio, connect to the Azure SQL Server. If you want to have your Synapse workspace connect to your Azure SQL Database using a managed identity, set the Azure Active Directory admin on Azure SQL Server, and use the same admin name to connect to Azure SQL Server with administrative privileges in order to have the privileges in step 5.

4. Expand **Databases**, right select the database you created above, and select **New Query**.



5. If you want to have your Synapse workspace connect to your source Azure SQL Database using a managed identity, run the following script to provide the managed identity permission to the source database.

   You can skip this step if you instead want to have your Synapse workspace connect to your source Azure SQL Database via SQL authentication.

```
CREATE USER <workspace name> FROM EXTERNAL PROVIDER;
ALTER ROLE [db_owner] ADD MEMBER <workspace name>;
```

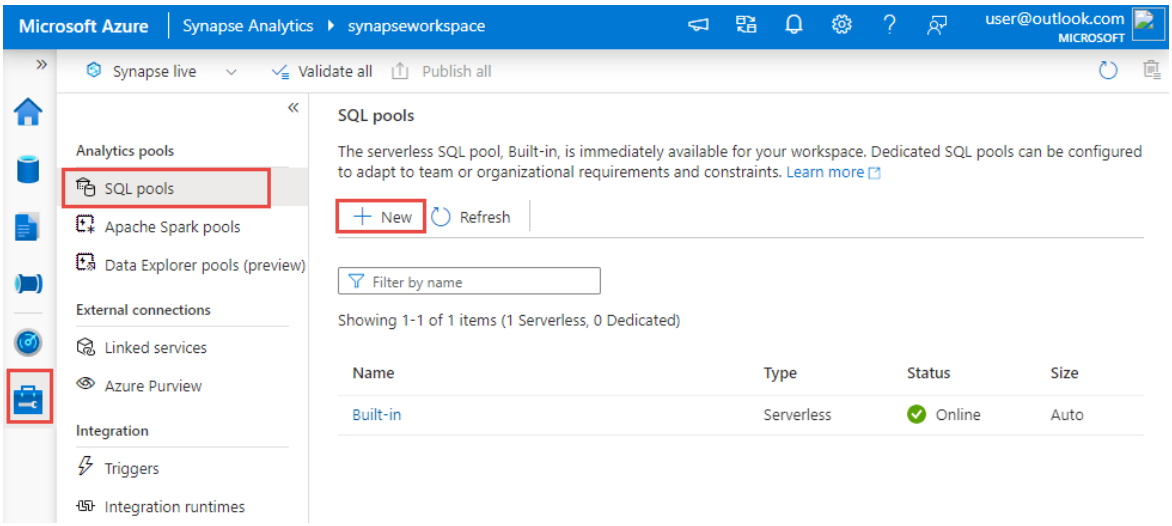
6. You can create a table with your own schema; the following is just an example for a `CREATE TABLE` query. You can also insert some rows into this table to ensure there's data to be replicated.

```
CREATE TABLE myTestTable1 (c1 int primary key, c2 int, c3 nvarchar(50))
```

Create your target Synapse SQL pool

1. Launch Synapse Studio.

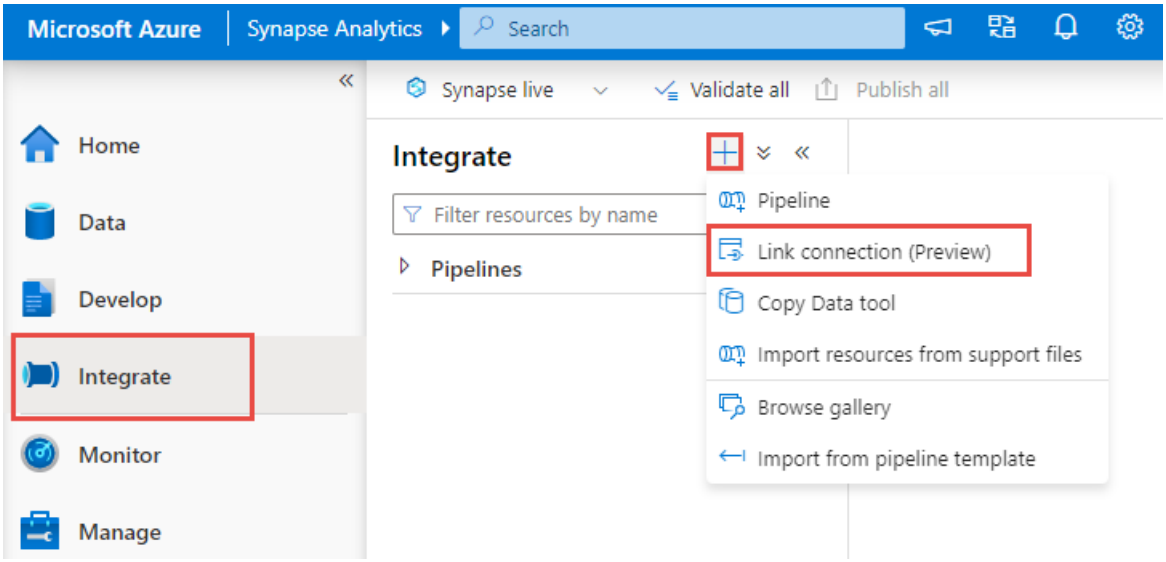
2. Open the **Manage** hub, navigate to **SQL pools**, and select **+ New**.



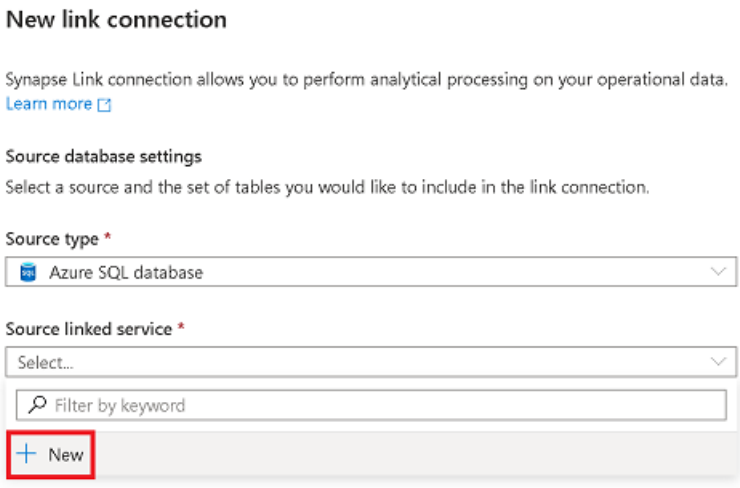
3. Enter a unique pool name, use the default settings, and create the dedicated pool.
4. You need to create a schema if your expected schema is not available in target Synapse SQL database. If your schema is dbo, you can skip this step.

Create the Azure Synapse Link connection

1. Open the **Integrate** hub, and select **+ Link connection(Preview)**.

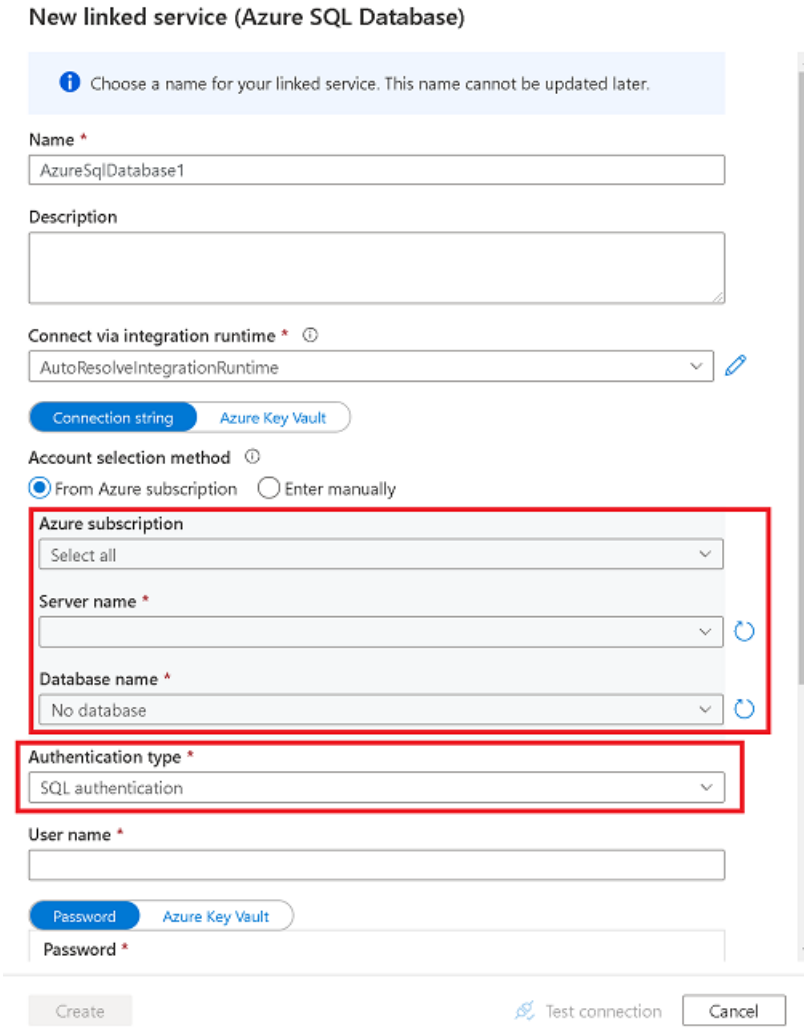


2. Under **Source linked service**, select **New**.



3. Enter the information for your source Azure SQL Database.

- Select the subscription, server, and database corresponding to your Azure SQL Database.
- If you wish to connect your Synapse workspace to the source DB using the workspace's managed identity, set **Authentication type** to **Managed Identity**.
- If you wish to use SQL authentication instead and know the username/password to use, select **SQL Authentication** instead.



4. Select **Test connection** to ensure the firewall rules are properly configured and the workspace can successfully connect to the source Azure SQL Database.

5. Select **Create**.

> **NOTE**
>
> The linked service that you create here is not dedicated to Azure Synapse Link for SQL - it can be used by any workspace user that has the appropriate permissions. Please take time to understand the scope of users who may have access to this linked service and its credentials. For more information on permissions in Azure Synapse workspaces, see Azure Synapse workspace access control overview - Azure Synapse Analytics.

6. Select one or more source tables to replicate to your Synapse workspace and select **Continue**.

> **NOTE**
>
> A given source table can only be enabled in at most one link connection at a time.

7. Select a target Synapse SQL database and pool.

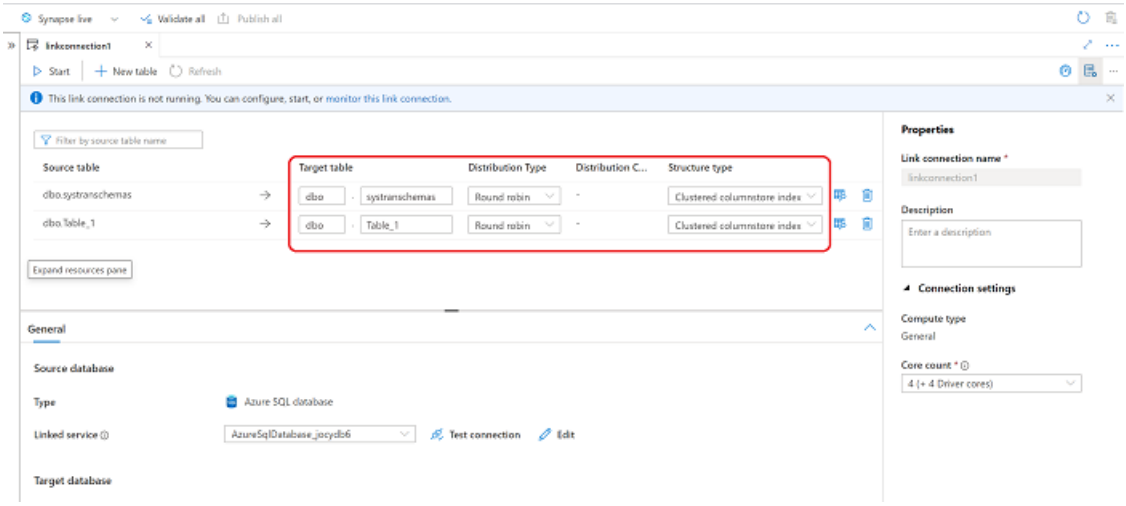
8. Provide a name for your Azure Synapse Link connection, and select the number of cores. These cores will be used for the movement of data from the source to the target.

   > **NOTE**
   >
   > We recommend starting low and increasing as needed.

9. Select **OK**.

10. With the new Azure Synapse Link connection open, you can update the target table name, distribution type and structure type.

    > **NOTE**
    >
    > - Consider heap table for structure type when your data contains varchar(max), nvarchar(max), and varbinary(max).
    > - Make sure the schema in your Synapse dedicated SQL pool has already been created before you start the link connection. Azure Synapse Link for SQL will create tables automatically under your schema in the Synapse dedicated SQL pool.



11. Select **Publish all** to save the new link connection to the service.

Start the Azure Synapse Link connection

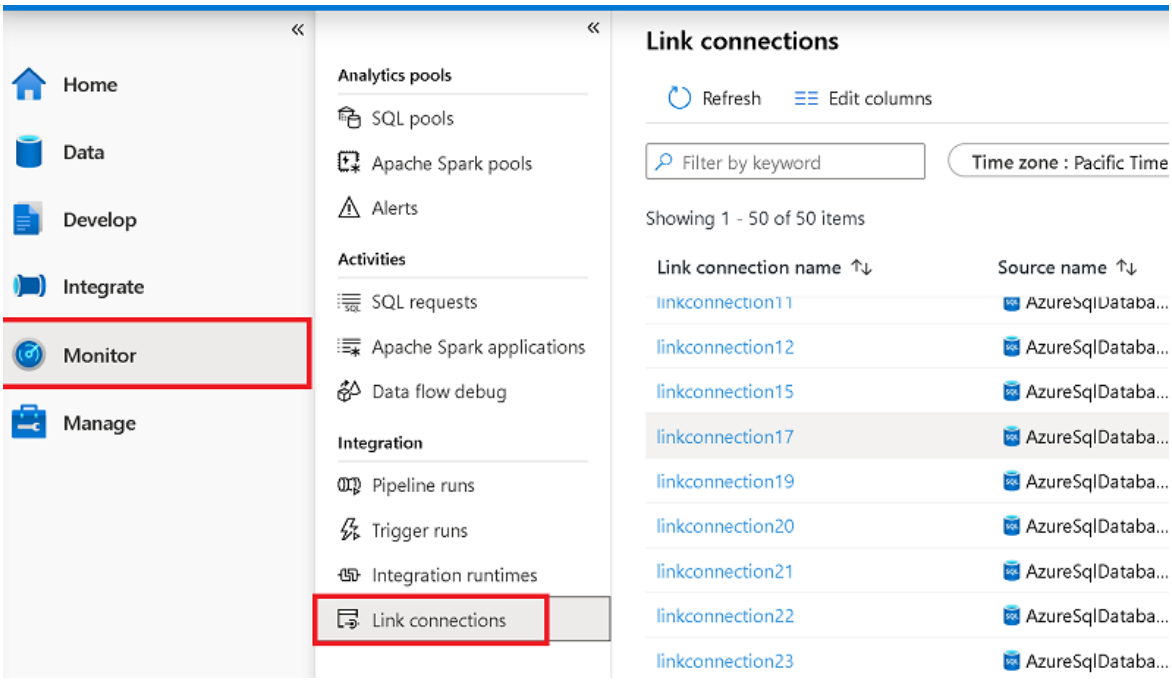
1. Select **Start** and wait a few minutes for the data to be replicated.

   > **NOTE**
   >
   > When being started, a link connection will start from a full initial load from your source database followed by incremental change feeds via the change feed feature in Azure SQL database. For more information, see Azure Synapse Link for SQL change feed.

Monitor the status of the Azure Synapse Link connection

You may monitor the status of your Azure Synapse Link connection, see which tables are being initially copied over (Snapshotting), and see which tables are in continuous replication mode (Replicating).

1. Navigate to the **Monitor** hub, and select **Link connections**.



2. Open the Azure Synapse Link connection you started and view the status of each table.
3. Select **Refresh** on the monitoring view for your connection to observe any updates to the status.

Query replicated data

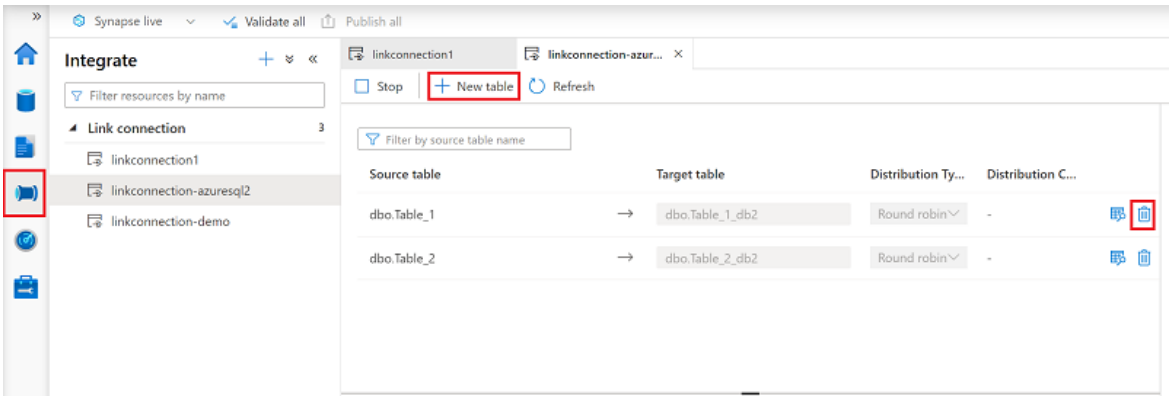
Wait for a few minutes, then check the target database has the expected table and data. You can also now explore the replicated tables in your target Synapse dedicated SQL pool.

1. In the **Data** hub, under **Workspace**, open your target database, and within **Tables**, right-click one of your target tables.
2. Choose **New SQL script**, then **Select TOP 100 rows**.
3. Run this query to view the replicated data in your target Synapse dedicated SQL pool.
4. You can also query the target database with SSMS (or other tools). Use the dedicated SQL endpoint for your workspace as the server name. This is typically `<workspacename>.sql.azuresynapse.net`. Add `Database=databasename@poolname` as another connection string parameter when connecting via SSMS (or other tools).

Add/remove table in existing Azure Synapse Link connection

You can add/remove tables on Synapse Studio as following:

1. Open the **Integrate Hub**.
2. Select the **Link connection** you want to edit and open it.
3. Select **+New** table to add tables on Synapse Studio or select the trash can icon to the right or a table to remove an existing table. You can add or remove tables when the link connection is running.

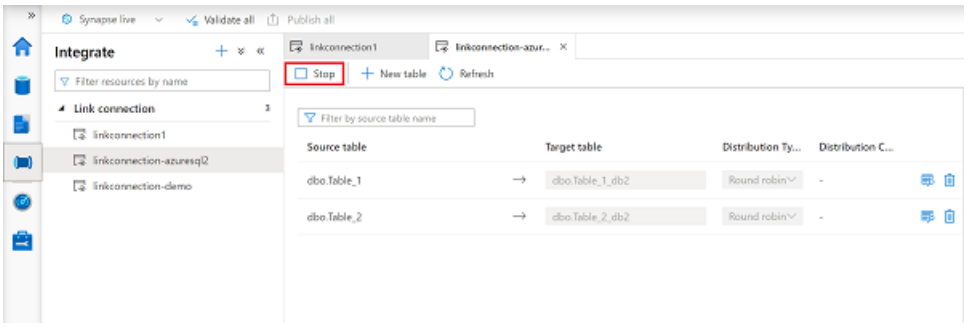


> **NOTE**
>
> You can directly add or remove tables when a link connection is running.

Stop the Azure Synapse Link connection

You can stop the Azure Synapse Link connection in Synapse Studio as follows:

1. Open the **Integrate Hub** of your Synapse workspace.
2. Select the **Link connection** you want to edit and open it.
3. Select **Stop** to stop the link connection, and it will stop replicating your data.



> **NOTE**
>
> If you restart a link connection after stopping it, it will start from a full initial load from your source database followed by incremental change feeds.

Next steps

If you are using a different type of database, see how to:

- Configure Azure Synapse Link for Azure Cosmos DB
- Configure Azure Synapse Link for Dataverse
- Get started with Azure Synapse Link for SQL Server 2022

Known limitations and issues with Azure Synapse Link for SQL

8/19/2022 • 5 minutes to read • Edit Online

This article lists the known limitations and issues with Azure Synapse Link for SQL.

Known limitations

This is the list of known limitations for Azure Synapse Link for SQL.

Azure SQL DB and SQL Server 2022

- Users must use an Azure Synapse Analytics workspace created on or after May 24, 2022, to get access to Azure Synapse Link for SQL functionality.
- Running Azure Synapse Analytics in a managed virtual network isn't supported. Users need to check "Disable Managed virtual network" and "Allow connections from all IP addresses" when creating their workspace.
- Source tables must have primary keys.
- The following data types aren't supported for primary keys in the source tables:
  - real
  - float
  - hierarchyid
  - sql_variant
  - timestamp
- Source table row size can't exceed 7,500 bytes. For tables where variable-length columns are stored off-row, a 24-byte pointer is stored in the main record.
- Tables enabled for Azure Synapse Link for SQL can have a maximum of 1,020 columns (not 1,024).
- While a database can have multiple links enabled, a given table can't belong to multiple links.
- When a database owner doesn't have a mapped log in, Azure Synapse link for SQL will run into an error when enabling a link connection. User can set database owner to a valid user with the `ALTER AUTHORIZATION` command to fix this issue.
- If the source table contains computed columns or columns with data types that aren't supported by Azure Synapse Analytics dedicated SQL pools, these columns won't be replicated to Azure Synapse Analytics. Unsupported columns include:
  - image
  - text
  - xml
  - timestamp
  - sql_variant
  - UDT
  - geometry
  - geography
- A maximum of 5,000 tables can be added to a single link connection.
- When a source column is of type datetime2(7) or time(7), the last digit will be truncated when data is replicated to Azure Synapse Analytics.
- The following table DDL operations aren't allowed on source tables when they are enabled for Azure Synapse Link for SQL. All other DDL operations are allowed, but they won't be replicated to Azure Synapse Analytics.

    - Switch Partition
    - Add/Drop/Alter Column
    - Alter Primary Key
    - Drop/Truncate Table
    - Rename Table
- If DDL + DML is executed in an explicit transaction (between `BEGIN TRANSACTION` and `END TRANSACTION` statements), replication for corresponding tables will fail within the link connection.

  > **NOTE**
  > If a table is critical for transactional consistency at the link connection level, please review the state of the Azure Synapse Link table in the Monitoring tab.

- Azure Synapse Link for SQL can't be enabled if any of the following features are in use for the source table:
    - Change Data Capture
    - Temporal history table
    - Always encrypted
    - In-Memory OLTP
    - Column Store Index
    - Graph
- System tables can't be replicated.
- The security configuration from the source database will **NOT** be reflected in the target dedicated SQL pool.
- Enabling Azure Synapse Link for SQL will create a new schema called `changefeed`. Don't use this schema, as it is reserved for system use.
- Source tables with non-default collations: UTF8, Japanese can't be replicated to Synapse. Here's the supported collations in Synapse SQL Pool.
- Single row updates (including off-page storage) of > 370MB are not supported.

Azure SQL DB only

- Azure Synapse Link for SQL isn't supported on Free, Basic or Standard tier with fewer than 100 DTUs.
- Azure Synapse Link for SQL isn't supported on SQL Managed Instances.
- Users need to check "Allow Azure services and resources to access this server" in the firewall settings of their source database server.
- Service principal isn't supported for authenticating to source Azure SQL DB, so when creating Azure SQL DB linked Service, choose SQL authentication, user-assigned managed identity (UAMI) or service assigned managed Identity (SAMI).
- Azure Synapse Link can't be enabled on the secondary database once a GeoDR failover has happened if the secondary database has a different name from the primary database.
- If you enabled Azure Synapse Link for SQL on your database as an Microsoft Azure Active Directory (Azure AD) user, Point-in-time restore (PITR) will fail. PITR will only work when you enable Azure Synapse Link for SQL on your database as a SQL user.
- If you create a database as an Azure AD user and enable Azure Synapse Link for SQL, a SQL authentication user (for example, even sysadmin role) won't be able to disable/make changes to Azure Synapse Link for SQL artifacts. However, another Azure AD user will be able to enable/disable Azure Synapse Link for SQL on the same database. Similarly, if you create a database as an SQL authentication user, enabling/disabling Azure Synapse Link for SQL as an Azure AD user won't work.
- When enabling Azure Synapse Link for SQL on your Azure SQL Database, you should ensure that aggressive log truncation is disabled.

SQL Server 2022 only

- Azure Synapse Link for SQL works with SQL Server on Linux, but HA scenarios with Linux Pacemaker aren't supported. Shelf hosted IR cannot be installed on Linux environment.
- Azure Synapse Link for SQL can't be enabled on databases that are transactional replication publishers or distributors.
- If the SAS key of landing zone expires and gets rotated during the snapshot process, the new key won't get picked up. The snapshot will fail and restart automatically with the new key.
- Prior to breaking an Availability Group, disable any running links. Otherwise both databases will attempt to write their changes to the landing zone.
- When using asynchronous replicas, transactions need to be written to all replicas prior to them being published to Azure Synapse Link for SQL.
- Azure Synapse Link for SQL isn't supported on databases with database mirroring enabled.
- Restoring an Azure Synapse Link for SQL-enabled database from on-premises to Azure SQL Managed Instance isn't supported.
- Azure Synapse Link for SQL is not supported on databases that are using Managed Instance Link.

Known issues

Deleting an Azure Synapse Analytics workspace with a running link could cause log in source database to fill

- Applies To - Azure SQL Database and SQL Server 2022
- Issue - When you delete an Azure Synapse Analytics workspace it is possible that running links might not be stopped, which will cause the source database to think that the link is still operational and could lead to the log filling and not being truncated.
- Resolution - There are two possible resolutions to this situation:

1. Stop any running links prior to deleting the Azure Synapse Analytics workspace.
2. Manually clean up the link definition in the source database.
   a. Find the table_group_id for the link(s) that need to be stopped using the following query:

```
SELECT table_group_id, workspace_id, synapse_workgroup_name
FROM [changefeed].[change_feed_table_groups]
WHERE synapse_workgroup_name = <synapse workspace name>
```

   b. Drop each link identified using the following procedure:

```
EXEC sys.sp_change_feed_drop_table_group @table_group_id = <table_group_id>
```

   c. Optionally, if you are disabling all of the table groups for a given database, you can also disable change feed on the database with the following command:

```
EXEC sys.sp_change_feed_disable_db
```

Next steps

If you are using a different type of database, see how to:

- Configure Synapse Link for Azure Cosmos DB
- Configure Synapse Link for Dataverse

Get started with Azure Synapse Link for SQL Server 2022 (Preview)

8/19/2022 • 8 minutes to read • Edit Online

This article provides a step-by-step guide for getting started with Azure Synapse Link for SQL Server 2022. For more information, see Get started with Azure Synapse Link for SQL Server 2022 (Preview).

> **IMPORTANT**
>
> Azure Synapse Link for SQL is currently in PREVIEW. See the Supplemental Terms of Use for Microsoft Azure Previews for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

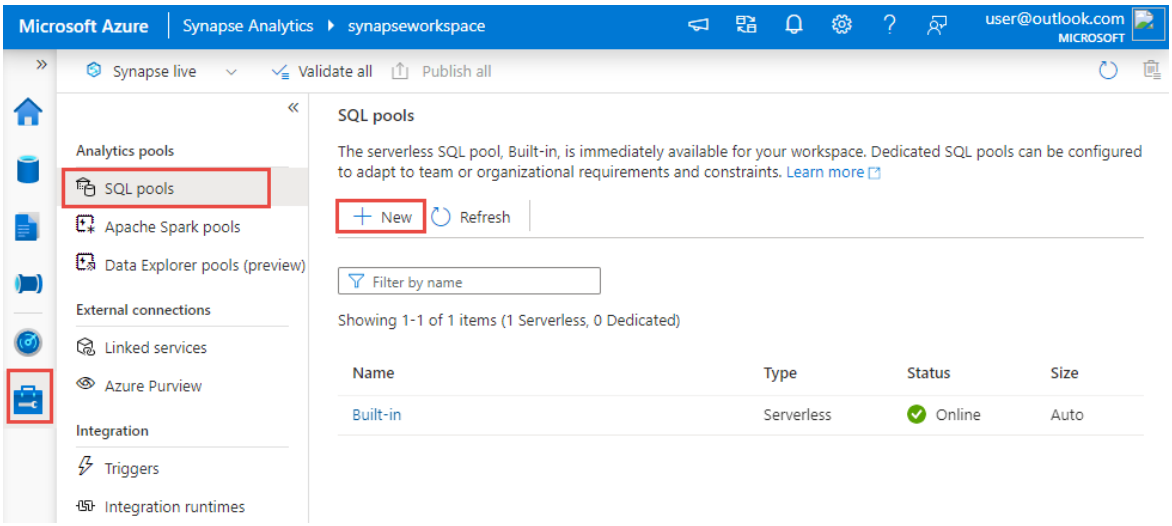
Prerequisites

- Create a new Synapse workspace to get Azure Synapse Link for SQL. Ensure to check "Disable Managed virtual network" and "Allow connections from all IP address" when creating Synapse workspace. If you have a workspace created after May 24, 2022, you do not need to create a new workspace.
- Create an Azure Data Lake Storage Gen2 account (different from the account created with the Azure Synapse Analytics workspace) used as the landing zone to stage the data submitted by SQL Server 2022. See how to create a Azure Data Lake Storage Gen2 account article for more details.
- Make sure your database in SQL Server 2022 has a master key created.

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<a new password>'
```

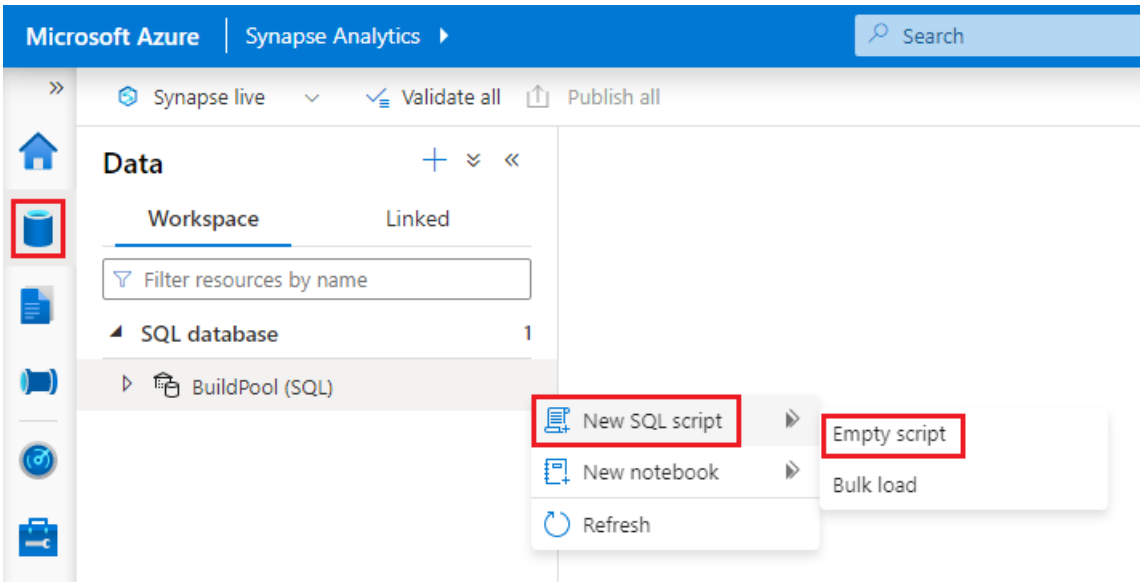
Create your target Synapse dedicated SQL pool

1. Launch Synapse Studio.
2. Open the **Manage** hub, navigate to **SQL pools**, and select **+ New**.



3. Enter a unique pool name, use the default settings, and create the dedicated pool.
4. From the **Data** hub, under **Workspace**, you should see your new Synapse SQL database listed under

Databases. From your new Synapse SQL database, select **New SQL script**, then **Empty script**.

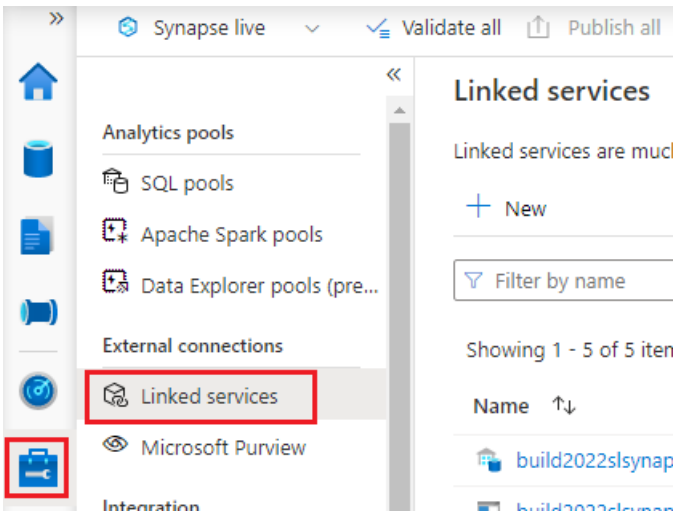


5. Paste the following script and select **Run** to create the master key for your target Synapse SQL database. You also need to create a schema if your expected schema is not available in target Synapse SQL database.

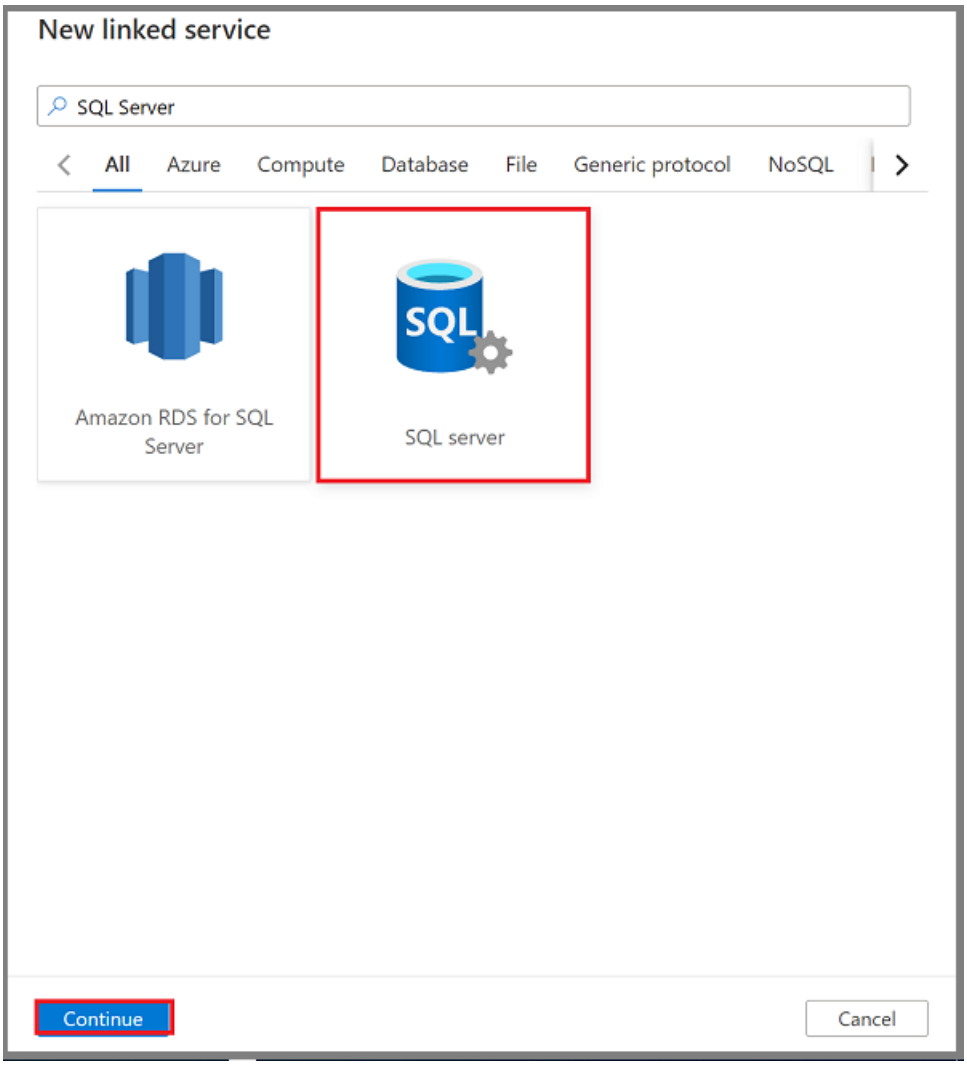
```
CREATE MASTER KEY
```

Create linked service for your source SQL Server 2022

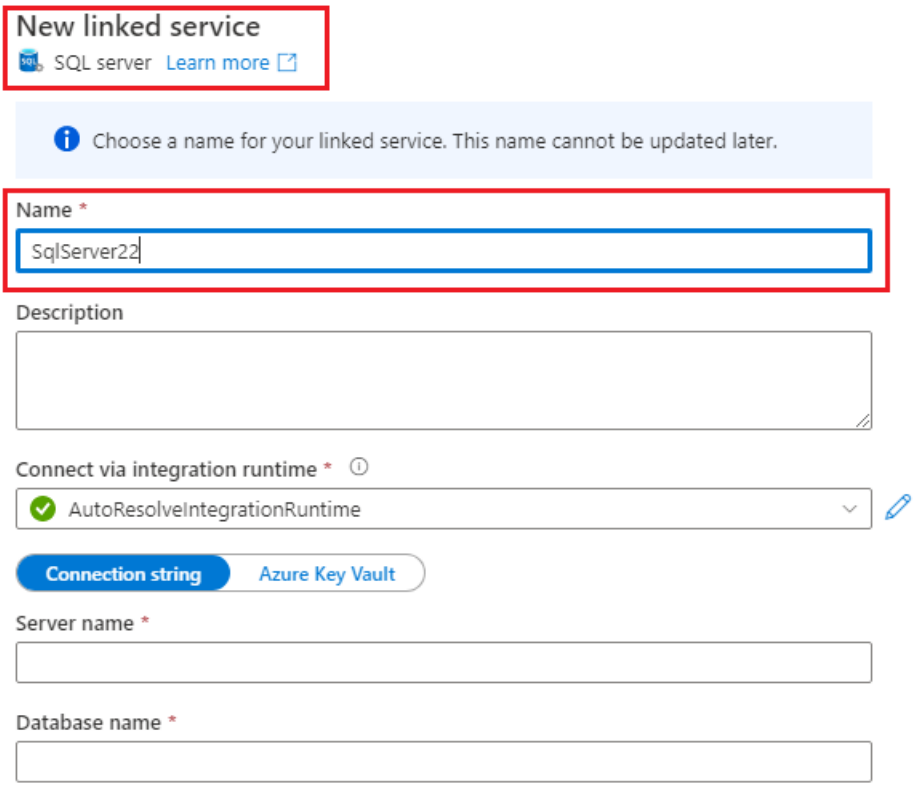
1. Open the **Manage** hub, and navigate to **Linked services**.



2. Press **+ New**, select **SQL Server** and select **Continue**.



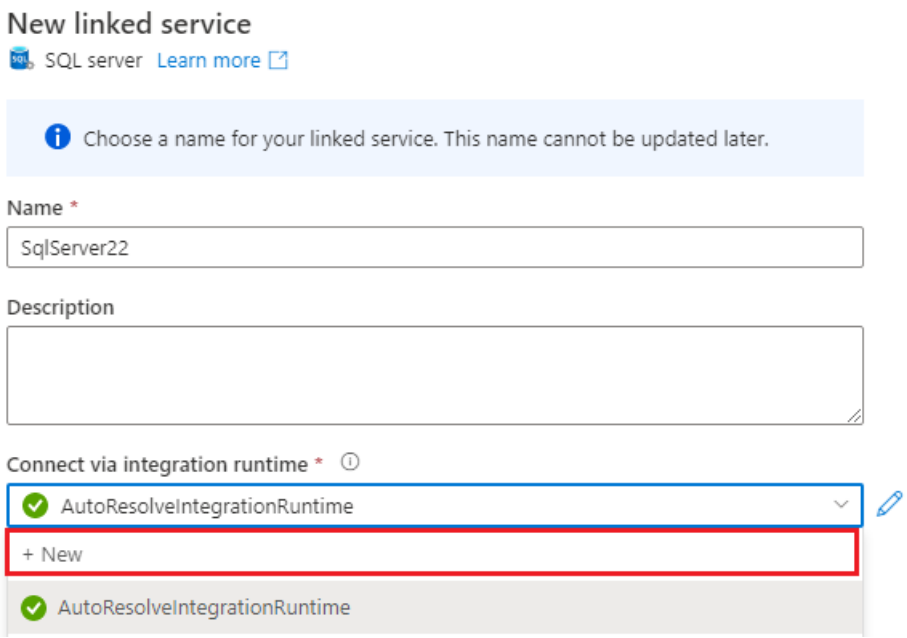
3. Enter the **name** of linked service of SQL Server 2022.



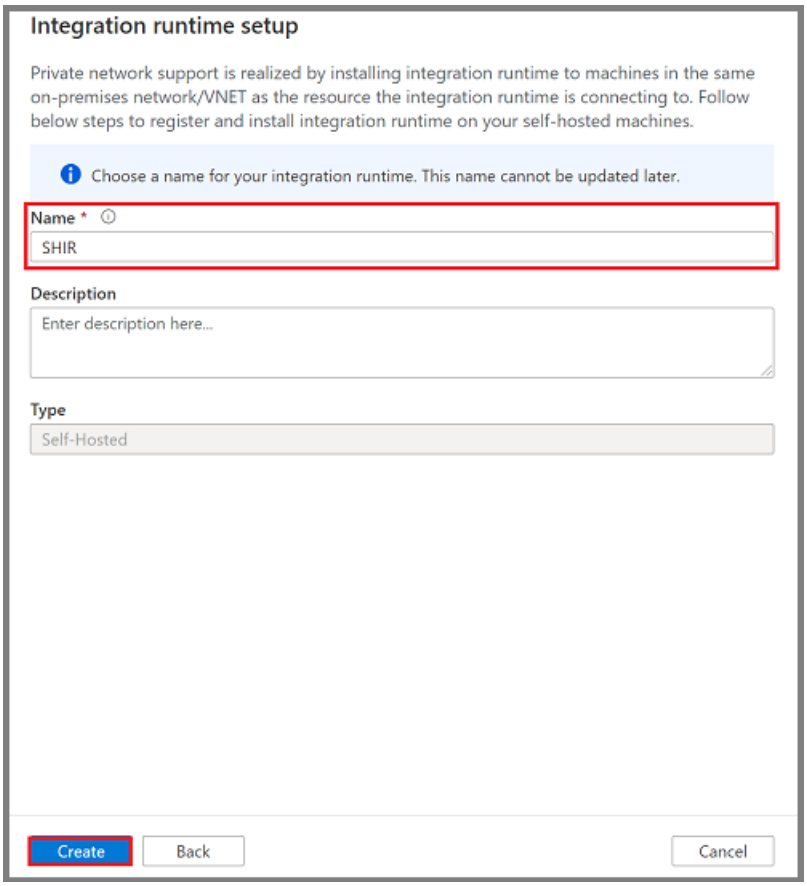
4. When selecting the integration runtime, choose your **self-hosted integration runtime**. If your synapse workspace doesn't have self-hosted integration runtime available, create one.

5. Use the following steps to create a self-hosted integration runtime to connect to your source SQL Server 2022 (optional)

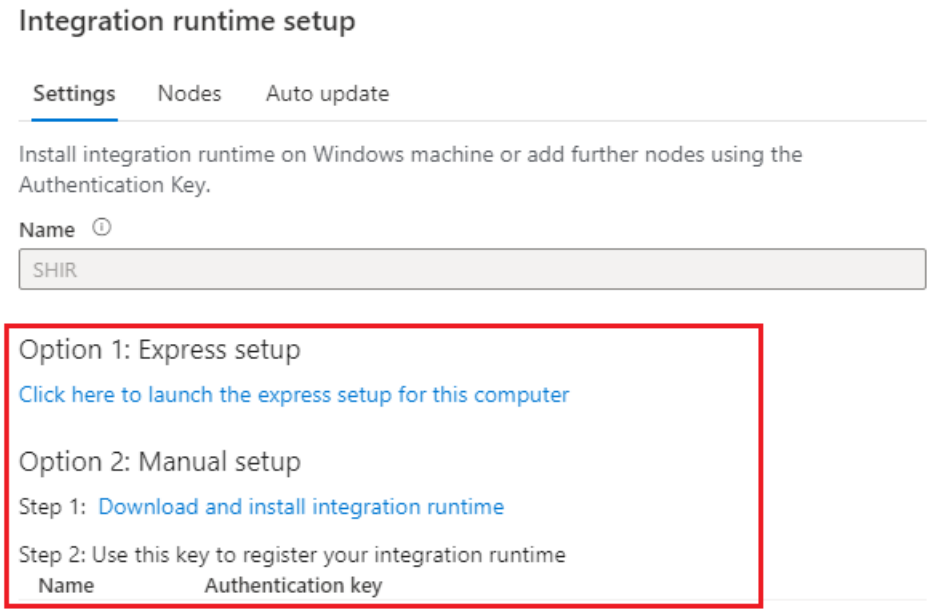
   - Select **+New**.



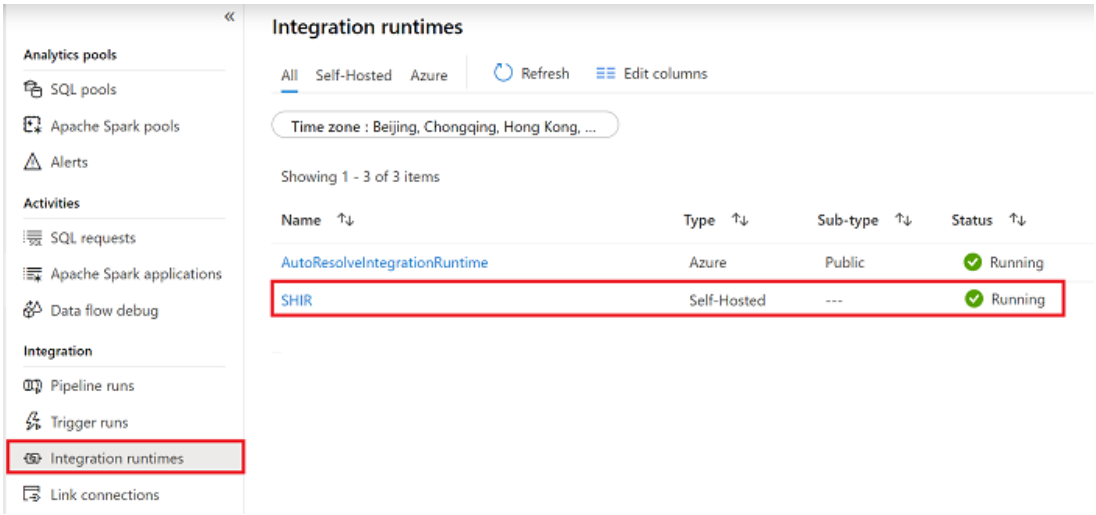
   - Select **Self-hosted** and select **continue**.
   - Input the **name** of Self-hosted integration runtime and select **Create**.



   - Now a self-hosted integration runtime is available in your Synapse workspace. Follow the prompts in the UI to **download**, **install** and use the key to **register** your integration runtime agent on your windows machine, which has direct access on your SQL Server 2022. For more information, see Create a self-hosted integration runtime - Azure Data Factory & Azure Synapse



- Select **Close**, and go to monitoring page to make sure your self-hosted integration runtime is running by selecting **refresh** to get the latest status of integration runtime.

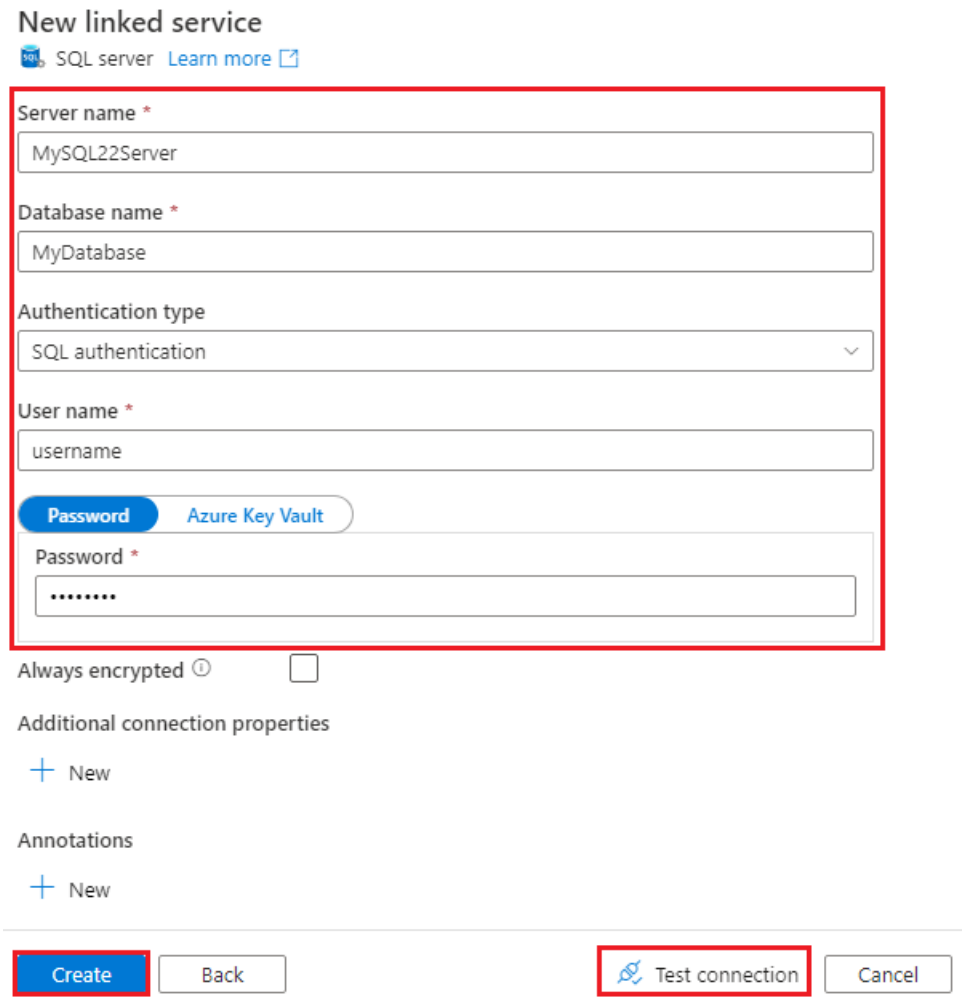


6. Continue to input the rest information on your linked service including **SQL Server name**, **Database name**, **Authentication type**, **User name** and **Password** to connect to your SQL Server 2022.

> **NOTE**
>
> We recommend that you enable encryption on this connection. To enable encryption, add the `Encrypt` property with a value of `true` as an Additional connection property, and also set the `Trust Server Certificate` property to either `true` or `false` - depending on your server configuration. For more information, see Enable encrypted connections to the Database Engine.

7. Select **Test Connection** to ensure your self-hosted integration runtime can access on your SQL Server.

8. Select **Create**, and you'll have your new linked service connecting to SQL Server 2022 available in your workspace.

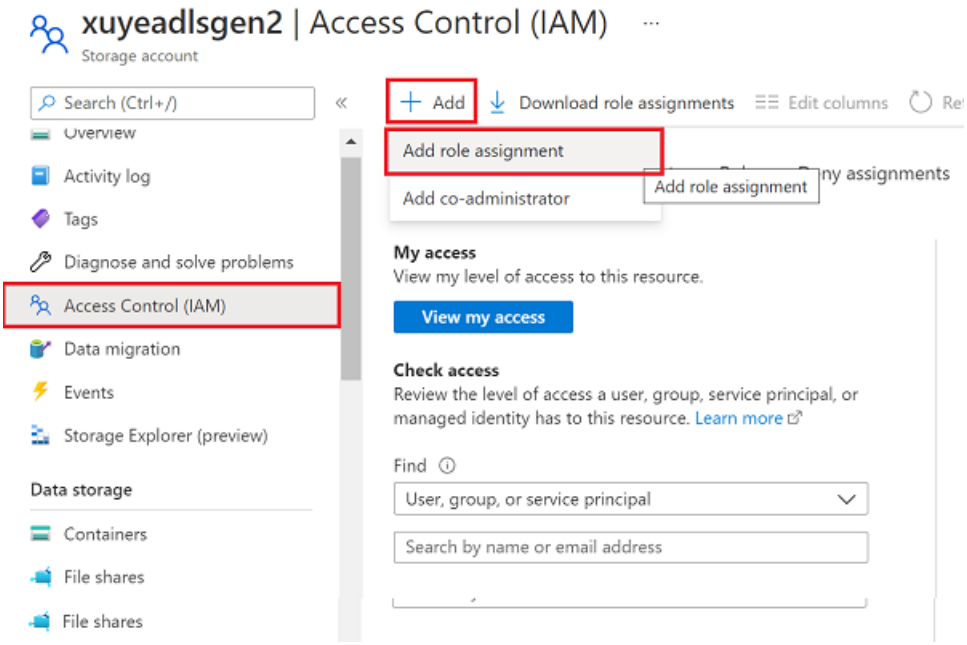


NOTE

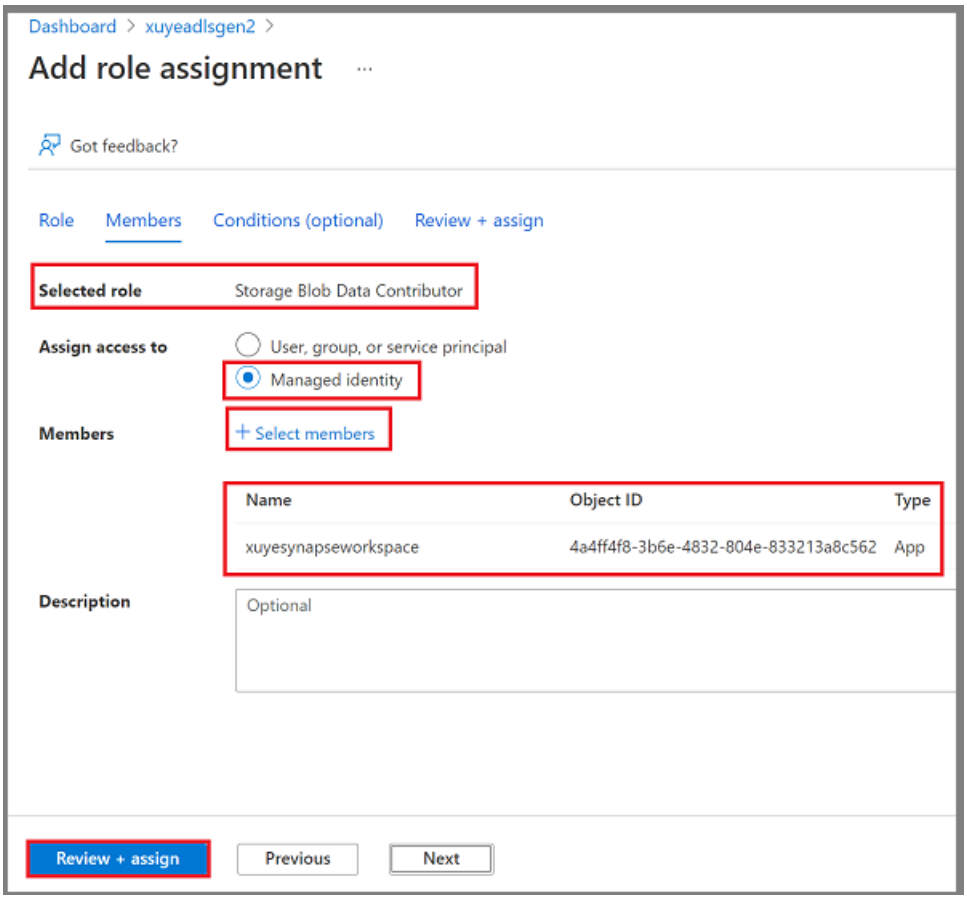
The linked service that you create here is not dedicated to Azure Synapse Link for SQL - it can be used by any workspace user that has the appropriate permissions. Please take time to understand the scope of users who may have access to this linked service and its credentials. For more information on permissions in Azure Synapse workspaces, see Azure Synapse workspace access control overview - Azure Synapse Analytics.

Create linked service to connect to your landing zone on Azure Data Lake Storage Gen2

1. Go to your created Azure Data Lake Storage Gen2 account, navigate to **Access Control (IAM)**, select **+Add**, and select **Add role assignment**.

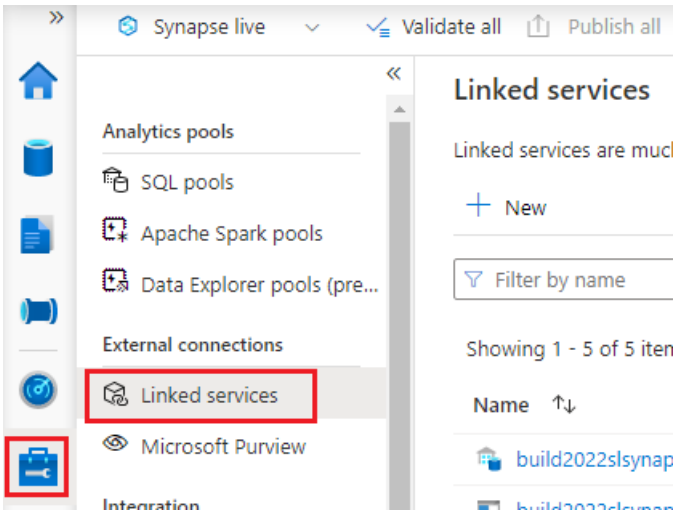


2. Select **Storage Blob Data Contributor** for the selected role, choose **Managed identity** in Managed identity, and select your Synapse workspace in **Members**. This may take a few minutes to take effect to add role assignment.

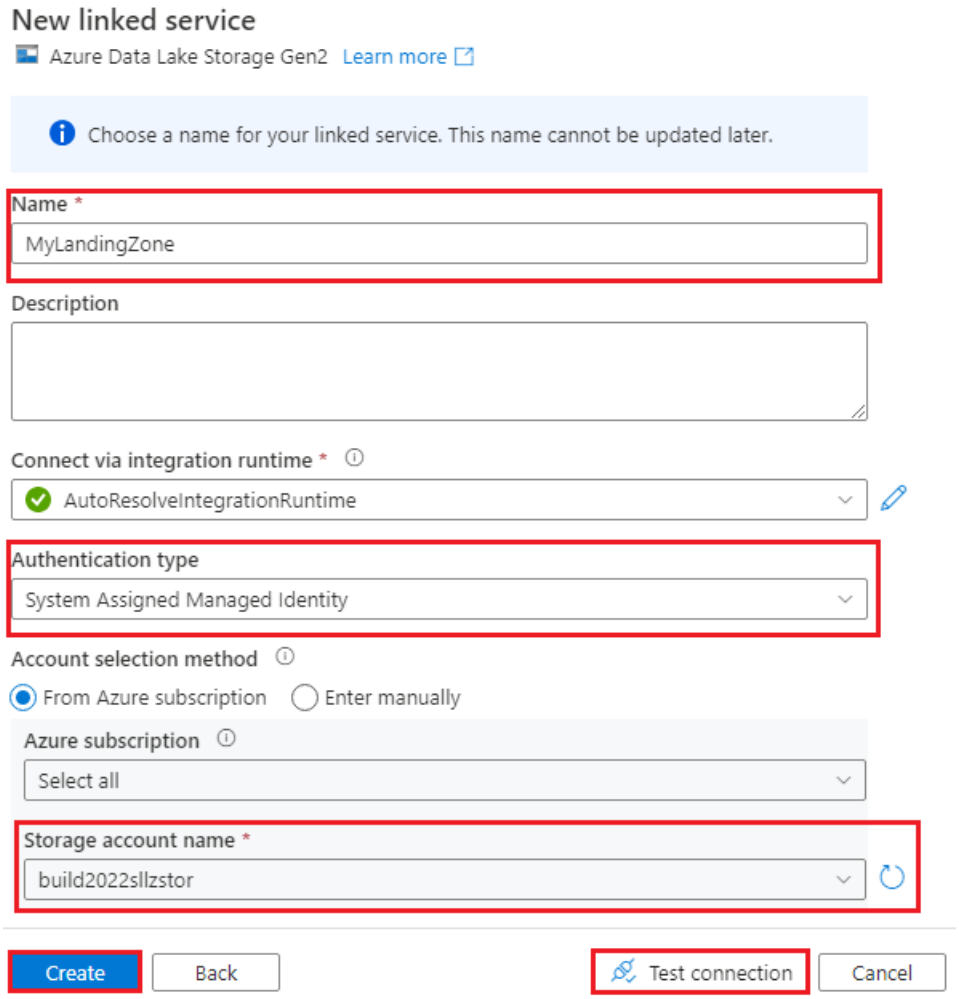


> **NOTE**
>
> Make sure that you have granted your Synapse workspace managed identity permissions to ADLS Gen2 storage account used as the landing zone. For more information, see how to Grant permissions to managed identity in Synapse workspace - Azure Synapse Analytics

3. Open the **Manage** hub in your Synapse workspace, and navigate to **Linked services**.



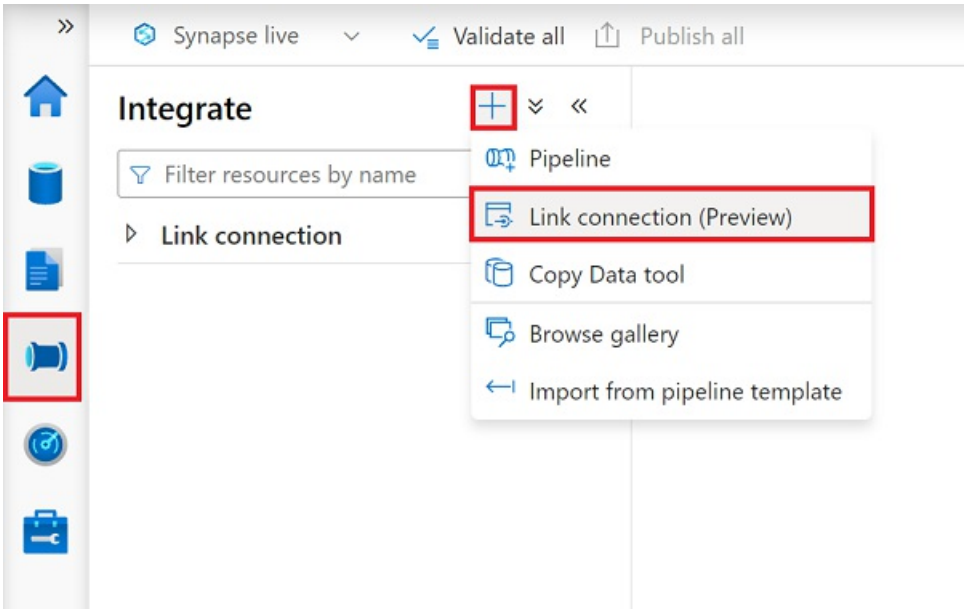
4. Press **+ New** and select **Azure Data Lake Storage Gen2**.
5. Input the following settings:
   - Enter the **name** of linked service for your landing zone.
   - Input **Authentication method**, and it must be **Managed Identity**.
   - Select the **Storage account name** which had already been created.
6. Select **Test Connection** to ensure you get access on your Azure Data Lake Storage Gen2.
7. Select **Create** and you'll have your new linked service connecting to Azure Data Lake Storage Gen2.



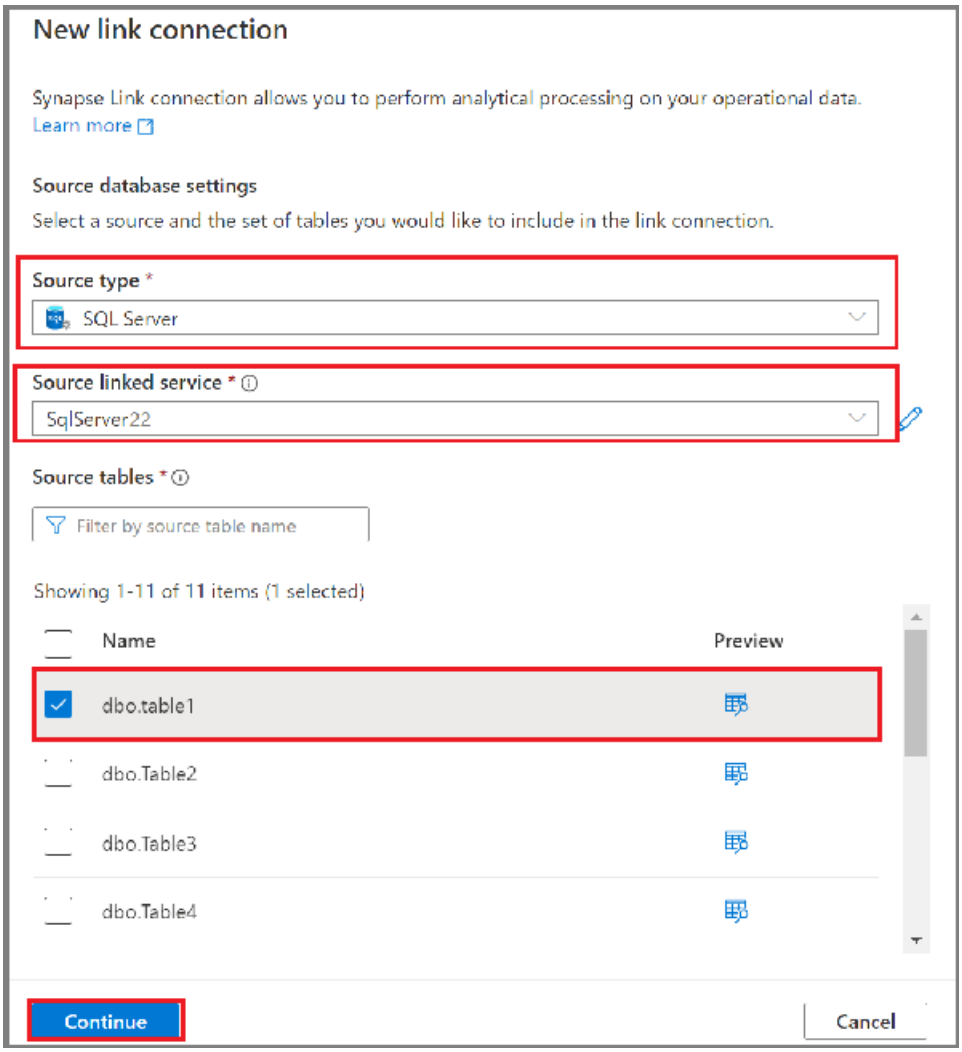
> **NOTE**
>
> The linked service that you create here is not dedicated to Azure Synapse Link for SQL - it can be used by any workspace user that has the appropriate permissions. Please take time to understand the scope of users who may have access to this linked service and its credentials. For more information on permissions in Azure Synapse workspaces, see Azure Synapse workspace access control overview - Azure Synapse Analytics.

Create the Azure Synapse Link connection

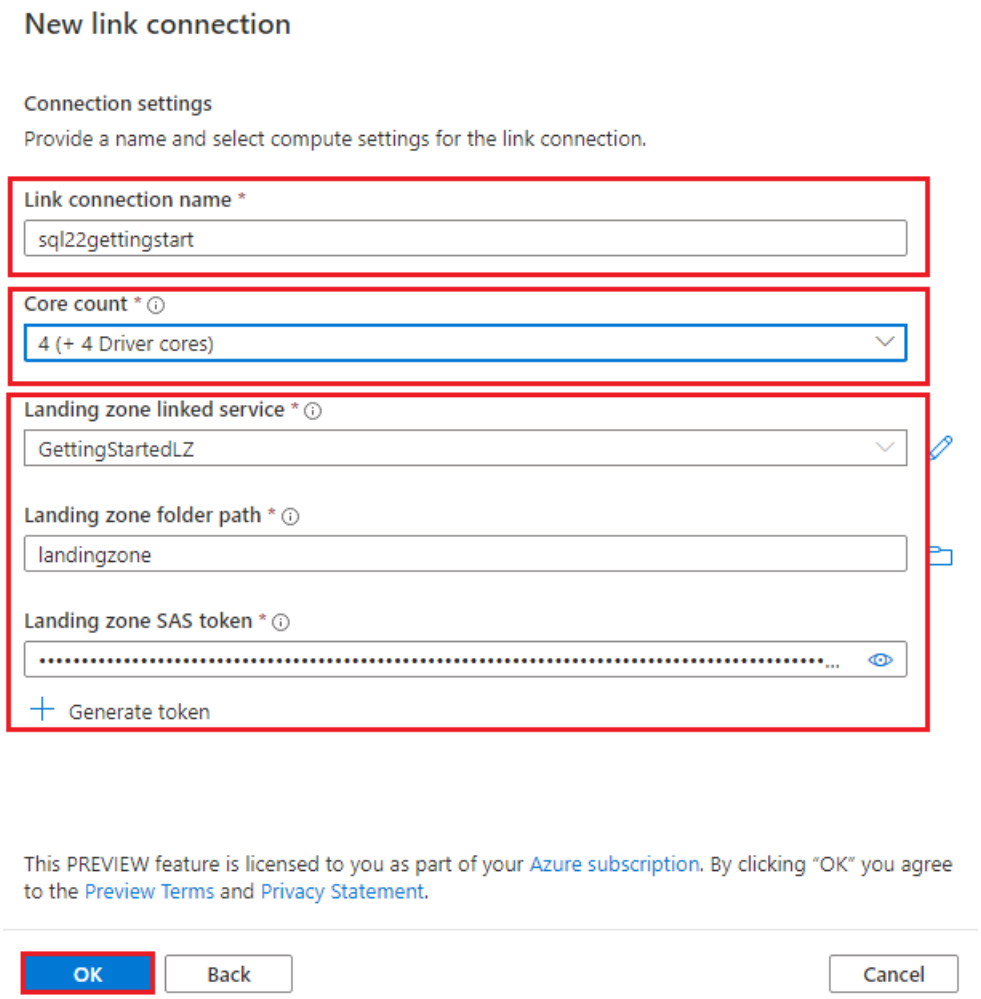
1. From the Synapse studio, open the **Integrate** hub, and select **+Link connection(Preview)**.



2. Input your source database:
   - Select Source type to **SQL Server**.
   - Select your source **linked service** to connect to your SQL Server 2022.
   - Select **table names** from your SQL Server to be replicated to your Synapse SQL pool.
   - Select **Continue**.



3. Select a target database name from **Synapse SQL Dedicated Pools**.
4. Select **Continue**.
5. Input your link connection settings:
   - Input your **link connection name**.
   - Select your **Core count**. We recommend starting from small number and increasing as needed.
   - Configure your landing zone. Select your **linked service** connecting to your landing zone.
   - Input your ADLS Gen2 **container name or container/folder name** as landing zone folder path for staging the data. The container is required to be created first.
   - Input your ADLS Gen2 shared access signature (SAS) token. SAS token is required for SQL change feed to get access on landing zone. If your ADLS Gen2 doesn't have SAS token, you can create one by selecting **+Generate token**.
   - Select **OK**.



6. With the new Azure Synapse Link connection open, you have chance to update the target table name, distribution type and structure type.

> **NOTE**
>
> - Consider heap table for structure type when your data contains varchar(max), nvarchar(max), and varbinary(max).
> - Make sure the schema in your Synapse SQL pool has already been created before you start the link connection. Azure Synapse Link will help you to create tables automatically under your schema in Azure Synapse SQL Pool.

7. Select **Publish all** to save the new link connection to the service.

Start the Azure Synapse Link connection

1. Select **Start** and wait a few minutes for the data to be replicated.

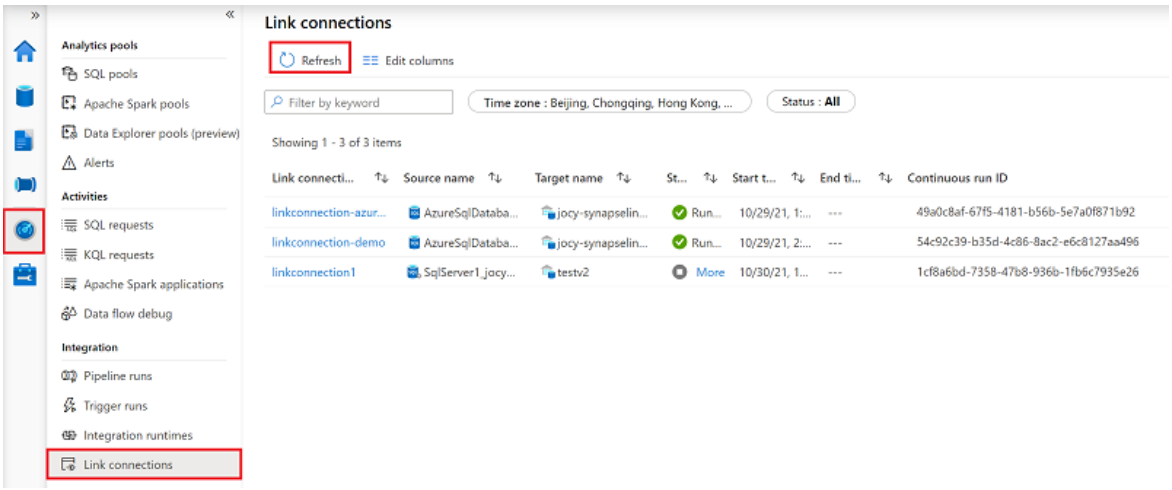
> **NOTE**
>
> When being started, a link connection will start from a full initial load from your source database followed by incremental change feeds via the change feed feature in SQL Server 2022. For more information, see Azure Synapse Link for SQL change feed.

Monitor Azure Synapse Link for SQL Server 2022

You may monitor the status of your Azure Synapse Link connection, see which tables are being initially copied

over (Snapshotting), and see which tables are in continuous replication mode (Replicating).

1. Navigate to the **Monitor hub** of your Synapse workspace.
2. Select **Link connections**.
3. Open the link connection you started and view the status of each table.
4. Select **Refresh** on the monitoring view for your connection to observe any updates to the status.



Query replicated data

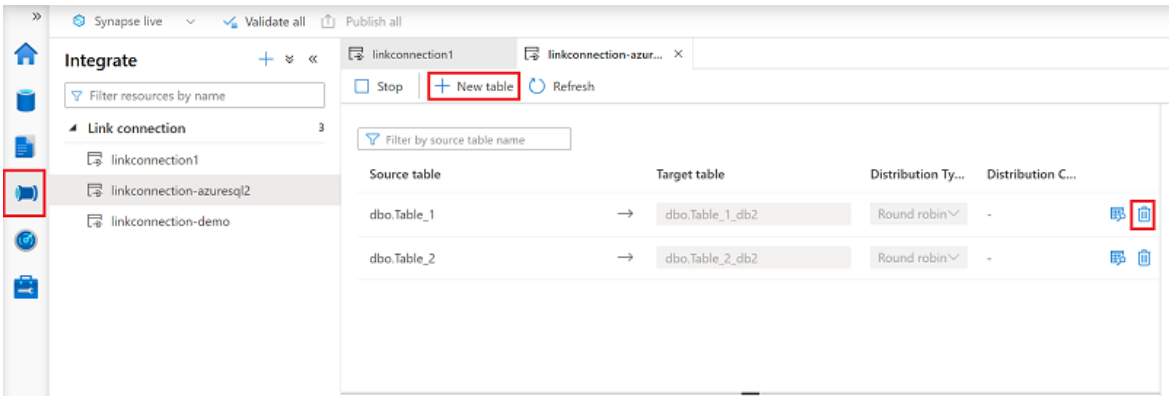
Wait for a few minutes, then check the target database has the expected table and data. See the data available in your Synapse dedicated SQL pool destination store. You can also now explore the replicated tables in your target Synapse dedicated SQL pool.

1. In the **Data** hub, under **Workspace**, open your target database, and within **Tables**, right-click one of your target tables.
2. Choose **New SQL script**, then **Select TOP 100 rows**.
3. Run this query to view the replicated data in your target Synapse dedicated SQL pool.
4. You can also query the target database with SSMS (or other tools). Use the dedicated SQL endpoint for your workspace as the server name. This is typically `<workspacename>.sql.azuresynapse.net`. Add `Database=databasename@poolname` as an extra connection string parameter when connecting via SSMS (or other tools).

Add/remove table in existing Azure Synapse Link connection

You can add/remove tables on Synapse Studio as follows:

1. Open the **Integrate Hub**.
2. Select the **Link connection** you want to edit and open it.
3. Select **+New** table to add tables on Synapse Studio or select the trash can icon to the right of a table to remove an existing table. You can add or remove tables when the link connection is running.



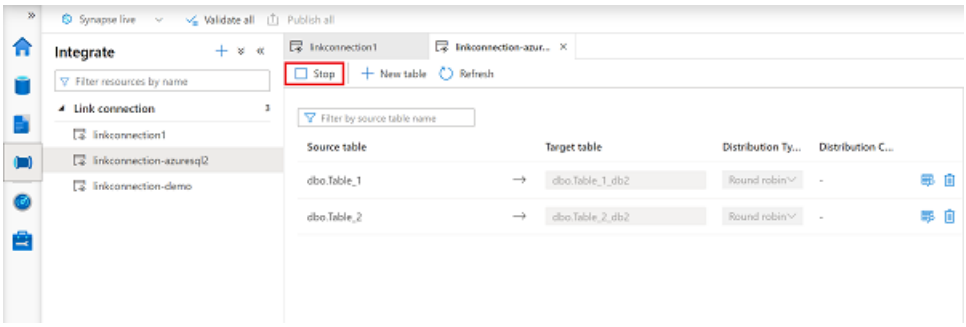
NOTE

You can directly add or remove tables when a link connection is running.

Stop the Azure Synapse Link connection

You can stop the Azure Synapse Link connection on Synapse Studio as follows:

1. Open the **Integrate Hub** of your Synapse workspace.
2. Select the **Link connection** you want to edit and open it.
3. Select **Stop** to stop the link connection, and it will stop replicating your data.



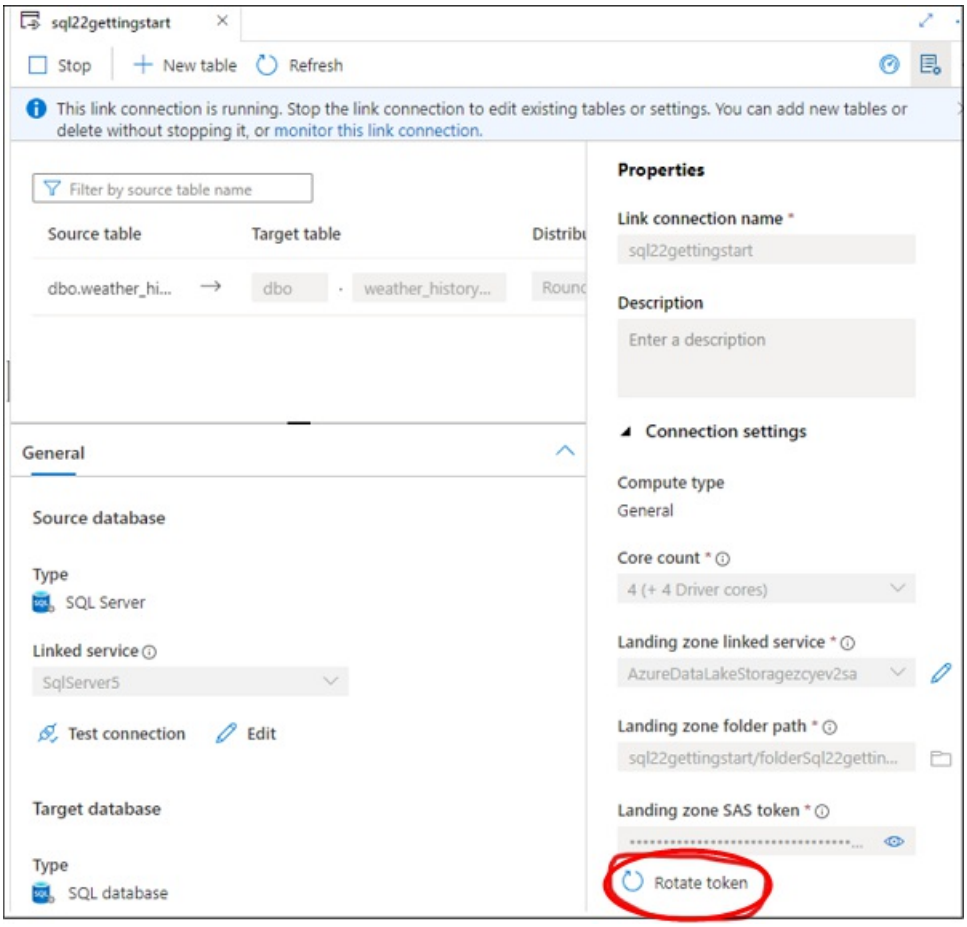
NOTE

If you restart a link connection after stopping it, it will start from a full initial load from your source database followed by incremental change feeds.

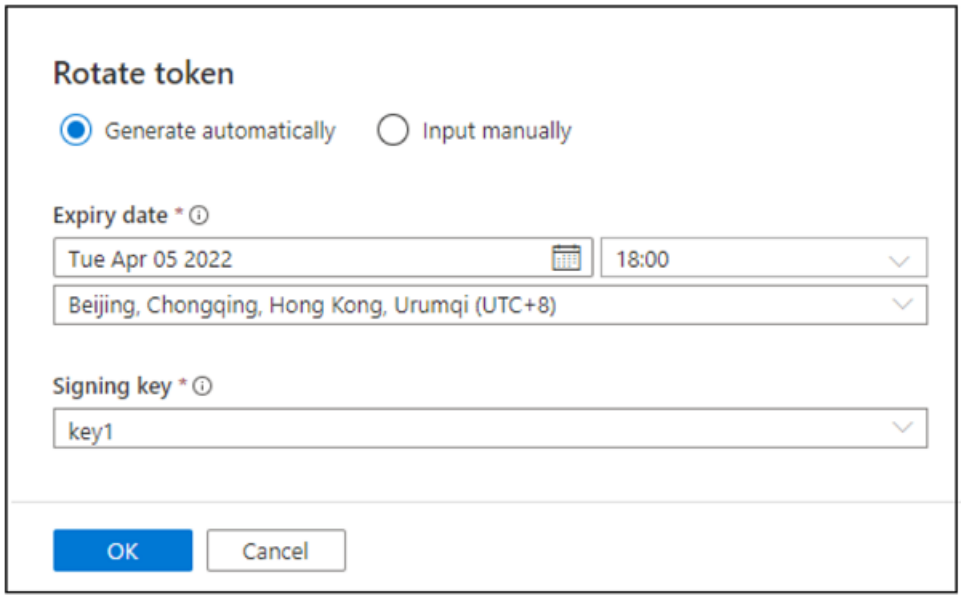
Rotate the SAS token for landing zone

A SAS token is required for SQL change feed to get access to the landing zone and push data there. It has an expiration date so you need to rotate the SAS token before the expiration date. Otherwise, Azure Synapse Link will fail to replicate the data from SQL Server to the Synapse dedicated SQL pool.

1. Open the **Integrate Hub** of your Synapse workspace.
2. Select the **Link connection** you want to edit and open it.
3. Select **Rotate token**.



4. Select **Generate automatically** or **Input manually** to get the new SAS token, and then select **OK**.



Next steps

If you are using a different type of database, see how to:

- Configure Azure Synapse Link for Azure Cosmos DB
- Configure Azure Synapse Link for Dataverse
- Get started with Azure Synapse Link for Azure SQL Database

Known limitations and issues with Azure Synapse Link for SQL

8/19/2022 • 5 minutes to read • Edit Online

This article lists the known limitations and issues with Azure Synapse Link for SQL.

Known limitations

This is the list of known limitations for Azure Synapse Link for SQL.

Azure SQL DB and SQL Server 2022

- Users must use an Azure Synapse Analytics workspace created on or after May 24, 2022, to get access to Azure Synapse Link for SQL functionality.
- Running Azure Synapse Analytics in a managed virtual network isn't supported. Users need to check "Disable Managed virtual network" and "Allow connections from all IP addresses" when creating their workspace.
- Source tables must have primary keys.
- The following data types aren't supported for primary keys in the source tables:
  - real
  - float
  - hierarchyid
  - sql_variant
  - timestamp
- Source table row size can't exceed 7,500 bytes. For tables where variable-length columns are stored off-row, a 24-byte pointer is stored in the main record.
- Tables enabled for Azure Synapse Link for SQL can have a maximum of 1,020 columns (not 1,024).
- While a database can have multiple links enabled, a given table can't belong to multiple links.
- When a database owner doesn't have a mapped log in, Azure Synapse link for SQL will run into an error when enabling a link connection. User can set database owner to a valid user with the `ALTER AUTHORIZATION` command to fix this issue.
- If the source table contains computed columns or columns with data types that aren't supported by Azure Synapse Analytics dedicated SQL pools, these columns won't be replicated to Azure Synapse Analytics. Unsupported columns include:
  - image
  - text
  - xml
  - timestamp
  - sql_variant
  - UDT
  - geometry
  - geography
- A maximum of 5,000 tables can be added to a single link connection.
- When a source column is of type datetime2(7) or time(7), the last digit will be truncated when data is replicated to Azure Synapse Analytics.
- The following table DDL operations aren't allowed on source tables when they are enabled for Azure Synapse Link for SQL. All other DDL operations are allowed, but they won't be replicated to Azure Synapse Analytics.

    - Switch Partition
    - Add/Drop/Alter Column
    - Alter Primary Key
    - Drop/Truncate Table
    - Rename Table
- If DDL + DML is executed in an explicit transaction (between `BEGIN TRANSACTION` and `END TRANSACTION` statements), replication for corresponding tables will fail within the link connection.

  > **NOTE**
  > If a table is critical for transactional consistency at the link connection level, please review the state of the Azure Synapse Link table in the Monitoring tab.

- Azure Synapse Link for SQL can't be enabled if any of the following features are in use for the source table:
    - Change Data Capture
    - Temporal history table
    - Always encrypted
    - In-Memory OLTP
    - Column Store Index
    - Graph
- System tables can't be replicated.
- The security configuration from the source database will **NOT** be reflected in the target dedicated SQL pool.
- Enabling Azure Synapse Link for SQL will create a new schema called `changefeed`. Don't use this schema, as it is reserved for system use.
- Source tables with non-default collations: UTF8, Japanese can't be replicated to Synapse. Here's the supported collations in Synapse SQL Pool.
- Single row updates (including off-page storage) of > 370MB are not supported.

Azure SQL DB only

- Azure Synapse Link for SQL isn't supported on Free, Basic or Standard tier with fewer than 100 DTUs.
- Azure Synapse Link for SQL isn't supported on SQL Managed Instances.
- Users need to check "Allow Azure services and resources to access this server" in the firewall settings of their source database server.
- Service principal isn't supported for authenticating to source Azure SQL DB, so when creating Azure SQL DB linked Service, choose SQL authentication, user-assigned managed identity (UAMI) or service assigned managed Identity (SAMI).
- Azure Synapse Link can't be enabled on the secondary database once a GeoDR failover has happened if the secondary database has a different name from the primary database.
- If you enabled Azure Synapse Link for SQL on your database as an Microsoft Azure Active Directory (Azure AD) user, Point-in-time restore (PITR) will fail. PITR will only work when you enable Azure Synapse Link for SQL on your database as a SQL user.
- If you create a database as an Azure AD user and enable Azure Synapse Link for SQL, a SQL authentication user (for example, even sysadmin role) won't be able to disable/make changes to Azure Synapse Link for SQL artifacts. However, another Azure AD user will be able to enable/disable Azure Synapse Link for SQL on the same database. Similarly, if you create a database as an SQL authentication user, enabling/disabling Azure Synapse Link for SQL as an Azure AD user won't work.
- When enabling Azure Synapse Link for SQL on your Azure SQL Database, you should ensure that aggressive log truncation is disabled.

SQL Server 2022 only

- Azure Synapse Link for SQL works with SQL Server on Linux, but HA scenarios with Linux Pacemaker aren't supported. Shelf hosted IR cannot be installed on Linux environment.
- Azure Synapse Link for SQL can't be enabled on databases that are transactional replication publishers or distributors.
- If the SAS key of landing zone expires and gets rotated during the snapshot process, the new key won't get picked up. The snapshot will fail and restart automatically with the new key.
- Prior to breaking an Availability Group, disable any running links. Otherwise both databases will attempt to write their changes to the landing zone.
- When using asynchronous replicas, transactions need to be written to all replicas prior to them being published to Azure Synapse Link for SQL.
- Azure Synapse Link for SQL isn't supported on databases with database mirroring enabled.
- Restoring an Azure Synapse Link for SQL-enabled database from on-premises to Azure SQL Managed Instance isn't supported.
- Azure Synapse Link for SQL is not supported on databases that are using Managed Instance Link.

Known issues

Deleting an Azure Synapse Analytics workspace with a running link could cause log in source database to fill

- Applies To - Azure SQL Database and SQL Server 2022
- Issue - When you delete an Azure Synapse Analytics workspace it is possible that running links might not be stopped, which will cause the source database to think that the link is still operational and could lead to the log filling and not being truncated.
- Resolution - There are two possible resolutions to this situation:

1. Stop any running links prior to deleting the Azure Synapse Analytics workspace.
2. Manually clean up the link definition in the source database.
   a. Find the table_group_id for the link(s) that need to be stopped using the following query:

```
SELECT table_group_id, workspace_id, synapse_workgroup_name
FROM [changefeed].[change_feed_table_groups]
WHERE synapse_workgroup_name = <synapse workspace name>
```

   b. Drop each link identified using the following procedure:

```
EXEC sys.sp_change_feed_drop_table_group @table_group_id = <table_group_id>
```

   c. Optionally, if you are disabling all of the table groups for a given database, you can also disable change feed on the database with the following command:

```
EXEC sys.sp_change_feed_disable_db
```

Next steps

If you are using a different type of database, see how to:

- Configure Synapse Link for Azure Cosmos DB
- Configure Synapse Link for Dataverse

Quickstart: Load data into dedicated SQL pool using the copy activity

8/19/2022 • 5 minutes to read • Edit Online

Azure Synapse Analytics offers various analytics engines to help you ingest, transform, model, and analyze your data. A dedicated SQL pool offers T-SQL based compute and storage capabilities. After creating a dedicated SQL pool in your Synapse workspace, data can be loaded, modeled, processed, and delivered for faster analytic insight.

In this quickstart, you learn how to *load data from Azure SQL Database into Azure Synapse Analytics*. You can follow similar steps to copy data from other types of data stores. This similar flow applies to data copy for other sources and sinks as well.

Prerequisites

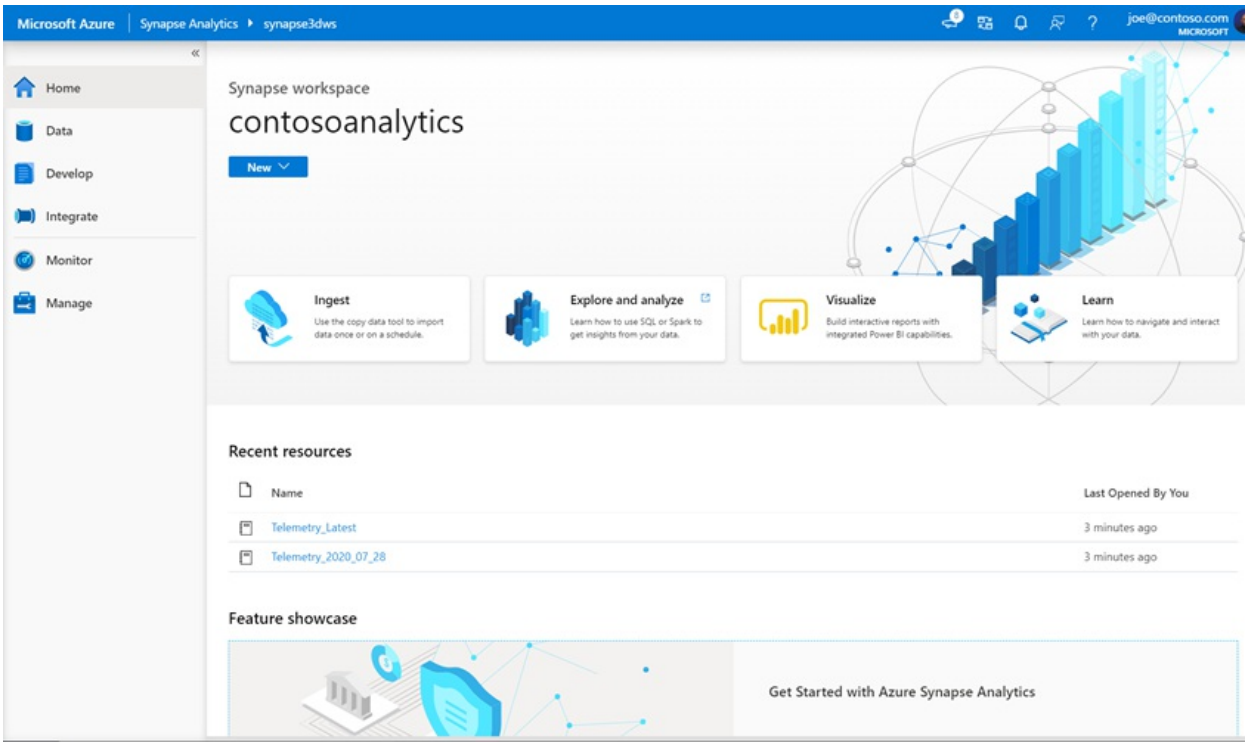
- Azure subscription: If you don't have an Azure subscription, create a free account before you begin.
- Azure Synapse workspace: Create a Synapse workspace using the Azure portal following the instructions in Quickstart: Create a Synapse workspace.
- Azure SQL Database: This tutorial copies data from the Adventure Works LT sample dataset in Azure SQL Database. You can create this sample database in SQL Database by following the instructions in Create a sample database in Azure SQL Database. Or you can use other data stores by following similar steps.
- Azure storage account: Azure Storage is used as the *staging* area in the copy operation. If you don't have an Azure storage account, see the instructions in Create a storage account.
- Azure Synapse Analytics: You use a dedicated SQL pool as a sink data store. If you don't have an Azure Synapse Analytics instance, see Create a dedicated SQL pool for steps to create one.

Navigate to the Synapse Studio

After your Synapse workspace is created, you have two ways to open Synapse Studio:

- Open your Synapse workspace in the Azure portal. Select **Open** on the Open Synapse Studio card under Getting started.
- Open Azure Synapse Analytics and sign in to your workspace.

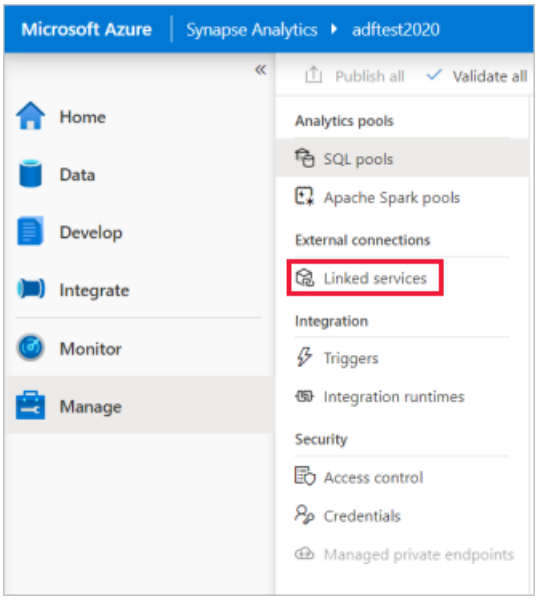
In this quickstart, we use the workspace named "adftest2020" as an example. It will automatically navigate you to the Synapse Studio home page.



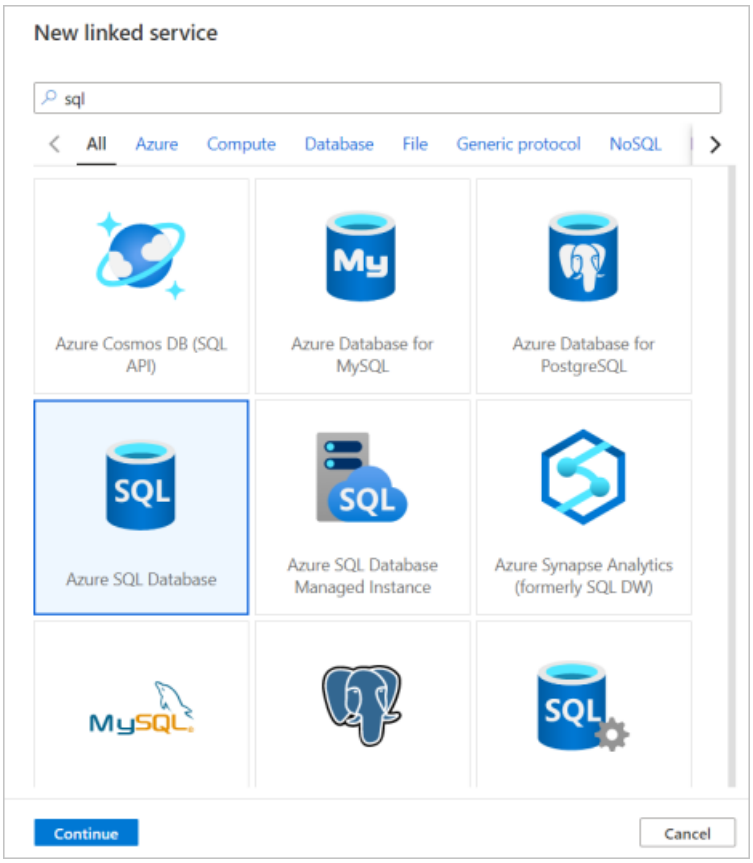
Create linked services

In Azure Synapse Analytics, a linked service is where you define your connection information to other services. In this section, you'll create following two kinds of linked services: Azure SQL Database and Azure Data Lake Storage Gen2 (ADLS Gen2) linked services.

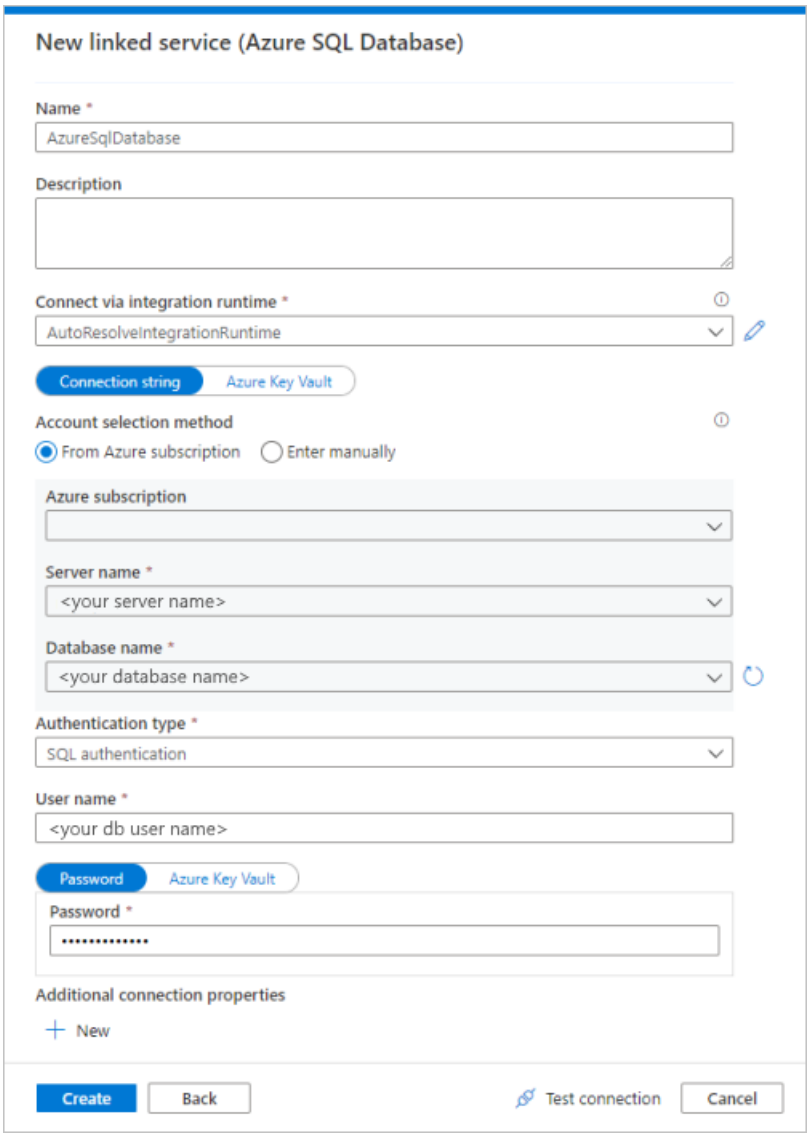
1. On the Synapse Studio home page, select the **Manage** tab in the left navigation.
2. Under External connections, select Linked services.



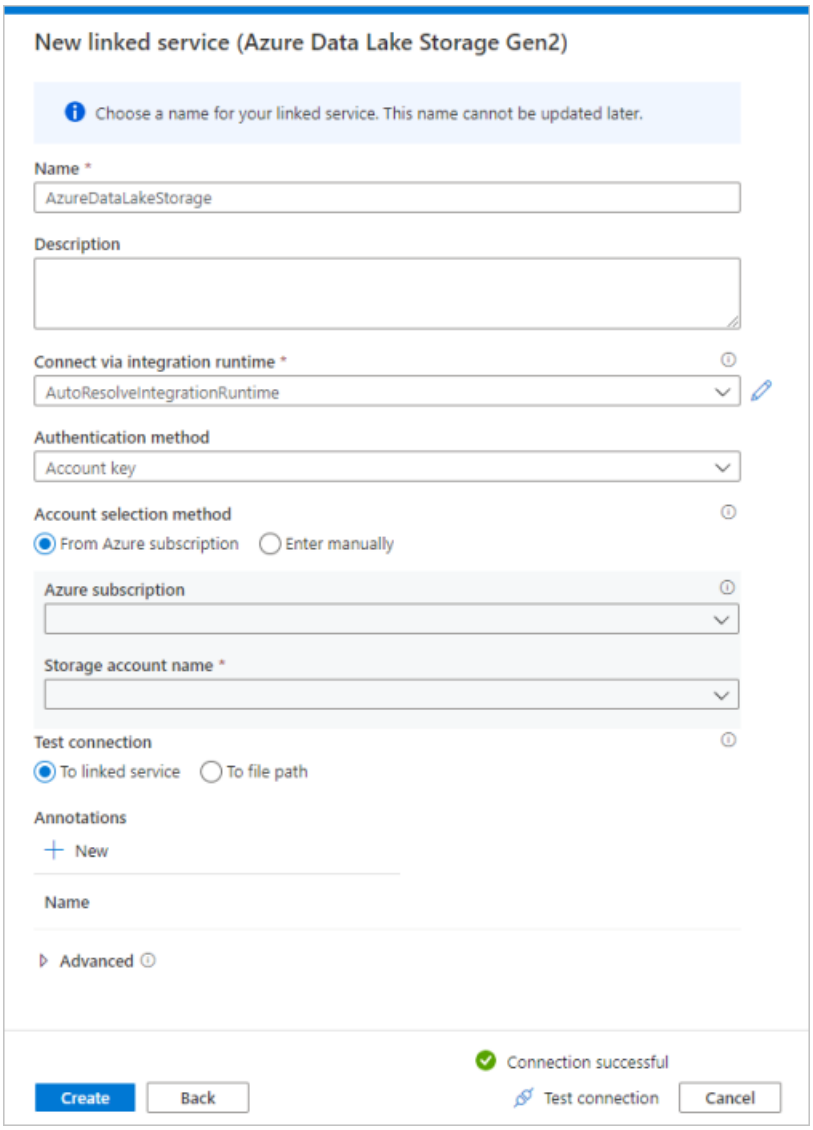
3. To add a linked service, select **New**.
4. Select **Azure SQL Database** from the gallery, and then select **Continue**. You can type "sql" in the search box to filter the connectors.



5. In the New Linked Service page, select your server name and DB name from the dropdown list, and specify the username and password. Click **Test connection** to validate the settings, then select **Create**.



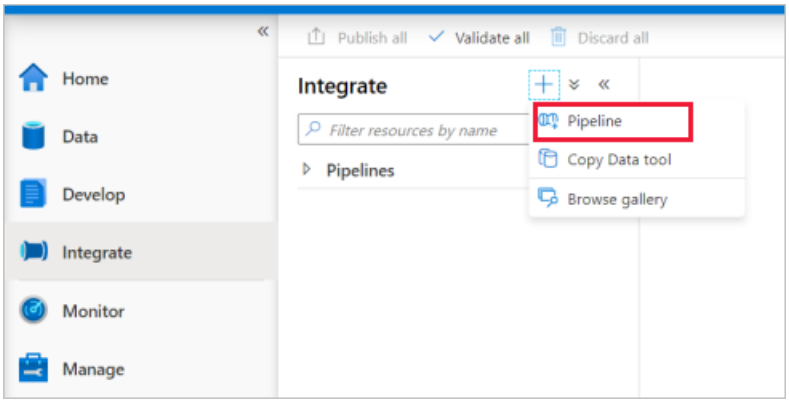
6. Repeat steps 3-4, but select **Azure Data Lake Storage Gen2** instead from the gallery. In the New Linked Service page, select your storage account name from the dropdown list. Click **Test connection** to validate the settings, then select **Create**.



Create a pipeline

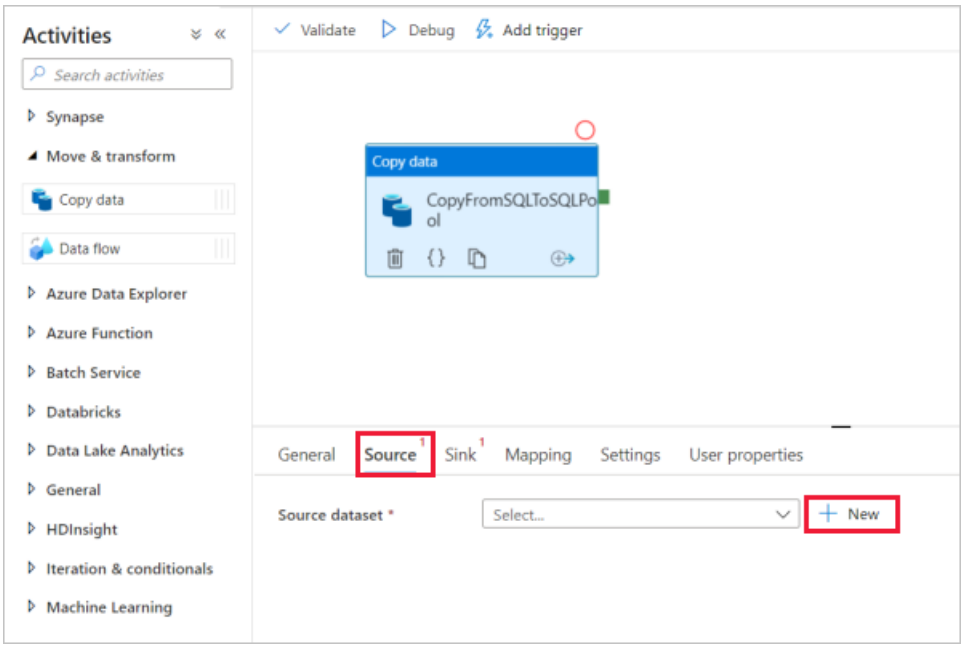
A pipeline contains the logical flow for an execution of a set of activities. In this section, you'll create a pipeline containing a copy activity that ingests data from Azure SQL Database into a dedicated SQL pool.

1. Go to the **Integrate** tab. Select on the plus icon next to the pipelines header and select Pipeline.



2. Under *Move and Transform* in the *Activities* pane, drag **Copy data** onto the pipeline canvas.

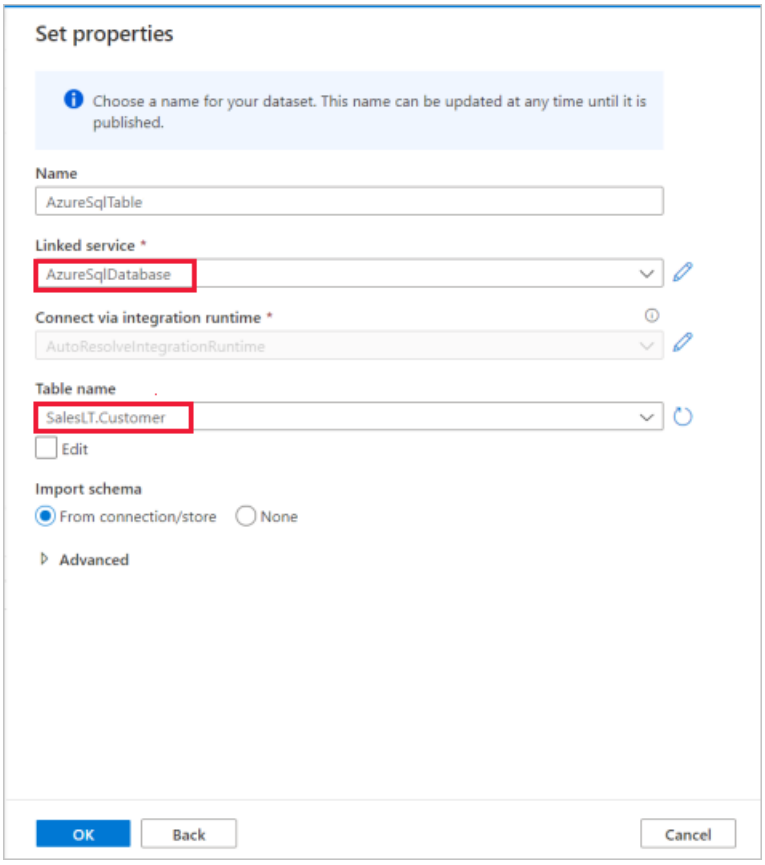
3. Select on the copy activity and go to the Source tab. Select **New** to create a new source dataset.



4. Select **Azure SQL Database** as your data store and select **Continue**.

5. In the *Set properties* pane, select the Azure SQL Database linked service you created in earlier step.

6. Under Table name, select a sample table to use in following copy activity. In this quickstart, we use "SalesLT.Customer" table as an example.



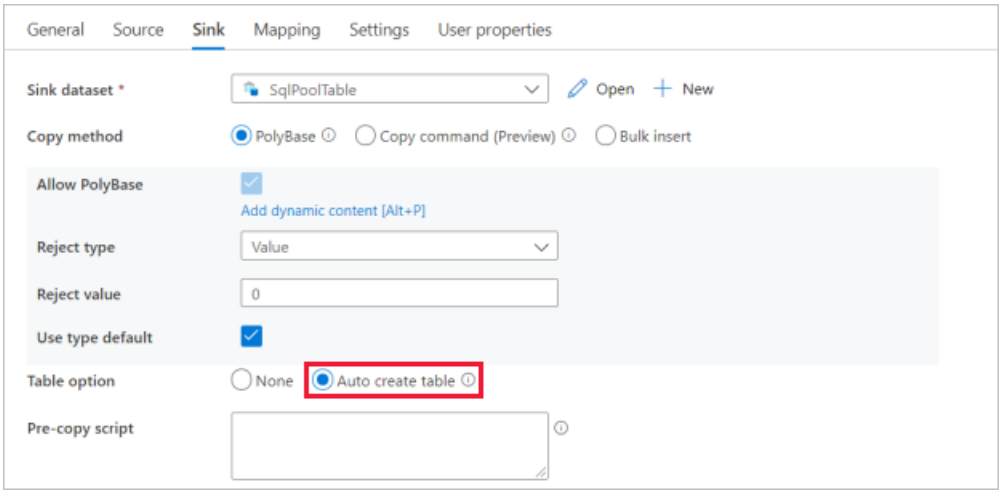
7. Select **OK** when finished.

8. Select the copy activity and go to the Sink tab. Select **New** to create a new sink dataset.

9. Select **Azure Synapse dedicated SQL pool** as your data store and select **Continue**.

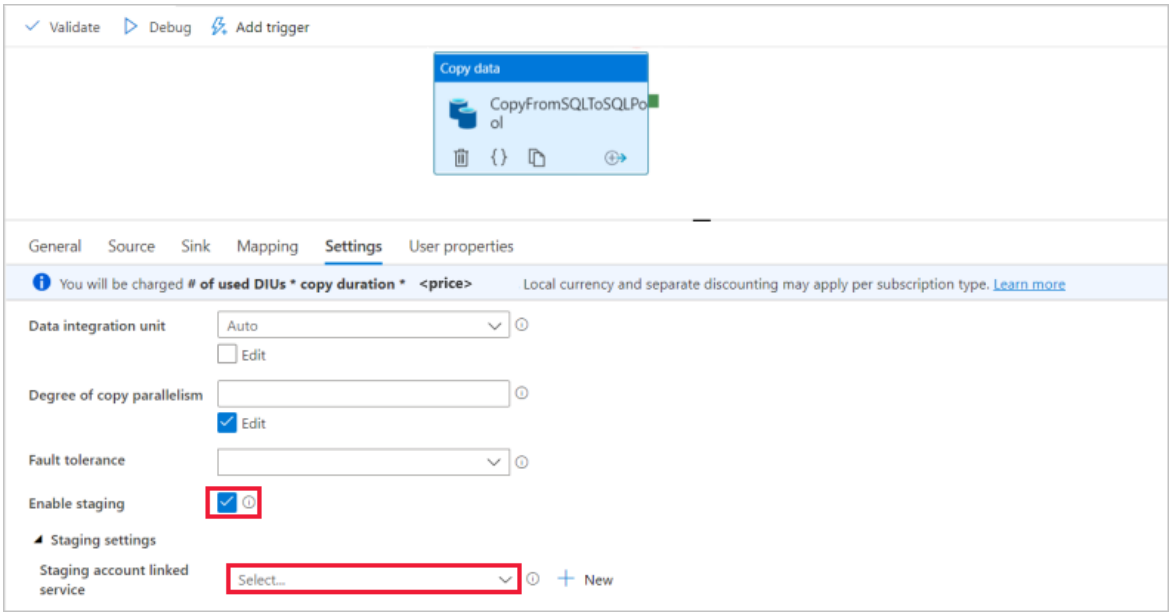
10. In the **Set properties** pane, select the SQL Analytics pool you created in earlier step. If you're writing to an existing table, under *Table name* select it from the dropdown. Otherwise, check "Edit" and enter in your new table name. Select **OK** when finished.

11. For Sink dataset settings, enable **Auto create table** in the Table option field.



12. In the **Settings** page, select the checkbox for **Enable staging**. This option applies if your source data is not compatible with PolyBase. In **Staging settings** section, select the Azure Data Lake Storage Gen2 linked service you created in earlier step as the staging storage.

    The storage is used for staging the data before it loads into Azure Synapse Analytics by using PolyBase. After the copy is complete, the interim data in Azure Data Lake Storage Gen2 is automatically cleaned up.

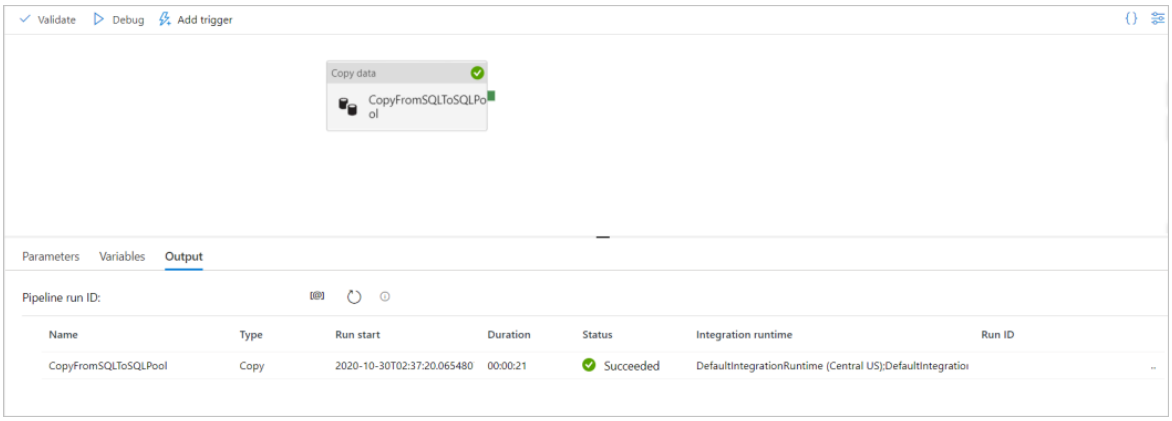


13. To validate the pipeline, select **Validate** on the toolbar. You see the result of the Pipeline validation output on the right side of the page.

Debug and publish the pipeline

Once you've finished configuring your pipeline, you can execute a debug run before you publish your artifacts to verify everything is correct.

1. To debug the pipeline, select **Debug** on the toolbar. You see the status of the pipeline run in the **Output** tab at the bottom of the window.



2. Once the pipeline run succeeds, in the top toolbar, select **Publish all**. This action publishes entities (datasets, and pipelines) you created to the Synapse Analytics service.

3. Wait until you see the **Successfully published** message. To see notification messages, select the bell button on the top-right.

Trigger and monitor the pipeline

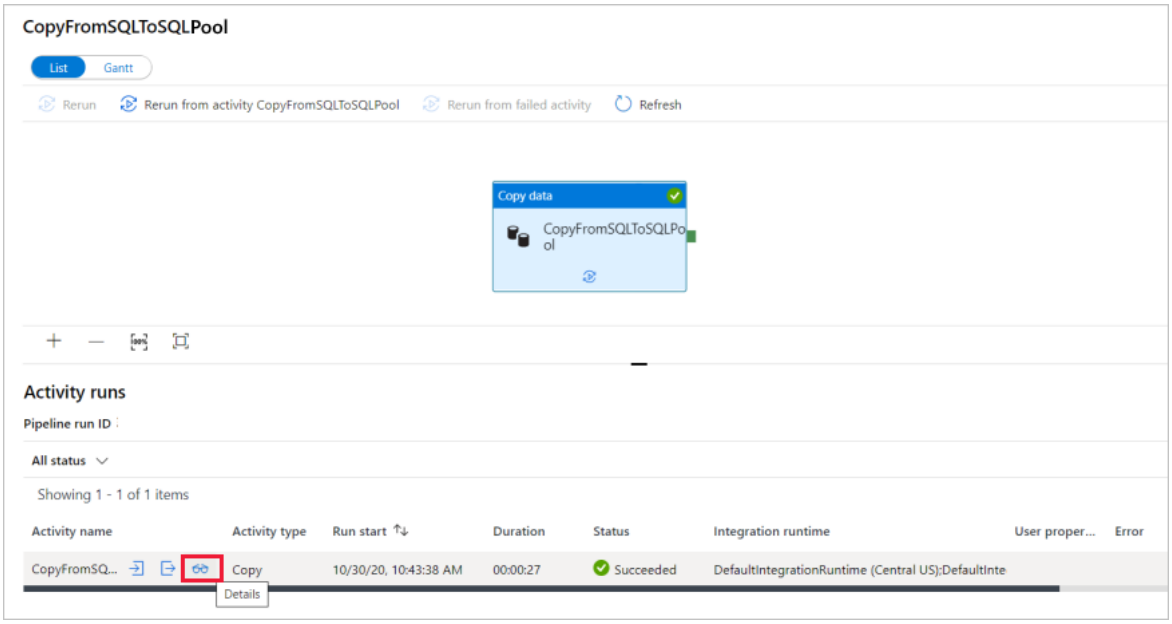
In this section, you manually trigger the pipeline published in the previous step.

1. Select **Add Trigger** on the toolbar, and then select **Trigger Now**. On the **Pipeline Run** page, select **OK**.

2. Go to the **Monitor** tab located in the left sidebar. You see a pipeline run that is triggered by a manual trigger.

3. When the pipeline run completes successfully, select the link under the **Pipeline name** column to view activity run details or to rerun the pipeline. In this example, there's only one activity, so you see only one entry in the list.

4. For details about the copy operation, select the **Details** link (eyeglasses icon) under the **Activity name** column. You can monitor details like the volume of data copied from the source to the sink, data throughput, execution steps with corresponding duration, and used configurations.



5. To switch back to the pipeline runs view, select the **All pipeline runs** link at the top. Select **Refresh** to refresh the list.

6. Verify your data is correctly written in the dedicated SQL pool.

Next steps

Advance to the following article to learn about Azure Synapse Analytics support:

Pipeline and activities Connector overview Copy activity

Quickstart: Transform data using mapping data flows

8/19/2022 • 7 minutes to read • Edit Online

In this quickstart, you'll use Azure Synapse Analytics to create a pipeline that transforms data from an Azure Data Lake Storage Gen2 (ADLS Gen2) source to an ADLS Gen2 sink using mapping data flow. The configuration pattern in this quickstart can be expanded upon when transforming data using mapping data flow

In this quickstart, you do the following steps:

- Create a pipeline with a Data Flow activity in Azure Synapse Analytics.
- Build a mapping data flow with four transformations.
- Test run the pipeline.
- Monitor a Data Flow activity

Prerequisites

- **Azure subscription**: If you don't have an Azure subscription, create a free account before you begin.
- **Azure Synapse workspace**: Create a Synapse workspace using the Azure portal following the instructions in Quickstart: Create a Synapse workspace.
- **Azure storage account**: You use ADLS storage as *source* and *sink* data stores. If you don't have a storage account, see Create an Azure storage account for steps to create one.

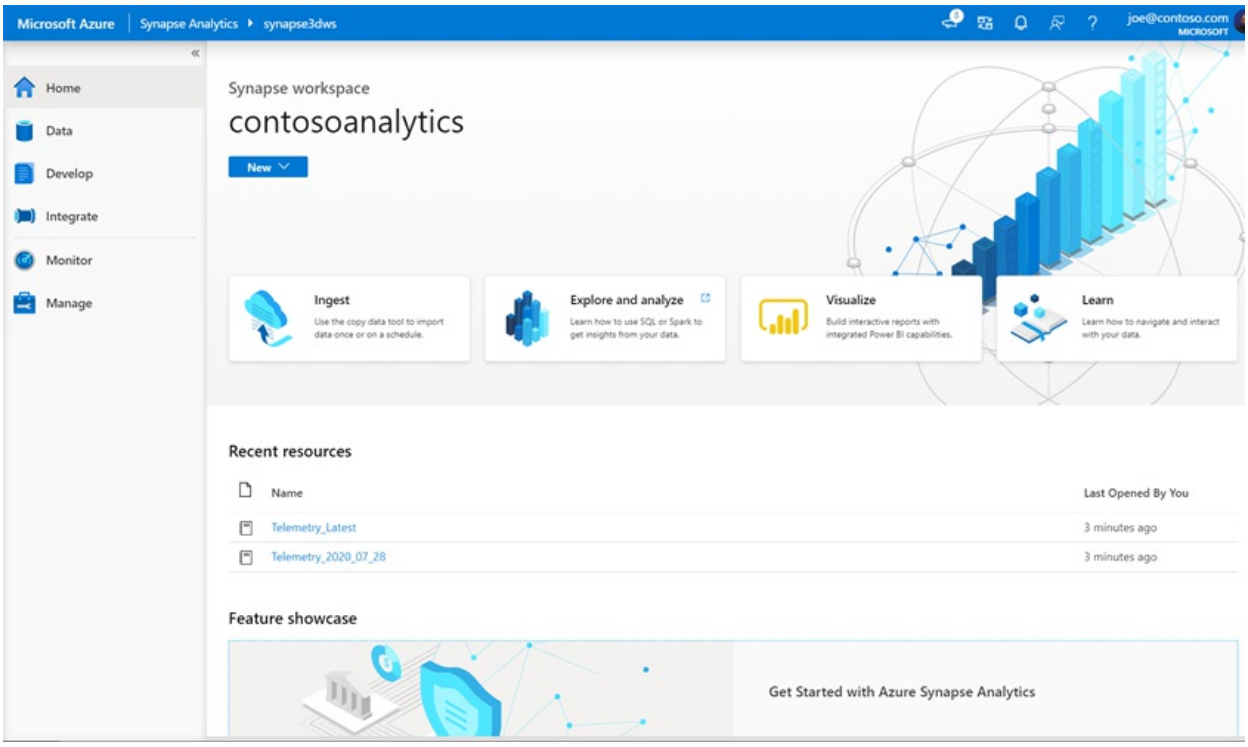
  The file that we are transforming in this tutorial is MoviesDB.csv, which can be found here. To retrieve the file from GitHub, copy the contents to a text editor of your choice to save locally as a .csv file. To upload the file to your storage account, see Upload blobs with the Azure portal. The examples will be referencing a container named 'sample-data'.

Navigate to the Synapse Studio

After your Azure Synapse workspace is created, you have two ways to open Synapse Studio:

- Open your Synapse workspace in the Azure portal. Select **Open** on the Open Synapse Studio card under Getting started.
- Open Azure Synapse Analytics and sign in to your workspace.

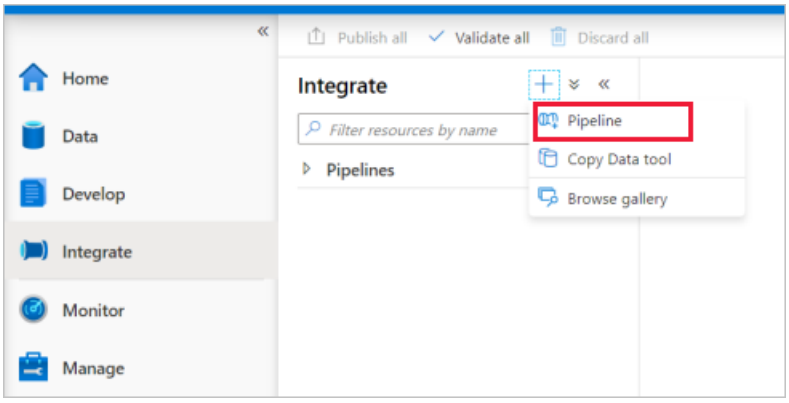
In this quickstart, we use the workspace named "adftest2020" as an example. It will automatically navigate you to the Synapse Studio home page.



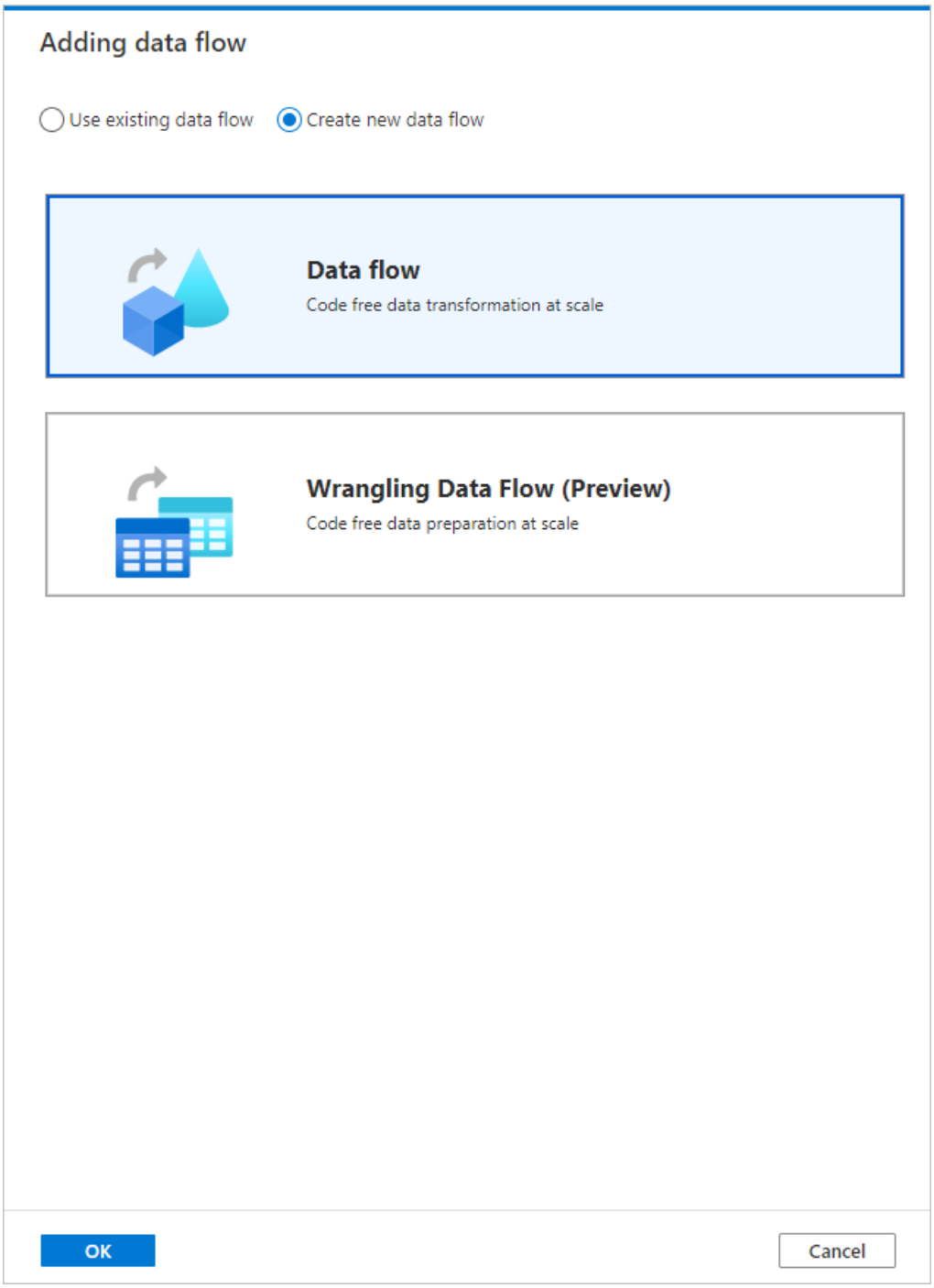
Create a pipeline with a Data Flow activity

A pipeline contains the logical flow for an execution of a set of activities. In this section, you'll create a pipeline that contains a Data Flow activity.

1. Go to the **Integrate** tab. Select on the plus icon next to the pipelines header and select Pipeline.



2. In the **Properties** settings page of the pipeline, enter **TransformMovies** for **Name**.
3. Under *Move and Transform* in the *Activities* pane, drag **Data flow** onto the pipeline canvas.
4. In the **Adding data flow** page pop-up, select **Create new data flow** -> **Data flow**. Click **OK** when done.

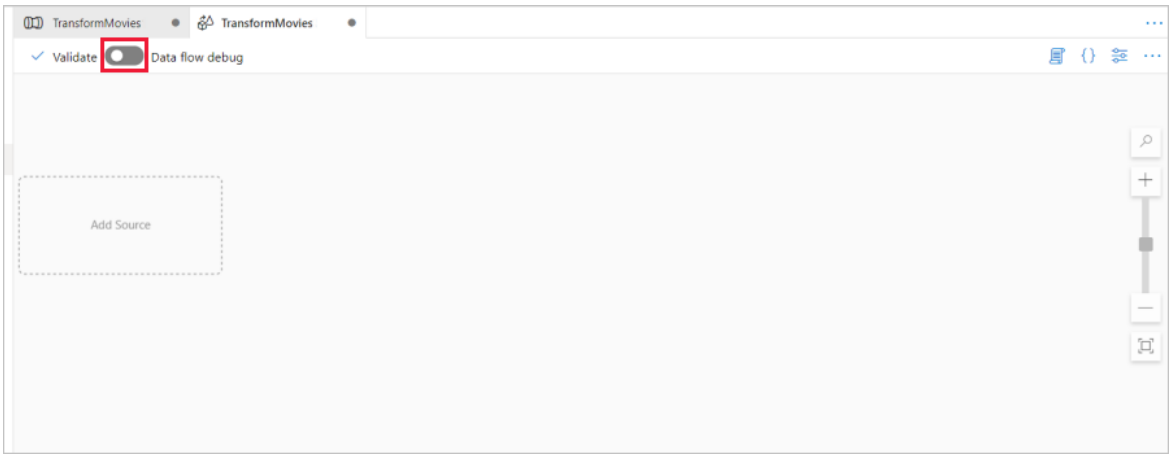


5. Name your data flow **TransformMovies** on the **Properties** page.

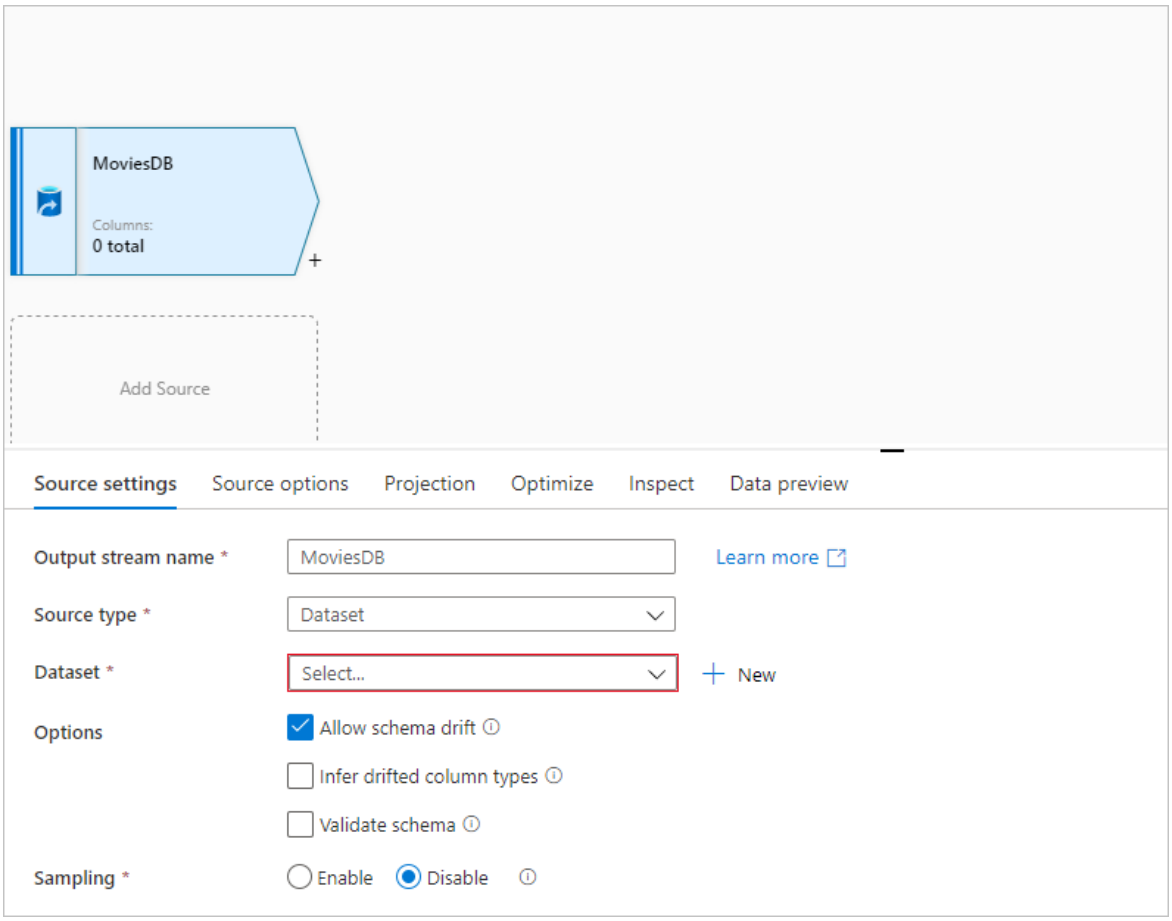
Build transformation logic in the data flow canvas

Once you create your Data Flow, you'll be automatically sent to the data flow canvas. In this step, you'll build a data flow that takes the MoviesDB.csv in ADLS storage and aggregates the average rating of comedies from 1910 to 2000. You'll then write this file back to the ADLS storage.

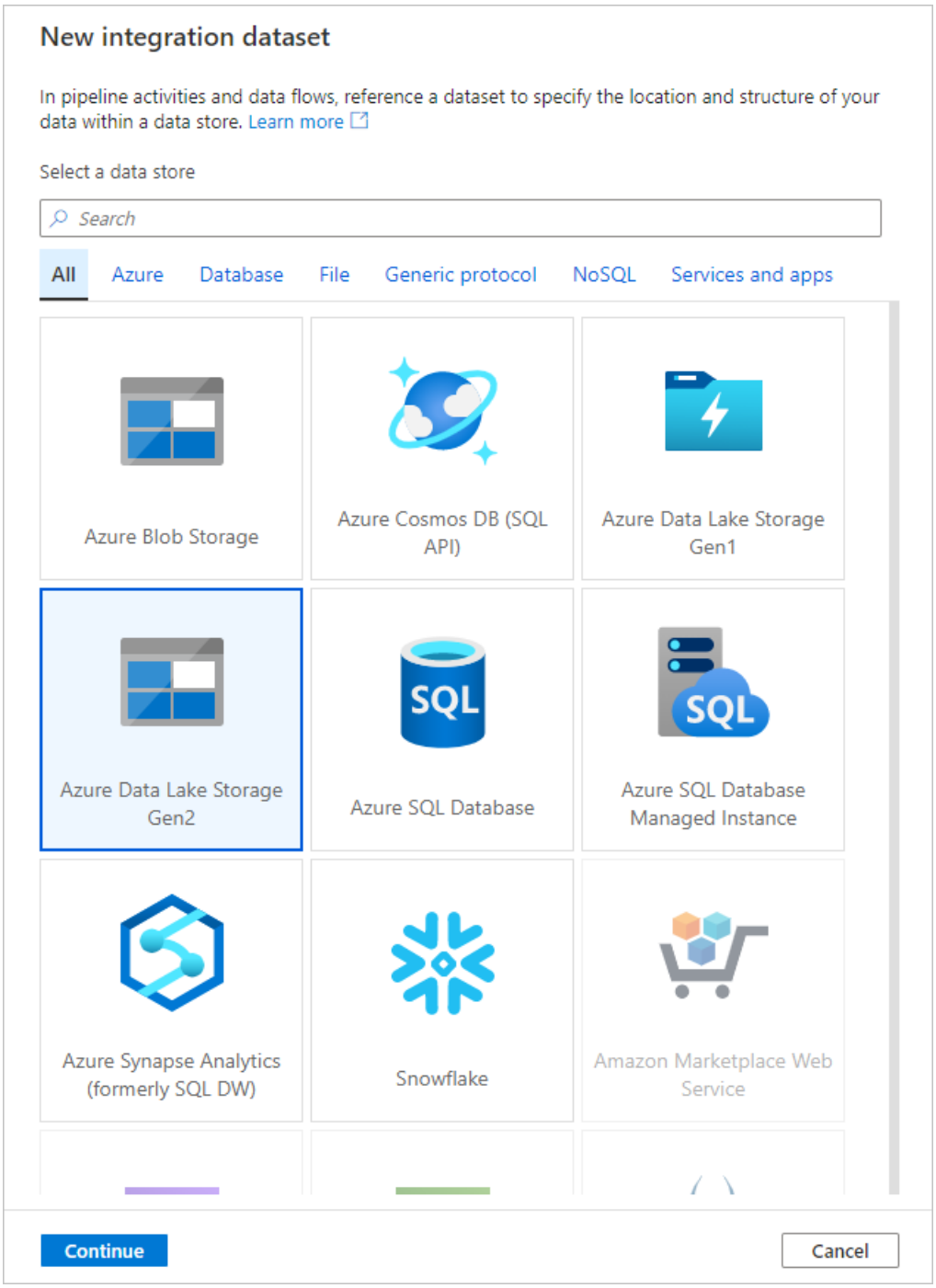
1. Above the data flow canvas, slide the **Data flow debug** slider on. Debug mode allows for interactive testing of transformation logic against a live Spark cluster. Data Flow clusters take 5-7 minutes to warm up and users are recommended to turn on debug first if they plan to do Data Flow development. For more information, see Debug Mode.



2. In the data flow canvas, add a source by clicking on the **Add Source** box.
3. Name your source **MoviesDB**. Click on **New** to create a new source dataset.



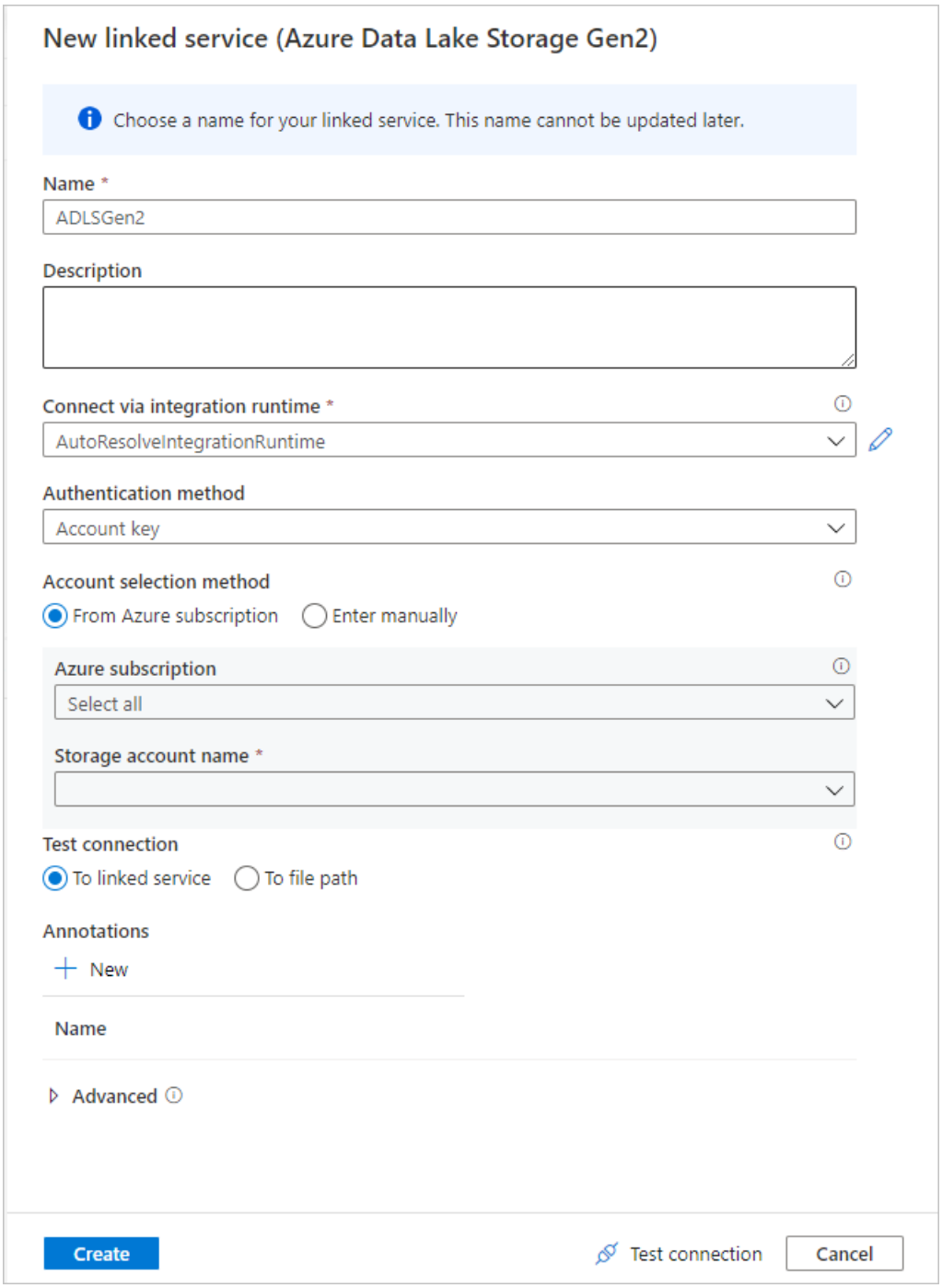
4. Choose **Azure Data Lake Storage Gen2**. Click Continue.



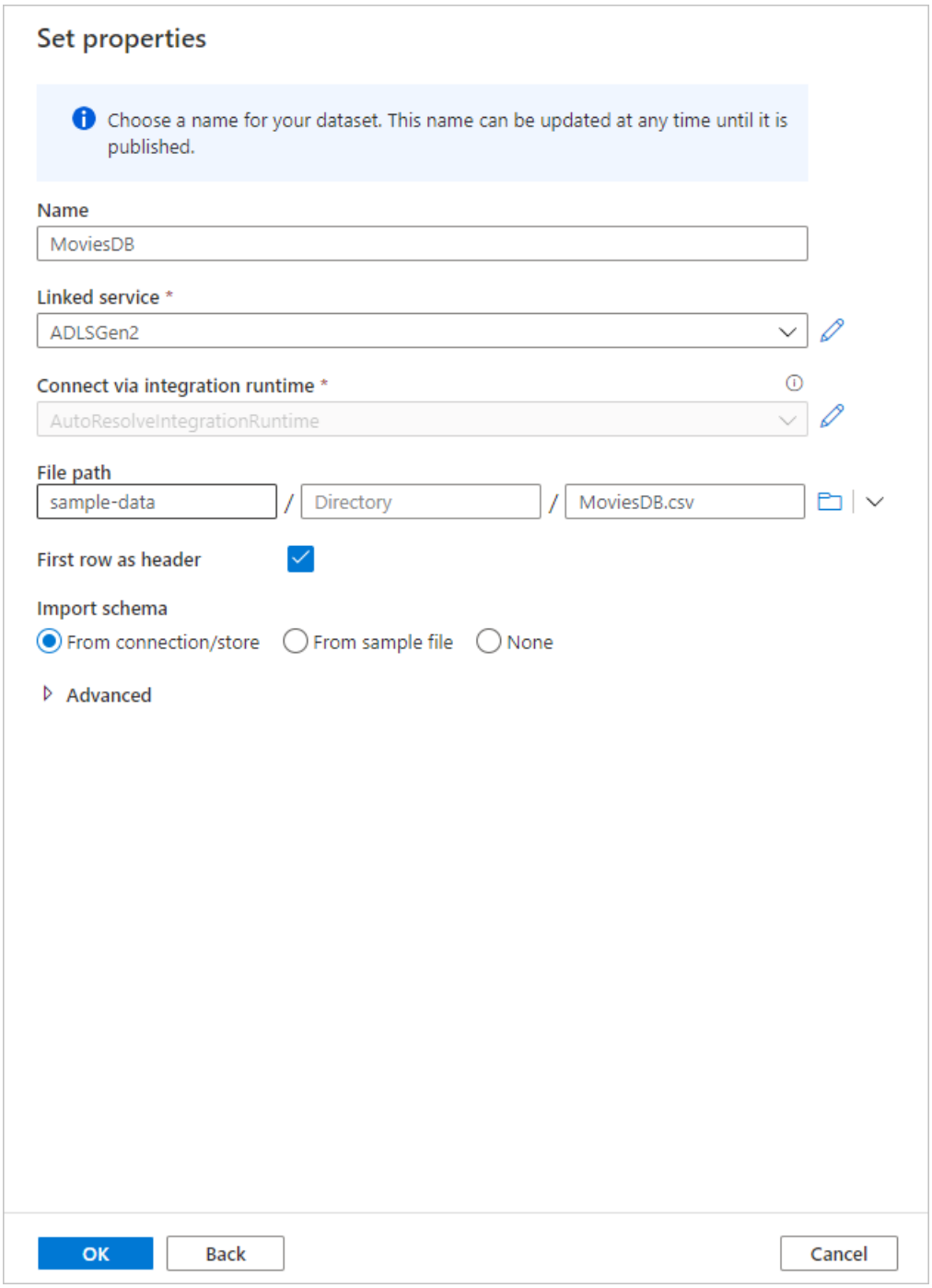
5. Choose **DelimitedText**. Click Continue.

6. Name your dataset **MoviesDB**. In the linked service dropdown, choose **New**.

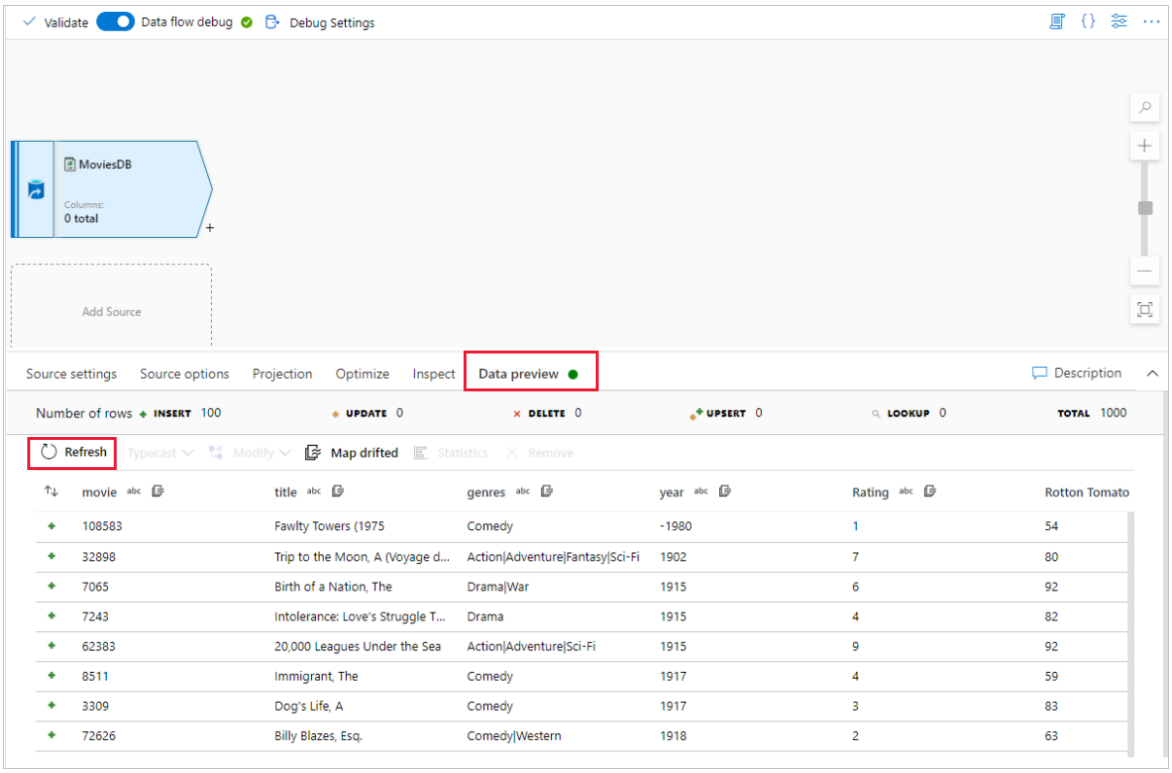
7. In the linked service creation screen, name your ADLS Gen2 linked service **ADLSGen2** and specify your authentication method. Then enter your connection credentials. In this quickstart, we're using Account key to connect to our storage account. You can click **Test connection** to verify your credentials were entered correctly. Click **Create** when finished.



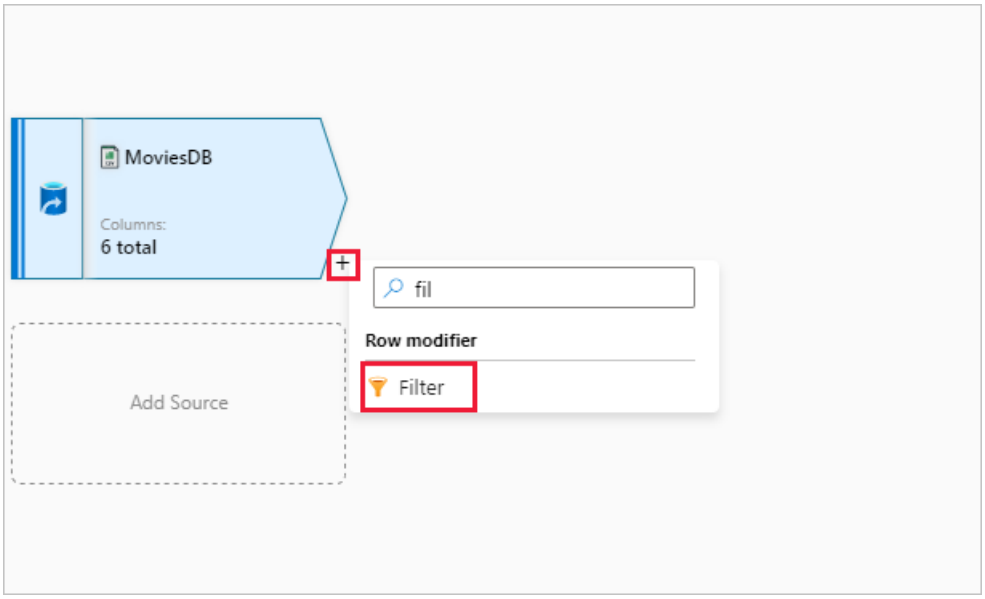
8. Once you're back at the dataset creation screen, under the **File path** field, enter where your file is located. In this quickstart, the file "MoviesDB.csv" is located in container "sample-data". As the file has headers, check **First row as header**. Select **From connection/store** to import the header schema directly from the file in storage. Click **OK** when done.



9. If your debug cluster has started, go to the **Data Preview** tab of the source transformation and click **Refresh** to get a snapshot of the data. You can use data preview to verify your transformation is configured correctly.



10. Next to your source node on the data flow canvas, click on the plus icon to add a new transformation. The first transformation you're adding is a **Filter**.



11. Name your filter transformation **FilterYears**. Click on the expression box next to **Filter on** to open the expression builder. Here you'll specify your filtering condition.

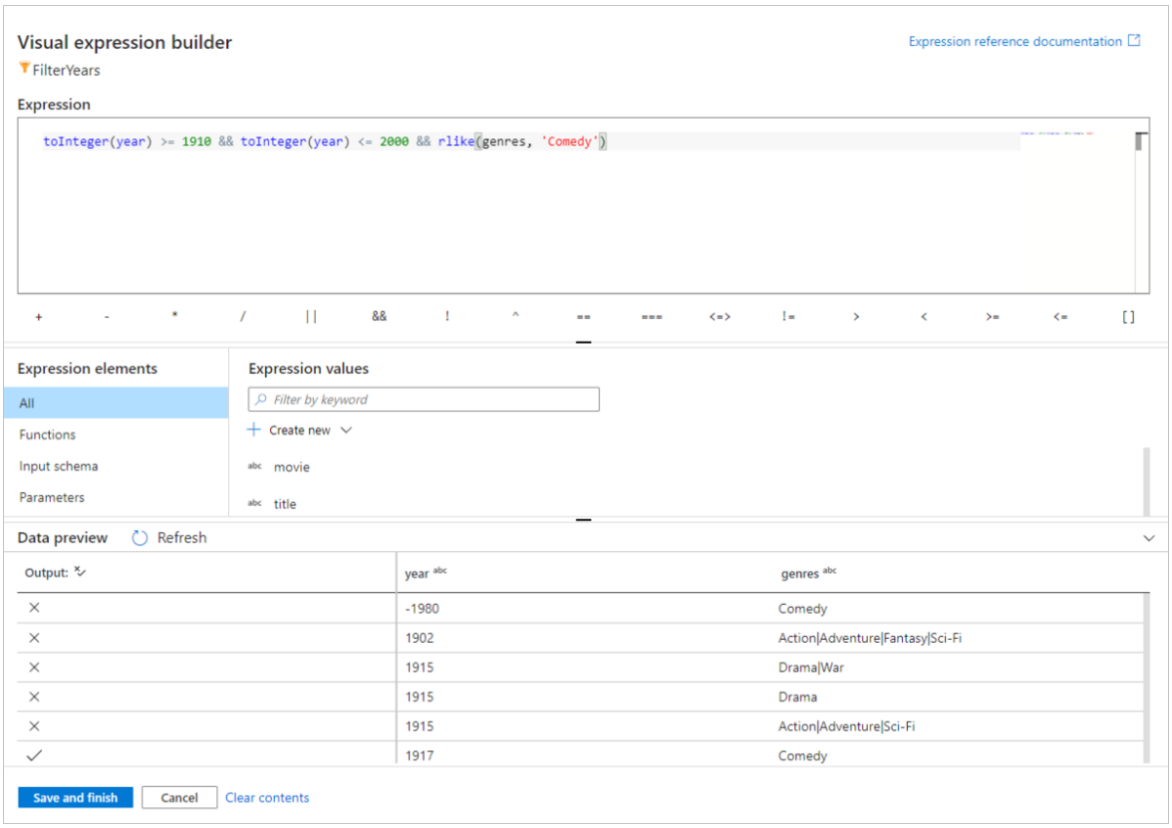
12. The data flow expression builder lets you interactively build expressions to use in various transformations. Expressions can include built-in functions, columns from the input schema, and user-defined parameters. For more information on how to build expressions, see Data Flow expression builder.

    In this quickstart, you want to filter movies of genre comedy that came out between the years 1910 and 2000. As year is currently a string, you need to convert it to an integer using the `toInteger()` function. Use the greater than or equals to (>=) and less than or equals to (<=) operators to compare against literal year values 1910 and 200-. Union these expressions together with the `&&` (and) operator. The expression comes out as:

    ```
    toInteger(year) >= 1910 && toInteger(year) <= 2000
    ```

To find which movies are comedies, you can use the `rlike()` function to find pattern 'Comedy' in the column genres. Union the `rlike` expression with the year comparison to get:

```
toInteger(year) >= 1910 && toInteger(year) <= 2000 && rlike(genres, 'Comedy')
```

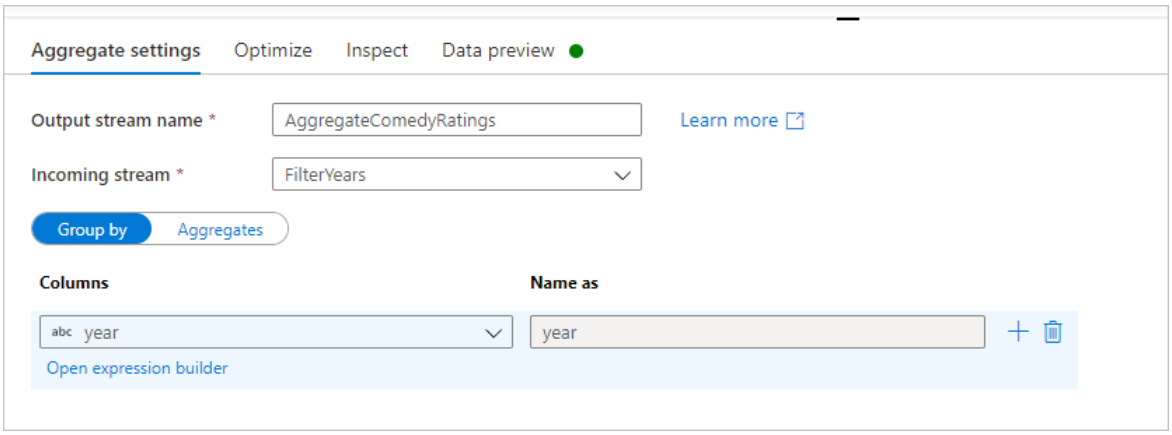


If you've a debug cluster active, you can verify your logic by clicking **Refresh** to see expression output compared to the inputs used. There's more than one right answer on how you can accomplish this logic using the data flow expression language.

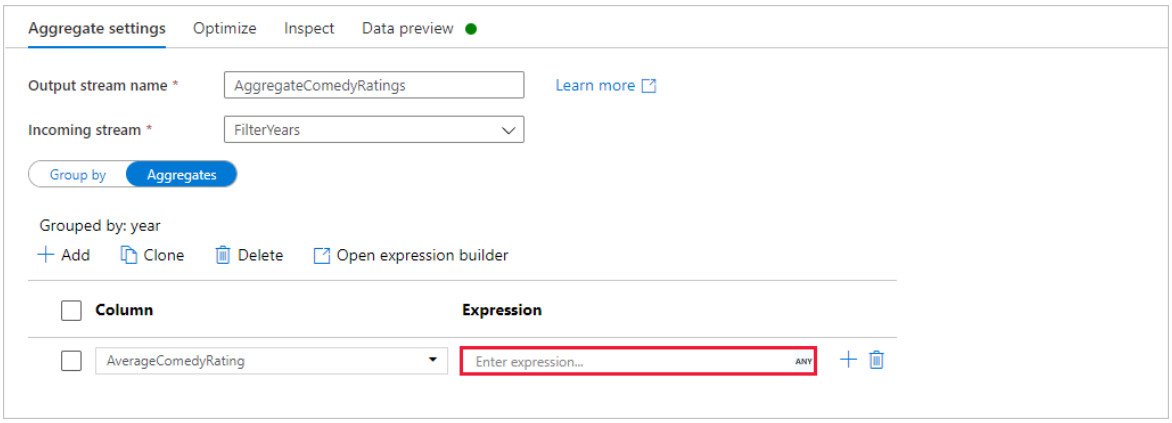
Click **Save and Finish** once you're done with your expression.

13. Fetch a **Data Preview** to verify the filter is working correctly.
14. The next transformation you'll add is an **Aggregate** transformation under **Schema modifier**.

15. Name your aggregate transformation **AggregateComedyRatings**. In the **Group by** tab, select **year** from the dropdown to group the aggregations by the year the movie came out.



16. Go to the **Aggregates** tab. In the left text box, name the aggregate column **AverageComedyRating**. Click on the right expression box to enter the aggregate expression via the expression builder.

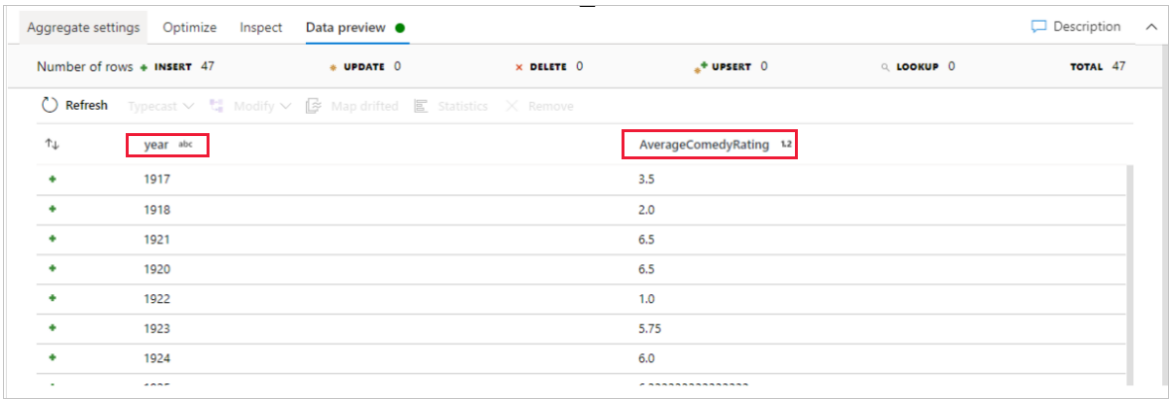


17. To get the average of column **Rating**, use the `avg()` aggregate function. As **Rating** is a string and `avg()` takes in a numerical input, we must convert the value to a number via the `toInteger()` function. This expression looks like:

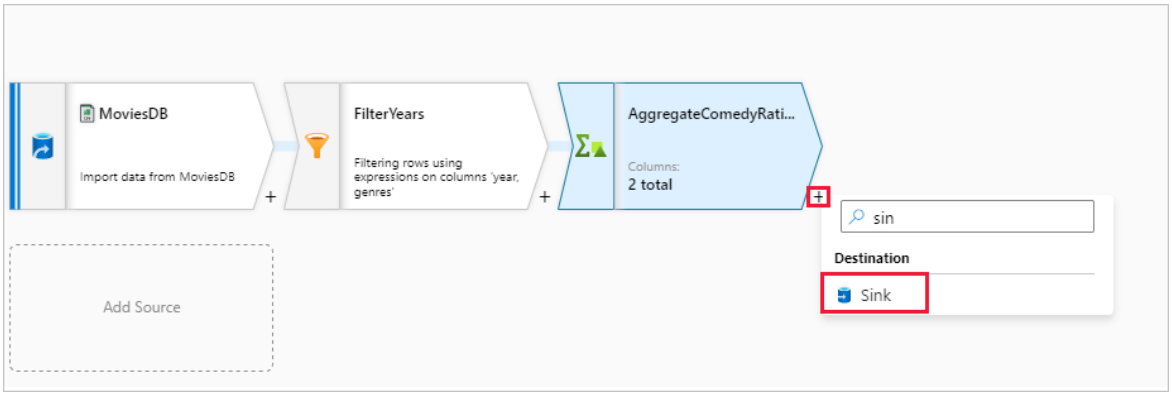
    ```
    avg(toInteger(Rating))
    ```

    Click **Save and Finish** when done.

18. Go to the **Data Preview** tab to view the transformation output. Notice only two columns are there, **year** and **AverageComedyRating**.



19. Next, you want to add a **Sink** transformation under **Destination**.

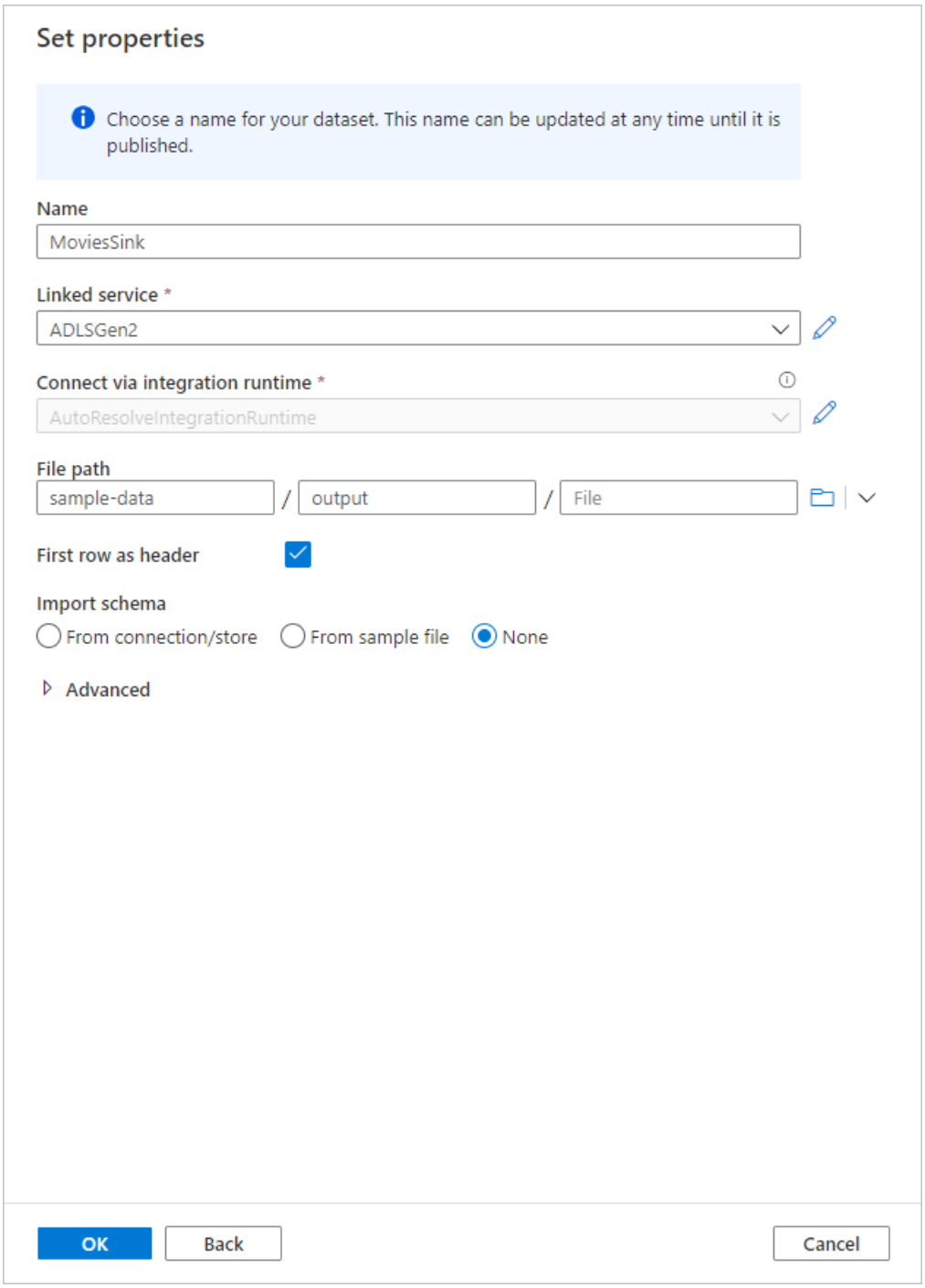


20. Name your sink **Sink**. Click **New** to create your sink dataset.

21. Choose **Azure Data Lake Storage Gen2**. Click Continue.

22. Choose **DelimitedText**. Click Continue.

23. Name your sink dataset **MoviesSink**. For linked service, choose the ADLS Gen2 linked service you created in step 7. Enter an output folder to write your data to. In this quickstart, we're writing to folder 'output' in container 'sample-data'. The folder doesn't need to exist beforehand and can be dynamically created. Set **First row as header** as true and select **None** for **Import schema**. Click **OK** when done.

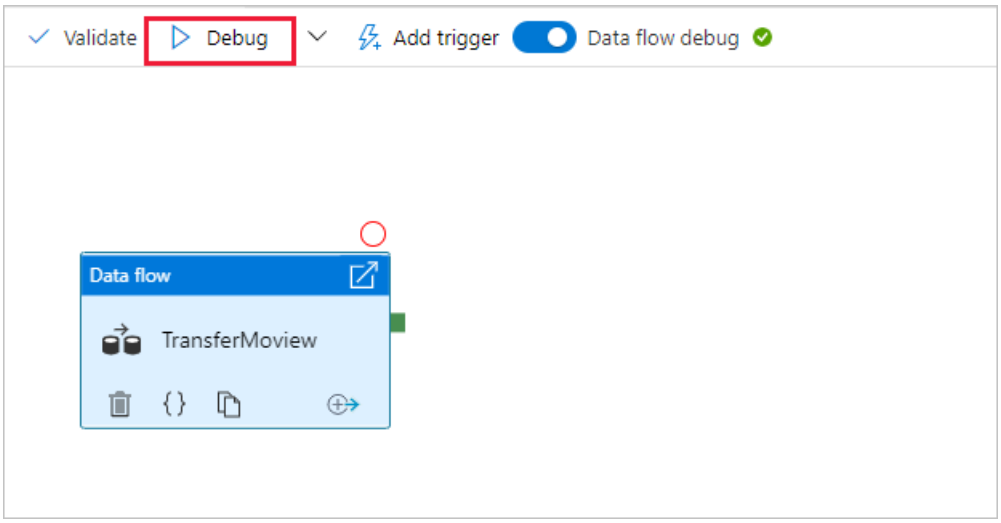


Now you've finished building your data flow. You're ready to run it in your pipeline.

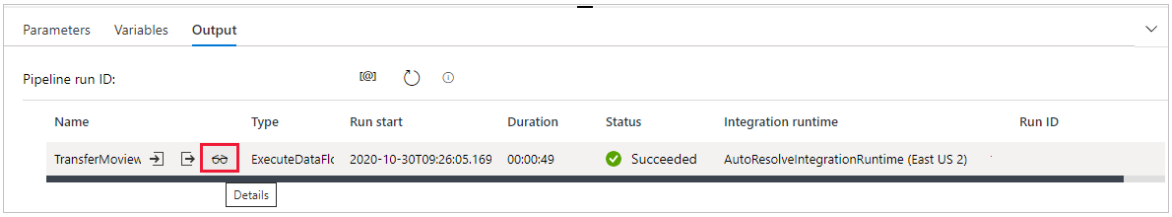
Running and monitoring the Data Flow

You can debug a pipeline before you publish it. In this step, you're going to trigger a debug run of the data flow pipeline. While data preview doesn't write data, a debug run will write data to your sink destination.

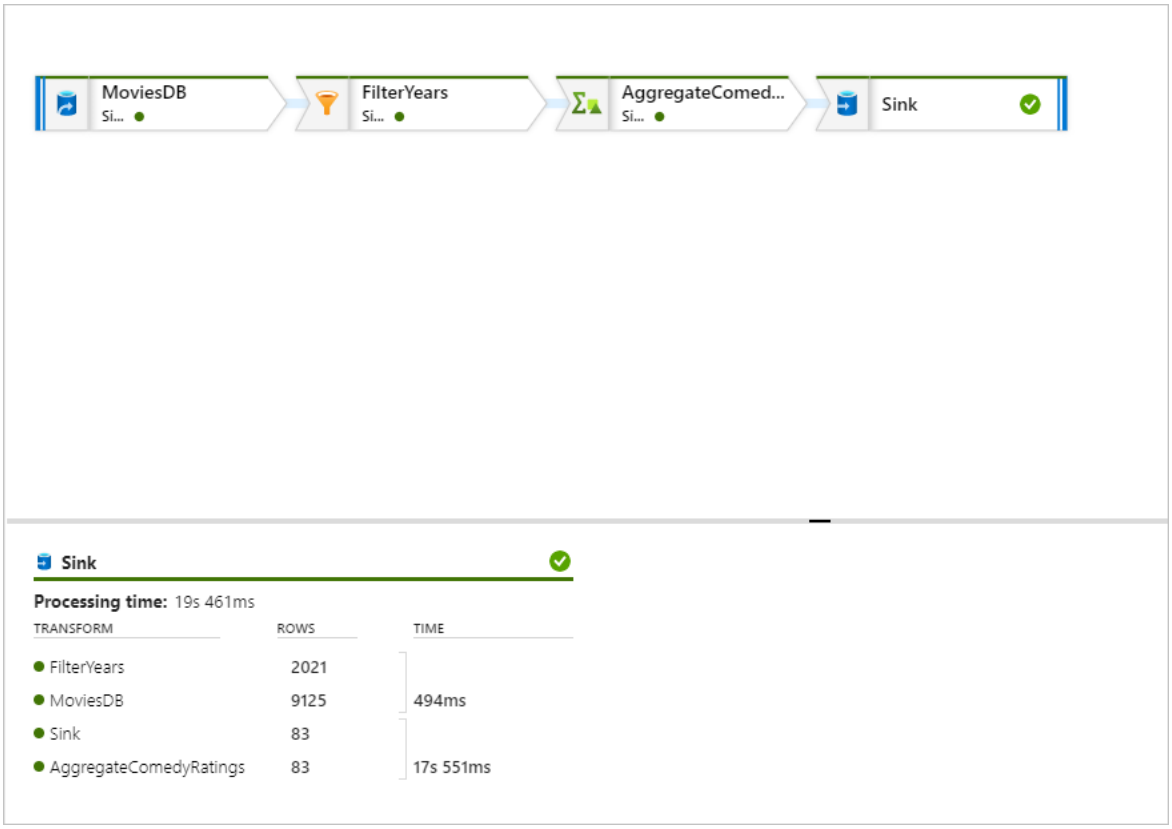
1. Go to the pipeline canvas. Click **Debug** to trigger a debug run.



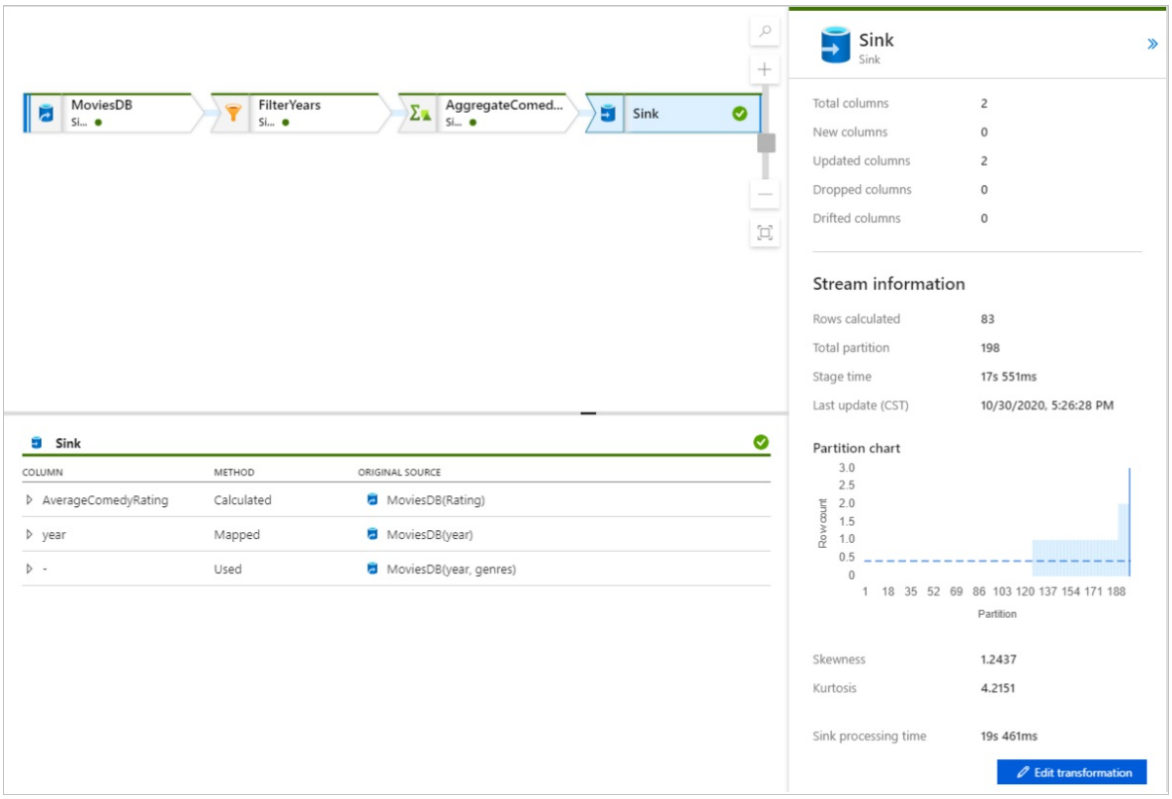
2. Pipeline debug of Data Flow activities uses the active debug cluster but still take at least a minute to initialize. You can track the progress via the **Output** tab. Once the run is successful, click on the eyeglasses icon to open the monitoring pane.



3. In the monitoring pane, you can see the number of rows and time spent in each transformation step.



4. Click on a transformation to get detailed information about the columns and partitioning of the data.



If you followed this quickstart correctly, you should have written 83 rows and 2 columns into your sink folder. You can verify the data by checking your blob storage.

Next steps

Advance to the following articles to learn about Azure Synapse Analytics support:

Pipeline and activities Mapping data flow overview Data flow expression language

Quickstart: Transform data using Apache Spark job definition.

8/19/2022 • 3 minutes to read • Edit Online

In this quickstart, you'll use Azure Synapse Analytics to create a pipeline using Apache Spark job definition.

Prerequisites

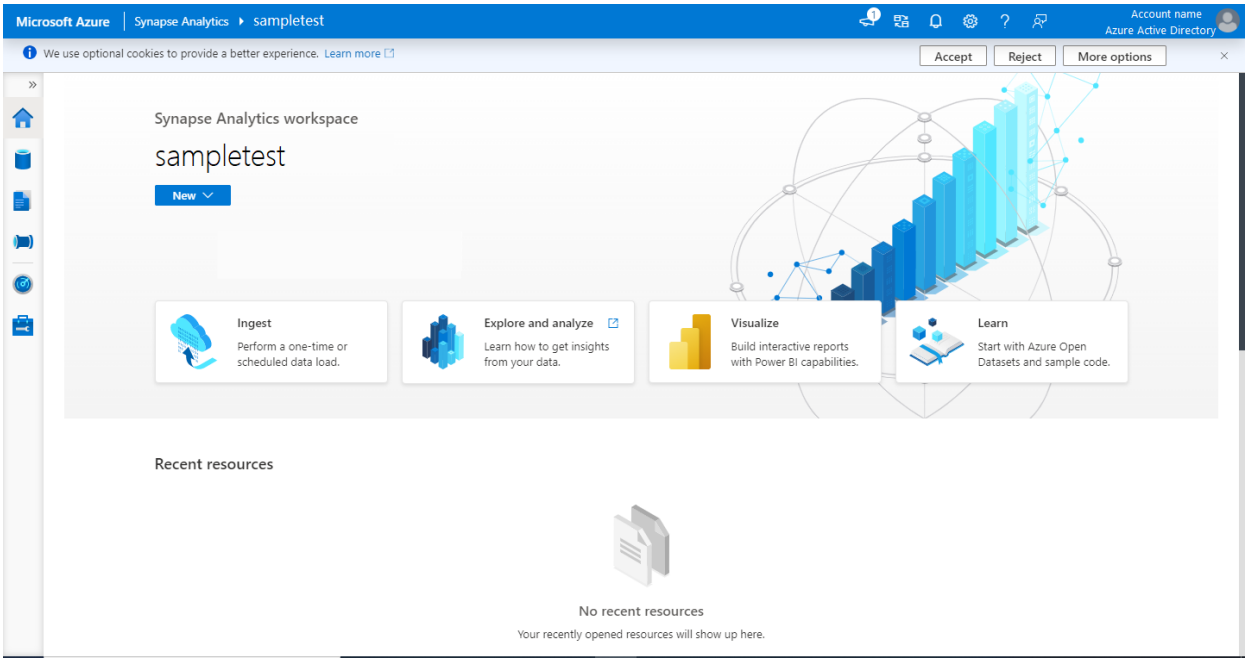
- **Azure subscription**: If you don't have an Azure subscription, create a free account before you begin.
- **Azure Synapse workspace**: Create a Synapse workspace using the Azure portal following the instructions in Quickstart: Create a Synapse workspace.
- **Apache Spark job definition**: Create an Apache Spark job definition in the Synapse workspace following the instructions in Tutorial: Create Apache Spark job definition in Synapse Studio.

Navigate to the Synapse Studio

After your Azure Synapse workspace is created, you have two ways to open Synapse Studio:

- Open your Synapse workspace in the Azure portal. Select **Open** on the Open Synapse Studio card under Getting started.
- Open Azure Synapse Analytics and sign in to your workspace.

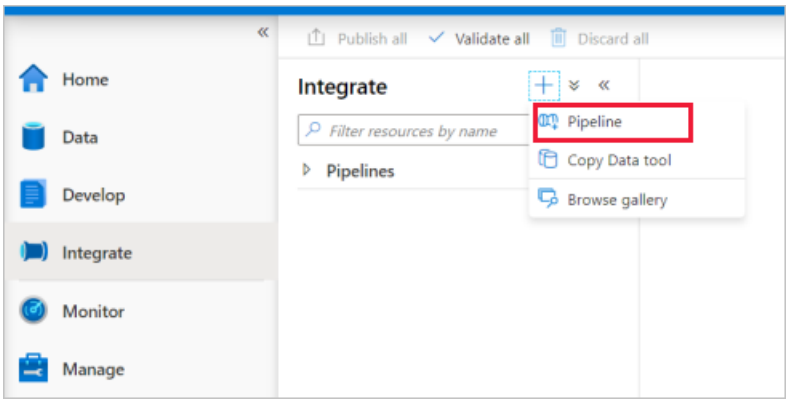
In this quickstart, we use the workspace named "sampletest" as an example. It will automatically navigate you to the Synapse Studio home page.



Create a pipeline with an Apache Spark job definition

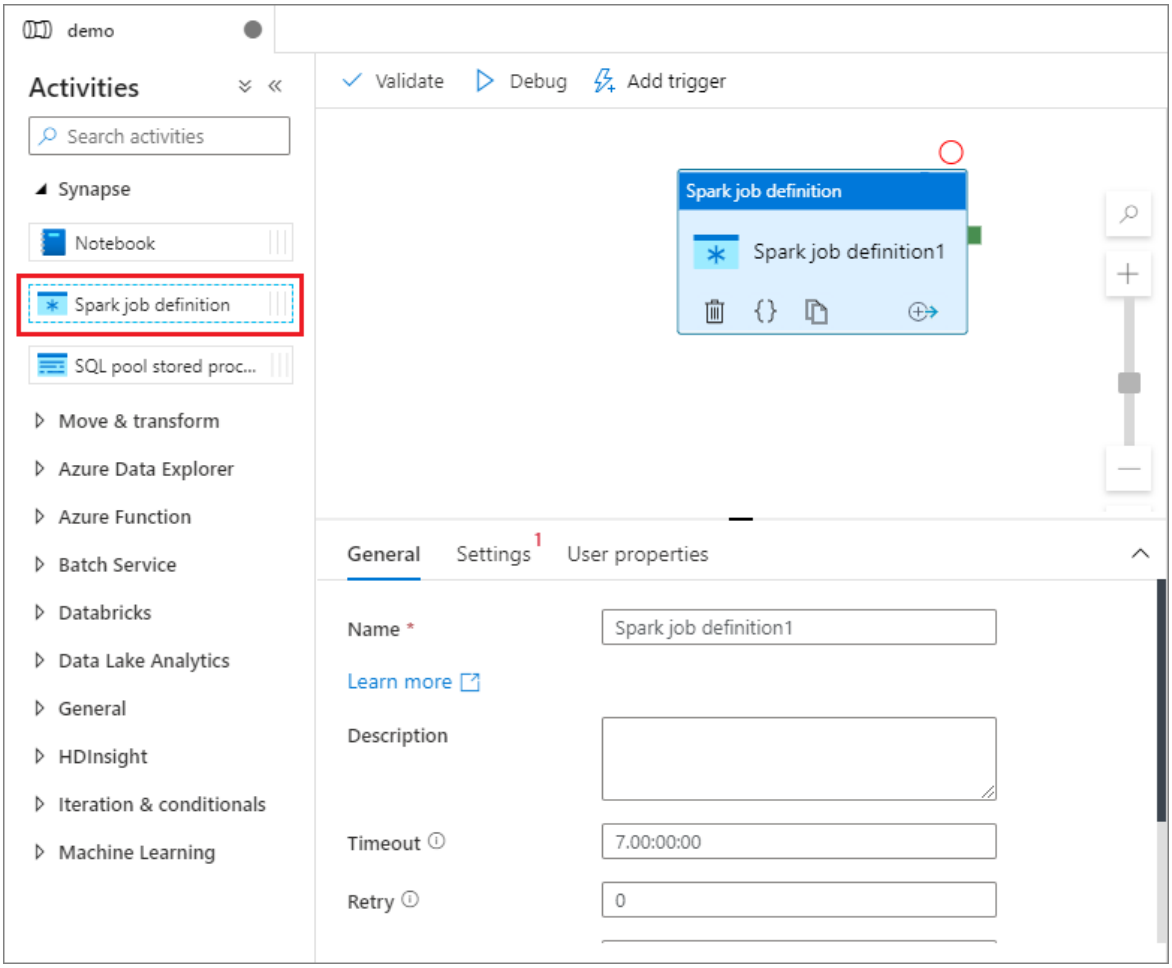
A pipeline contains the logical flow for an execution of a set of activities. In this section, you'll create a pipeline that contains an Apache Spark job definition activity.

1. Go to the **Integrate** tab. Select the plus icon next to the pipelines header and select **Pipeline**.



2. In the **Properties** settings page of the pipeline, enter **demo** for **Name**.

3. Under **Synapse** in the **Activities** pane, drag **Spark job definition** onto the pipeline canvas.



Set Apache Spark job definition canvas

Once you create your Apache Spark job definition, you'll be automatically sent to the Spark job definition canvas.

General settings

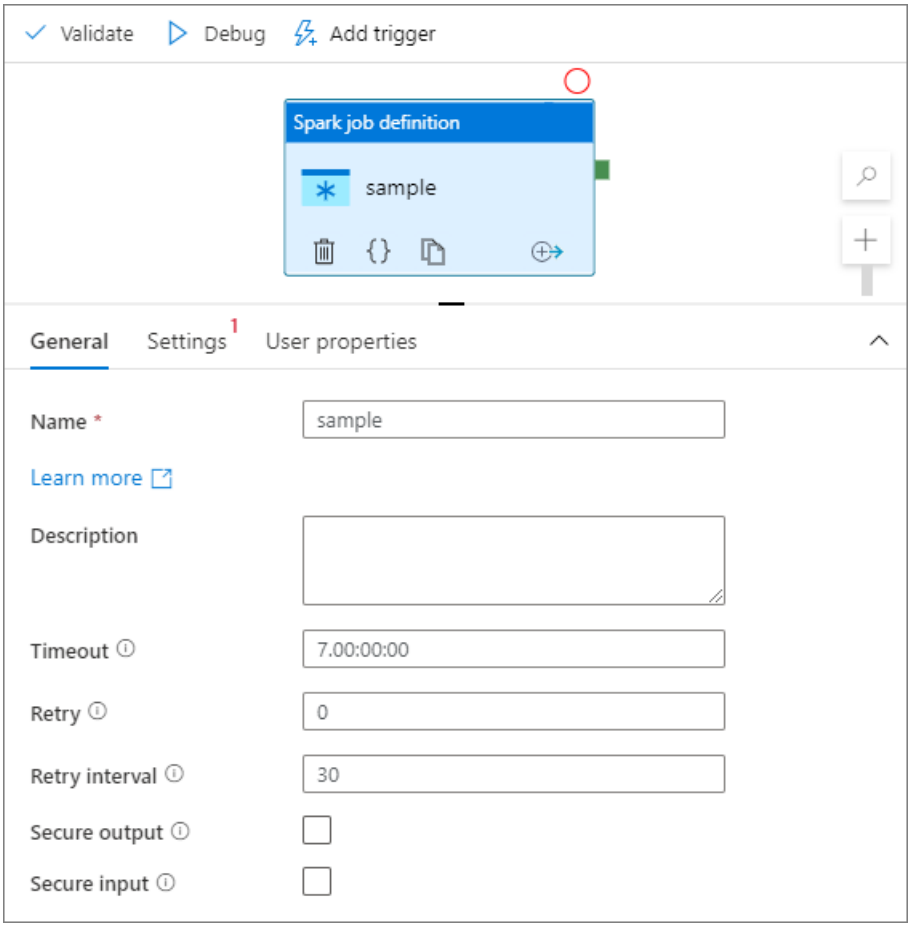
1. Select the spark job definition module on the canvas.

2. In the **General** tab, enter **sample** for **Name**.

3. (Option) You can also enter a description.

4. Timeout: Maximum amount of time an activity can run. Default is seven days, which is also the maximum amount of time allowed. Format is in D.HH:MM:SS.

5. Retry: Maximum number of retry attempts.
6. Retry interval: The number of seconds between each retry attempt.
7. Secure output: When checked, output from the activity won't be captured in logging.
8. Secure input: When checked, input from the activity won't be captured in logging.



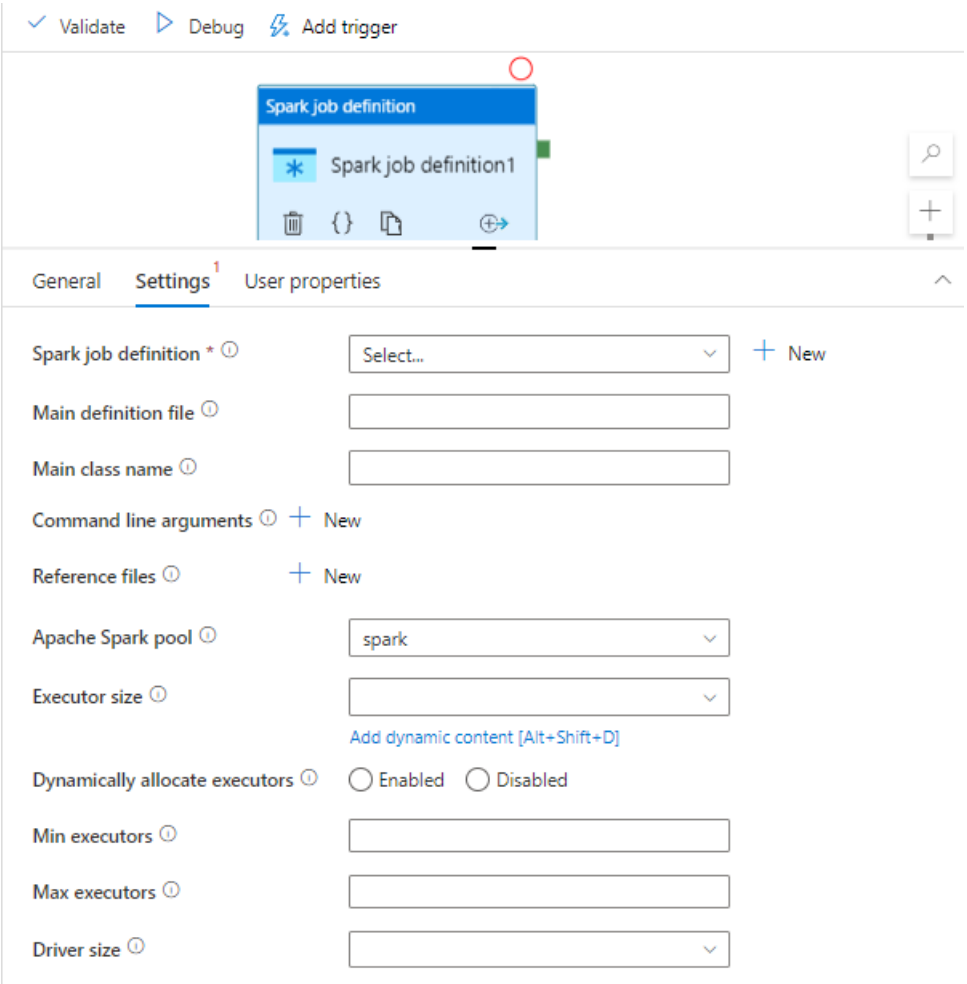
Settings tab

On this panel, you can reference to the Spark job definition to run.

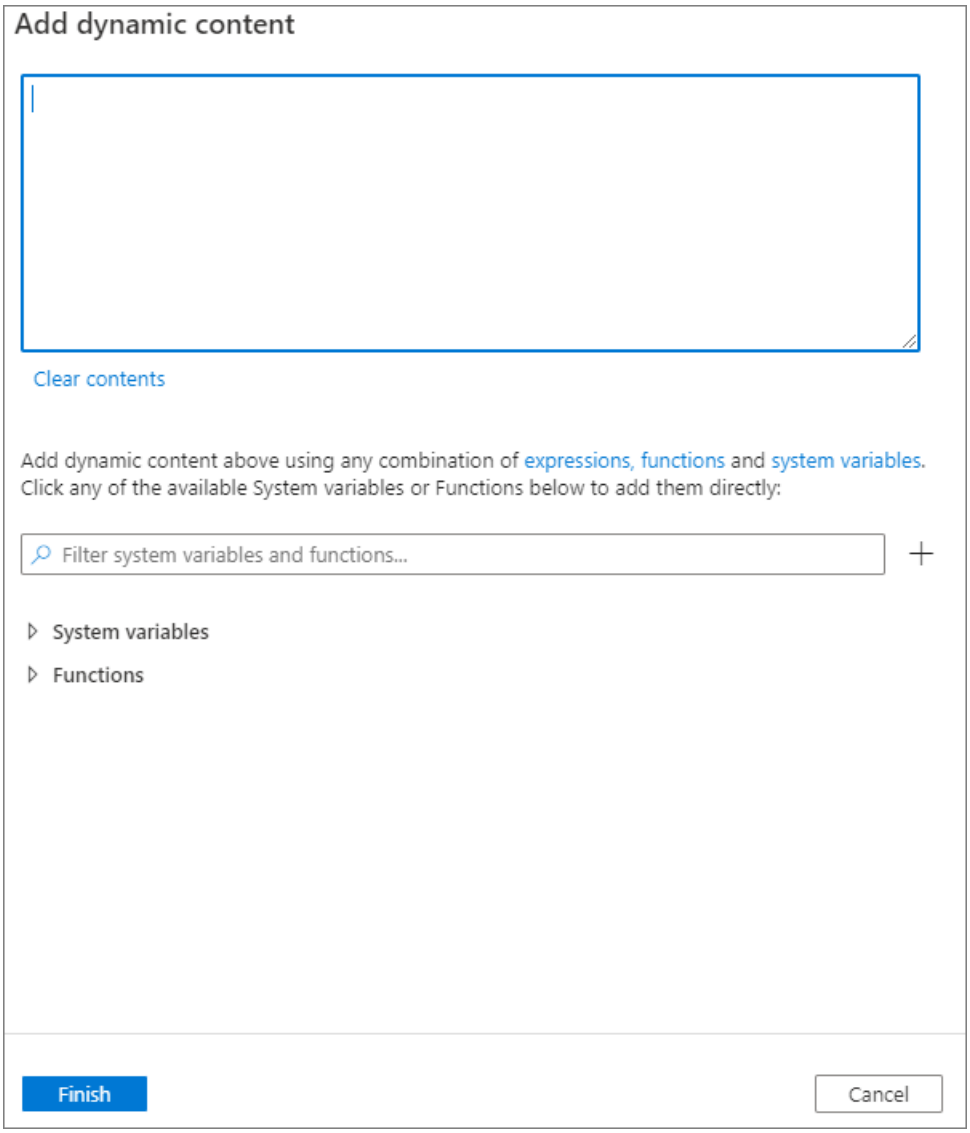
- Expand the Spark job definition list, you can choose an existing Apache Spark job definition. You can also create a new Apache Spark job definition by selecting the **New** button to reference the Spark job definition to be run.
- (Optional) You can fill in information for Apache Spark job definition. If the following settings are empty, the settings of the spark job definition itself will be used to run; if the following settings are not empty, these settings will replace the settings of the spark job definition itself.

PROPERTY	DESCRIPTION
Main definition file	The main file used for the job. Select a PY/JAR/ZIP file from your storage. You can select **Upload file** to upload the file to a storage account. Sample: `abfss://…/path/to/wordcount.jar`
Main class name	The fully qualified identifier or the main class that is in the main definition file. Sample: `WordCount`

PROPERTY	DESCRIPTION
Command-line arguments	You can add command-line arguments by clicking the **New** button. It should be noted that adding command-line arguments will override the command-line arguments defined by the Spark job definition. *Sample:* `abfss://…/path/to/shakespeare.txt` `abfss://…/path/to/result`
Apache Spark pool	You can select Apache Spark pool from the list.
Dynamically allocate executors	This setting maps to the dynamic allocation property in Spark configuration for Spark Application executors allocation.
Min executors	Min number of executors to be allocated in the specified Spark pool for the job.
Max executors	Max number of executors to be allocated in the specified Spark pool for the job.
Driver size	Number of cores and memory to be used for driver given in the specified Apache Spark pool for the job.



- You can add dynamic content by clicking the **Add Dynamic Content** button or by pressing the shortcut key `Alt+Shift+D`. In the **Add Dynamic Content** page, you can use any combination of expressions, functions, and system variables to add to dynamic content.



User properties tab

You can add properties for Apache Spark job definition activity in this panel.

Next steps

Advance to the following articles to learn about Azure Synapse Analytics support:

Pipeline and activities Mapping data flow overview Data flow expression language

Pipelines and activities in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 18 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

IMPORTANT

Support for Machine Learning Studio (classic) will end on 31 August 2024. We recommend you transition to Azure Machine Learning by that date.

Beginning 1 December 2021, you will not be able to create new Machine Learning Studio (classic) resources (workspace and web service plan). Through 31 August 2024, you can continue to use the existing Machine Learning Studio (classic) experiments and web services.

- See information on moving machine learning projects from ML Studio (classic) to Azure Machine Learning.
- Learn more about Azure Machine Learning

ML Studio (classic) documentation is being retired and may not be updated in the future.

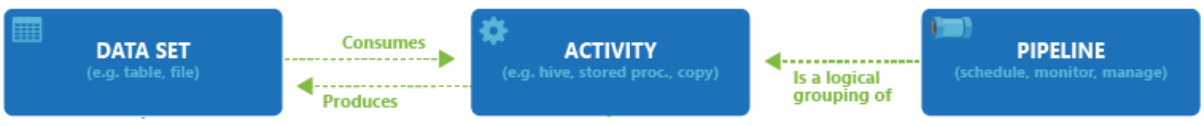
This article helps you understand pipelines and activities in Azure Data Factory and Azure Synapse Analytics and use them to construct end-to-end data-driven workflows for your data movement and data processing scenarios.

Overview

A Data Factory or Synapse Workspace can have one or more pipelines. A pipeline is a logical grouping of activities that together perform a task. For example, a pipeline could contain a set of activities that ingest and clean log data, and then kick off a mapping data flow to analyze the log data. The pipeline allows you to manage the activities as a set instead of each one individually. You deploy and schedule the pipeline instead of the activities independently.

The activities in a pipeline define actions to perform on your data. For example, you may use a copy activity to copy data from SQL Server to an Azure Blob Storage. Then, use a data flow activity or a Databricks Notebook activity to process and transform data from the blob storage to an Azure Synapse Analytics pool on top of which business intelligence reporting solutions are built.

Azure Data Factory and Azure Synapse Analytics have three groupings of activities: data movement activities, data transformation activities, and control activities. An activity can take zero or more input datasets and produce one or more output datasets. The following diagram shows the relationship between pipeline, activity, and dataset:



An input dataset represents the input for an activity in the pipeline, and an output dataset represents the output for the activity. Datasets identify data within different data stores, such as tables, files, folders, and documents. After you create a dataset, you can use it with activities in a pipeline. For example, a dataset can be an input/output dataset of a Copy Activity or an HDInsightHive Activity. For more information about datasets, see

Datasets in Azure Data Factory article.

Data movement activities

Copy Activity in Data Factory copies data from a source data store to a sink data store. Data Factory supports the data stores listed in the table in this section. Data from any source can be written to any sink. Click a data store to learn how to copy data to and from that store.

CATEGORY	DATA STORE	SUPPORTED AS A SOURCE	SUPPORTED AS A SINK	SUPPORTED BY AZURE IR	SUPPORTED BY SELF-HOSTED IR
Azure	Azure Blob storage	✓	✓	✓	✓
	Azure Cognitive Search index		✓	✓	✓
	Azure Cosmos DB (SQL API)	✓	✓	✓	✓
	Azure Cosmos DB's API for MongoDB	✓	✓	✓	✓
	Azure Data Explorer	✓	✓	✓	✓
	Azure Data Lake Storage Gen1	✓	✓	✓	✓
	Azure Data Lake Storage Gen2	✓	✓	✓	✓
	Azure Database for MariaDB	✓		✓	✓
	Azure Database for MySQL	✓	✓	✓	✓
	Azure Database for PostgreSQL	✓	✓	✓	✓
	Azure Databricks Delta Lake	✓	✓	✓	✓
	Azure Files	✓	✓	✓	✓
	Azure SQL Database	✓	✓	✓	✓
	Azure SQL Managed Instance	✓	✓	✓	✓
	Azure Synapse Analytics	✓	✓	✓	✓

CATEGORY	DATA STORE	SUPPORTED AS A SOURCE	SUPPORTED AS A SINK	SUPPORTED BY AZURE IR	SUPPORTED BY SELF-HOSTED IR
	Azure Table storage	✓	✓	✓	✓
Database	Amazon RDS for Oracle	✓		✓	✓
	Amazon RDS for SQL Server	✓		✓	✓
	Amazon Redshift	✓		✓	✓
	DB2	✓		✓	✓
	Drill	✓		✓	✓
	Google BigQuery	✓		✓	✓
	Greenplum	✓		✓	✓
	HBase	✓		✓	✓
	Hive	✓		✓	✓
	Apache Impala	✓		✓	✓
	Informix	✓	✓		✓
	MariaDB	✓		✓	✓
	Microsoft Access	✓	✓		✓
	MySQL	✓		✓	✓
	Netezza	✓		✓	✓
	Oracle	✓	✓	✓	✓
	Phoenix	✓		✓	✓
	PostgreSQL	✓		✓	✓
	Presto	✓		✓	✓
	SAP Business Warehouse via Open Hub	✓			✓
	SAP Business Warehouse via MDX	✓			✓

CATEGORY	DATA STORE	SUPPORTED AS A SOURCE	SUPPORTED AS A SINK	SUPPORTED BY AZURE IR	SUPPORTED BY SELF-HOSTED IR
	SAP HANA	✓	Sink supported only with the ODBC Connector and the SAP HANA ODBC driver		✓
	SAP table	✓			✓
	Snowflake	✓	✓	✓	✓
	Spark	✓		✓	✓
	SQL Server	✓	✓	✓	✓
	Sybase	✓			✓
	Teradata	✓		✓	✓
	Vertica	✓		✓	✓
NoSQL	Cassandra	✓		✓	✓
	Couchbase (Preview)	✓		✓	✓
	MongoDB	✓	✓	✓	✓
	MongoDB Atlas	✓	✓	✓	✓
File	Amazon S3	✓		✓	✓
	Amazon S3 Compatible Storage	✓		✓	✓
	File system	✓	✓	✓	✓
	FTP	✓		✓	✓
	Google Cloud Storage	✓		✓	✓
	HDFS	✓		✓	✓
	Oracle Cloud Storage	✓		✓	✓
	SFTP	✓	✓	✓	✓

CATEGORY	DATA STORE	SUPPORTED AS A SOURCE	SUPPORTED AS A SINK	SUPPORTED BY AZURE IR	SUPPORTED BY SELF-HOSTED IR
Generic protocol	Generic HTTP	✓		✓	✓
	Generic OData	✓		✓	✓
	Generic ODBC	✓	✓		✓
	Generic REST	✓	✓	✓	✓
Services and apps	Amazon Marketplace Web Service	✓		✓	✓
	Concur (Preview)	✓		✓	✓
	Dataverse	✓	✓	✓	✓
	Dynamics 365	✓	✓	✓	✓
	Dynamics AX	✓		✓	✓
	Dynamics CRM	✓	✓	✓	✓
	Google AdWords	✓		✓	✓
	HubSpot	✓		✓	✓
	Jira	✓		✓	✓
	Magento (Preview)	✓		✓	✓
	Marketo (Preview)	✓		✓	✓
	Microsoft 365	✓		✓	✓
	Oracle Eloqua (Preview)	✓		✓	✓
	Oracle Responsys (Preview)	✓		✓	✓
	Oracle Service Cloud (Preview)	✓		✓	✓
	PayPal (Preview)	✓		✓	✓
	QuickBooks (Preview)	✓		✓	✓

CATEGORY	DATA STORE	SUPPORTED AS A SOURCE	SUPPORTED AS A SINK	SUPPORTED BY AZURE IR	SUPPORTED BY SELF-HOSTED IR
	Salesforce	✓	✓	✓	✓
	Salesforce Service Cloud	✓	✓	✓	✓
	Salesforce Marketing Cloud	✓		✓	✓
	SAP Cloud for Customer (C4C)	✓	✓	✓	✓
	SAP ECC	✓		✓	✓
	ServiceNow	✓		✓	✓
	SharePoint Online List	✓		✓	✓
	Shopify (Preview)	✓		✓	✓
	Square (Preview)	✓		✓	✓
	Web table (HTML table)	✓			✓
	Xero	✓		✓	✓
	Zoho (Preview)	✓		✓	✓

NOTE

If a connector is marked *Preview*, you can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, contact Azure support.

For more information, see Copy Activity - Overview article.

Data transformation activities

Azure Data Factory and Azure Synapse Analytics support the following transformation activities that can be added either individually or chained with another activity.

DATA TRANSFORMATION ACTIVITY	COMPUTE ENVIRONMENT
Data Flow	Apache Spark clusters managed by Azure Data Factory
Azure Function	Azure Functions
Hive	HDInsight [Hadoop]
Pig	HDInsight [Hadoop]

DATA TRANSFORMATION ACTIVITY	COMPUTE ENVIRONMENT
MapReduce	HDInsight [Hadoop]
Hadoop Streaming	HDInsight [Hadoop]
Spark	HDInsight [Hadoop]
ML Studio (classic) activities: Batch Execution and Update Resource	Azure VM
Stored Procedure	Azure SQL, Azure Synapse Analytics, or SQL Server
U-SQL	Azure Data Lake Analytics
Custom Activity	Azure Batch
Databricks Notebook	Azure Databricks
Databricks Jar Activity	Azure Databricks
Databricks Python Activity	Azure Databricks

For more information, see the data transformation activities article.

Control flow activities

The following control flow activities are supported:

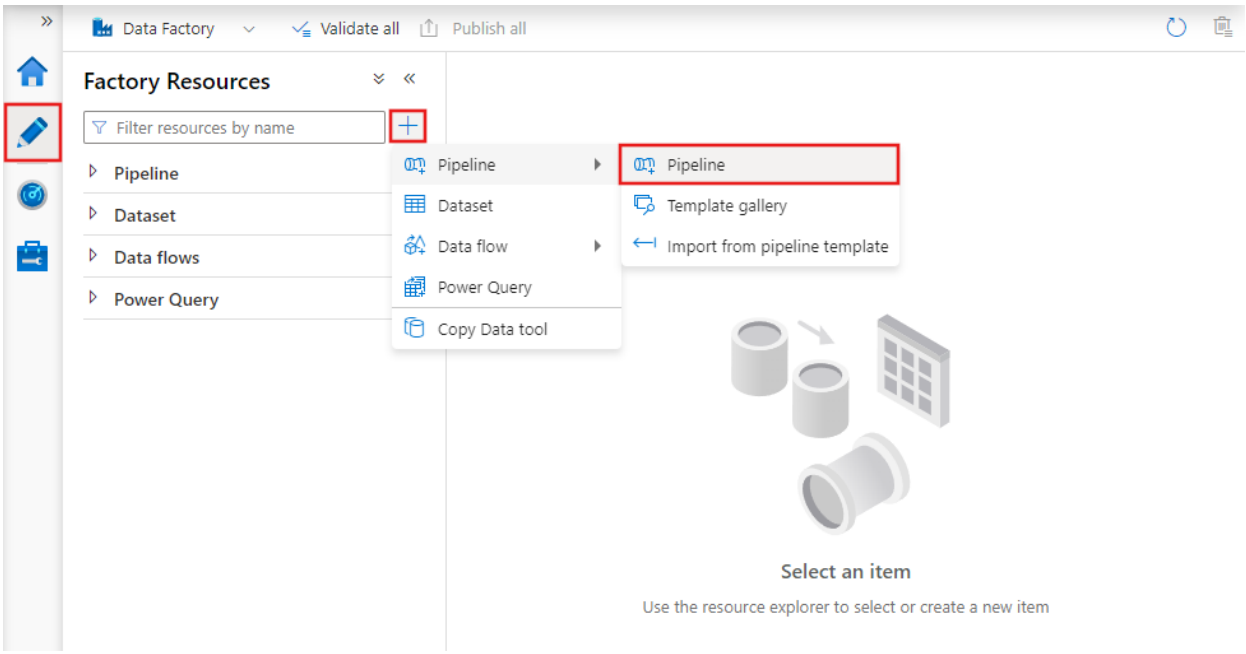
CONTROL ACTIVITY	DESCRIPTION
Append Variable	Add a value to an existing array variable.
Execute Pipeline	Execute Pipeline activity allows a Data Factory or Synapse pipeline to invoke another pipeline.
Filter	Apply a filter expression to an input array
For Each	ForEach Activity defines a repeating control flow in your pipeline. This activity is used to iterate over a collection and executes specified activities in a loop. The loop implementation of this activity is similar to the Foreach looping structure in programming languages.
Get Metadata	GetMetadata activity can be used to retrieve metadata of any data in a Data Factory or Synapse pipeline.
If Condition Activity	The If Condition can be used to branch based on condition that evaluates to true or false. The If Condition activity provides the same functionality that an if statement provides in programming languages. It evaluates a set of activities when the condition evaluates to `true` and another set of activities when the condition evaluates to `false.`

CONTROL ACTIVITY	DESCRIPTION
Lookup Activity	Lookup Activity can be used to read or look up a record/ table name/ value from any external source. This output can further be referenced by succeeding activities.
Set Variable	Set the value of an existing variable.
Until Activity	Implements Do-Until loop that is similar to Do-Until looping structure in programming languages. It executes a set of activities in a loop until the condition associated with the activity evaluates to true. You can specify a timeout value for the until activity.
Validation Activity	Ensure a pipeline only continues execution if a reference dataset exists, meets a specified criteria, or a timeout has been reached.
Wait Activity	When you use a Wait activity in a pipeline, the pipeline waits for the specified time before continuing with execution of subsequent activities.
Web Activity	Web Activity can be used to call a custom REST endpoint from a pipeline. You can pass datasets and linked services to be consumed and accessed by the activity.
Webhook Activity	Using the webhook activity, call an endpoint, and pass a callback URL. The pipeline run waits for the callback to be invoked before proceeding to the next activity.

Creating a pipeline with UI

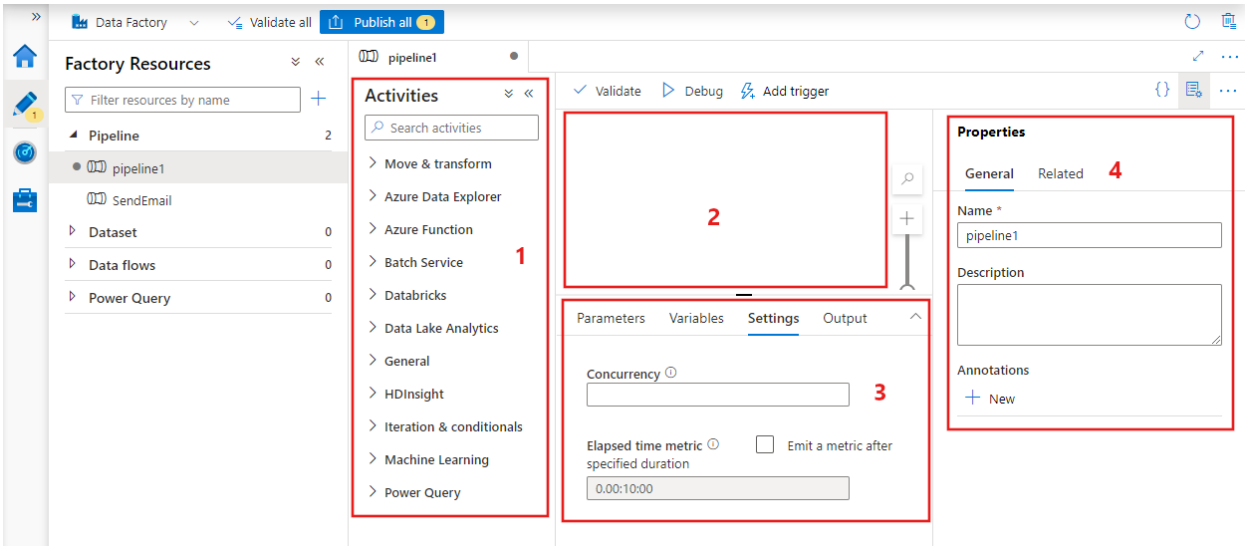
- Azure Data Factory
- Synapse Analytics

To create a new pipeline, navigate to the Author tab in Data Factory Studio (represented by the pencil icon), then click the plus sign and choose Pipeline from the menu, and Pipeline again from the submenu.



Data factory will display the pipeline editor where you can find:

1. All activities that can be used within the pipeline.
2. The pipeline editor canvas, where activities will appear when added to the pipeline.
3. The pipeline configurations pane, including parameters, variables, general settings, and output.
4. The pipeline properties pane, where the pipeline name, optional description, and annotations can be configured. This pane will also show any related items to the pipeline within the data factory.



Pipeline JSON

Here is how a pipeline is defined in JSON format:

```
{
    "name": "PipelineName",
    "properties":
    {
        "description": "pipeline description",
        "activities":
        [
        ],
        "parameters": {
        },
        "concurrency": <your max pipeline concurrency>,
        "annotations": [
        ]
    }
}
```

TAG	DESCRIPTION	TYPE	REQUIRED

TAG	DESCRIPTION	TYPE	REQUIRED
name	Name of the pipeline. Specify a name that represents the action that the pipeline performs. • Maximum number of characters: 140 • Must start with a letter, number, or an underscore (_) • Following characters are not allowed: ".", "+", "?", "/", "<","<","*"," %","&",":"," "	String	Yes
description	Specify the text describing what the pipeline is used for.	String	No
activities	The **activities** section can have one or more activities defined within it. See the Activity JSON section for details about the activities JSON element.	Array	Yes
parameters	The **parameters** section can have one or more parameters defined within the pipeline, making your pipeline flexible for reuse.	List	No
concurrency	The maximum number of concurrent runs the pipeline can have. By default, there is no maximum. If the concurrency limit is reached, additional pipeline runs are queued until earlier ones complete	Number	No
annotations	A list of tags associated with the pipeline	Array	No

Activity JSON

The **activities** section can have one or more activities defined within it. There are two main types of activities: Execution and Control Activities.

Execution activities

Execution activities include data movement and data transformation activities. They have the following top-level structure:

```
{
    "name": "Execution Activity Name",
    "description": "description",
    "type": "<ActivityType>",
    "typeProperties":
    {
    },
    "linkedServiceName": "MyLinkedService",
    "policy":
    {
    },
    "dependsOn":
    {
    }
}
```

Following table describes properties in the activity JSON definition:

TAG	DESCRIPTION	REQUIRED
name	Name of the activity. Specify a name that represents the action that the activity performs. • Maximum number of characters: 55 • Must start with a letter-number, or an underscore (_) • Following characters are not allowed: “.”, "+", "?", "/", "<",">","*"," %"," &","."," "	Yes
description	Text describing what the activity or is used for	Yes
type	Type of the activity. See the Data Movement Activities, Data Transformation Activities, and Control Activities sections for different types of activities.	Yes
linkedServiceName	Name of the linked service used by the activity. An activity may require that you specify the linked service that links to the required compute environment.	Yes for HDInsight Activity, ML Studio (classic) Batch Scoring Activity, Stored Procedure Activity. No for all others
typeProperties	Properties in the typeProperties section depend on each type of activity. To see type properties for an activity, click links to the activity in the previous section.	No
policy	Policies that affect the run-time behavior of the activity. This property includes a timeout and retry behavior. If it isn't specified, default values are used. For more information, see Activity policy section.	No

TAG	DESCRIPTION	REQUIRED
dependsOn	This property is used to define activity dependencies, and how subsequent activities depend on previous activities. For more information, see Activity dependency	No

Activity policy

Policies affect the run-time behavior of an activity, giving configuration options. Activity Policies are only available for execution activities.

Activity policy JSON definition

```
{
    "name": "MyPipelineName",
    "properties": {
      "activities": [
        {
          "name": "MyCopyBlobtoSqlActivity",
          "type": "Copy",
          "typeProperties": {
            ...
          },
         "policy": {
            "timeout": "00:10:00",
            "retry": 1,
            "retryIntervalInSeconds": 60,
            "secureOutput": true
         }
        }
      ],
        "parameters": {
           ...
        }
    }
}
```

JSON NAME	DESCRIPTION	ALLOWED VALUES	REQUIRED
timeout	Specifies the timeout for the activity to run.	Timespan	No. Default timeout is 7 days.
retry	Maximum retry attempts	Integer	No. Default is 0
retryIntervalInSeconds	The delay between retry attempts in seconds	Integer	No. Default is 30 seconds
secureOutput	When set to true, the output from activity is considered as secure and aren't logged for monitoring.	Boolean	No. Default is false.

Control activity

Control activities have the following top-level structure:

```
{
    "name": "Control Activity Name",
    "description": "description",
    "type": "<ActivityType>",
    "typeProperties":
    {
    },
    "dependsOn":
    {
    }
}
```

TAG	DESCRIPTION	REQUIRED
name	Name of the activity. Specify a name that represents the action that the activity performs. • Maximum number of characters: 55 • Must start with a letter number, or an underscore (_) • Following characters are not allowed: ".", "+", "?", "/", "<","<","*"," %"," &",":"," "	Yes
description	Text describing what the activity or is used for	Yes
type	Type of the activity. See the data movement activities, data transformation activities, and control activities sections for different types of activities.	Yes
typeProperties	Properties in the typeProperties section depend on each type of activity. To see type properties for an activity, click links to the activity in the previous section.	No
dependsOn	This property is used to define Activity Dependency, and how subsequent activities depend on previous activities. For more information, see activity dependency.	No

Activity dependency

Activity Dependency defines how subsequent activities depend on previous activities, determining the condition of whether to continue executing the next task. An activity can depend on one or multiple previous activities with different dependency conditions.

The different dependency conditions are: Succeeded, Failed, Skipped, Completed.

For example, if a pipeline has Activity A -> Activity B, the different scenarios that can happen are:

- Activity B has dependency condition on Activity A with **succeeded**: Activity B only runs if Activity A has a final status of succeeded
- Activity B has dependency condition on Activity A with **failed**: Activity B only runs if Activity A has a final

status of failed

- Activity B has dependency condition on Activity A with **completed**: Activity B runs if Activity A has a final status of succeeded or failed
- Activity B has a dependency condition on Activity A with **skipped**: Activity B runs if Activity A has a final status of skipped. Skipped occurs in the scenario of Activity X -> Activity Y -> Activity Z, where each activity runs only if the previous activity succeeds. If Activity X fails, then Activity Y has a status of "Skipped" because it never executes. Similarly, Activity Z has a status of "Skipped" as well.

Example: Activity 2 depends on the Activity 1 succeeding

```
{
    "name": "PipelineName",
    "properties":
    {
        "description": "pipeline description",
        "activities": [
         {
            "name": "MyFirstActivity",
            "type": "Copy",
            "typeProperties": {
            },
            "linkedServiceName": {
            }
        },
        {
            "name": "MySecondActivity",
            "type": "Copy",
            "typeProperties": {
            },
            "linkedServiceName": {
            },
            "dependsOn": [
            {
                "activity": "MyFirstActivity",
                "dependencyConditions": [
                    "Succeeded"
                ]
            }
          ]
        }
      ],
      "parameters": {
       }
    }
}
```

Sample copy pipeline

In the following sample pipeline, there is one activity of type **Copy** in the **activities** section. In this sample, the copy activity copies data from an Azure Blob storage to a database in Azure SQL Database.

```
{
  "name": "CopyPipeline",
  "properties": {
    "description": "Copy data from a blob to Azure SQL table",
    "activities": [
      {
        "name": "CopyFromBlobToSQL",
        "type": "Copy",
        "inputs": [
          {
            "name": "InputDataset"
          }
        ],
        "outputs": [
          {
            "name": "OutputDataset"
          }
        ],
        "typeProperties": {
          "source": {
            "type": "BlobSource"
          },
          "sink": {
            "type": "SqlSink",
            "writeBatchSize": 10000,
            "writeBatchTimeout": "60:00:00"
          }
        },
        "policy": {
          "retry": 2,
          "timeout": "01:00:00"
        }
      }
    ]
  }
}
```

Note the following points:

- In the activities section, there is only one activity whose **type** is set to **Copy**.
- Input for the activity is set to **InputDataset** and output for the activity is set to **OutputDataset**. See Datasets article for defining datasets in JSON.
- In the **typeProperties** section, **BlobSource** is specified as the source type and **SqlSink** is specified as the sink type. In the data movement activities section, click the data store that you want to use as a source or a sink to learn more about moving data to/from that data store.

For a complete walkthrough of creating this pipeline, see Quickstart: create a Data Factory.

Sample transformation pipeline

In the following sample pipeline, there is one activity of type **HDInsightHive** in the **activities** section. In this sample, the HDInsight Hive activity transforms data from an Azure Blob storage by running a Hive script file on an Azure HDInsight Hadoop cluster.

```
{
    "name": "TransformPipeline",
    "properties": {
        "description": "My first Azure Data Factory pipeline",
        "activities": [
            {
                "type": "HDInsightHive",
                "typeProperties": {
                    "scriptPath": "adfgetstarted/script/partitionweblogs.hql",
                    "scriptLinkedService": "AzureStorageLinkedService",
                    "defines": {
                        "inputtable":
"wasb://adfgetstarted@<storageaccountname>.blob.core.windows.net/inputdata",
                        "partitionedtable":
"wasb://adfgetstarted@<storageaccountname>.blob.core.windows.net/partitioneddata"
                    }
                },
                "inputs": [
                    {
                        "name": "AzureBlobInput"
                    }
                ],
                "outputs": [
                    {
                        "name": "AzureBlobOutput"
                    }
                ],
                "policy": {
                    "retry": 3
                },
                "name": "RunSampleHiveActivity",
                "linkedServiceName": "HDInsightOnDemandLinkedService"
            }
        ]
    }
}
```

Note the following points:

- In the activities section, there is only one activity whose **type** is set to **HDInsightHive**.
- The Hive script file, **partitionweblogs.hql**, is stored in the Azure Storage account (specified by the scriptLinkedService, called AzureStorageLinkedService), and in script folder in the container `adfgetstarted`.
- The `defines` section is used to specify the runtime settings that are passed to the hive script as Hive configuration values (for example, $ `{hiveconf:inputtable}`, `${hiveconf:partitionedtable}`).

The **typeProperties** section is different for each transformation activity. To learn about type properties supported for a transformation activity, click the transformation activity in the Data transformation activities.

For a complete walkthrough of creating this pipeline, see Tutorial: transform data using Spark.

Multiple activities in a pipeline

The previous two sample pipelines have only one activity in them. You can have more than one activity in a pipeline. If you have multiple activities in a pipeline and subsequent activities are not dependent on previous activities, the activities may run in parallel.

You can chain two activities by using activity dependency, which defines how subsequent activities depend on previous activities, determining the condition whether to continue executing the next task. An activity can depend on one or more previous activities with different dependency conditions.

Scheduling pipelines

Pipelines are scheduled by triggers. There are different types of triggers (Scheduler trigger, which allows pipelines to be triggered on a wall-clock schedule, as well as the manual trigger, which triggers pipelines on-demand). For more information about triggers, see pipeline execution and triggers article.

To have your trigger kick off a pipeline run, you must include a pipeline reference of the particular pipeline in the trigger definition. Pipelines & triggers have an n-m relationship. Multiple triggers can kick off a single pipeline, and the same trigger can kick off multiple pipelines. Once the trigger is defined, you must start the trigger to have it start triggering the pipeline. For more information about triggers, see pipeline execution and triggers article.

For example, say you have a Scheduler trigger, "Trigger A," that I wish to kick off my pipeline, "MyCopyPipeline." You define the trigger, as shown in the following example:

Trigger A definition

```
{
  "name": "TriggerA",
  "properties": {
    "type": "ScheduleTrigger",
    "typeProperties": {
      ...
      }
    },
    "pipeline": {
      "pipelineReference": {
        "type": "PipelineReference",
        "referenceName": "MyCopyPipeline"
      },
      "parameters": {
        "copySourceName": "FileSource"
      }
    }
  }
}
```

Next steps

See the following tutorials for step-by-step instructions for creating pipelines with activities:

- Build a pipeline with a copy activity
- Build a pipeline with a data transformation activity

How to achieve CI/CD (continuous integration and delivery) using Azure Data Factory

- Continuous integration and delivery in Azure Data Factory

Linked services in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article describes what linked services are, how they're defined in JSON format, and how they're used in Azure Data Factory and Azure Synapse Analytics.

To learn more read the introductory article for Azure Data Factory or Azure Synapse.

Overview

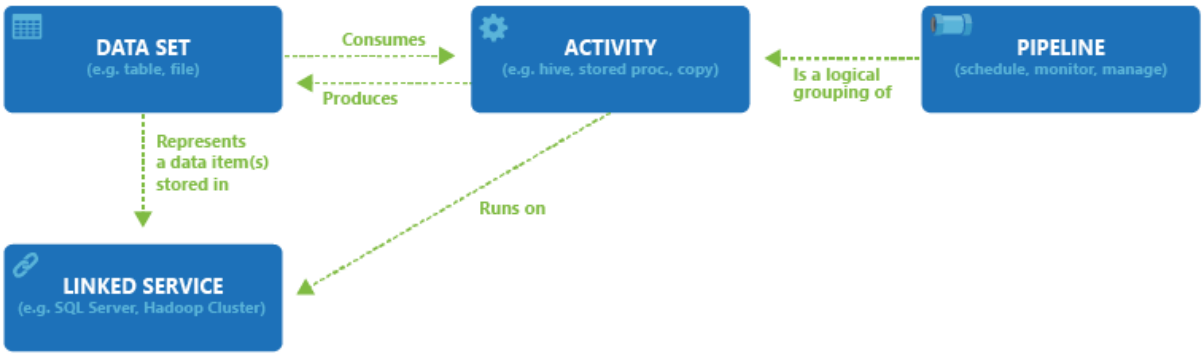
Azure Data Factory and Azure Synapse Analytics can have one or more pipelines. A **pipeline** is a logical grouping of **activities** that together perform a task. The activities in a pipeline define actions to perform on your data. For example, you might use a copy activity to copy data from SQL Server to Azure Blob storage. Then, you might use a Hive activity that runs a Hive script on an Azure HDInsight cluster to process data from Blob storage to produce output data. Finally, you might use a second copy activity to copy the output data to Azure Synapse Analytics, on top of which business intelligence (BI) reporting solutions are built. For more information about pipelines and activities, see Pipelines and activities.

Now, a **dataset** is a named view of data that simply points or references the data you want to use in your **activities** as inputs and outputs.

Before you create a dataset, you must create a **linked service** to link your data store to the Data Factory or Synapse Workspace. Linked services are much like connection strings, which define the connection information needed for the service to connect to external resources. Think of it this way; the dataset represents the structure of the data within the linked data stores, and the linked service defines the connection to the data source. For example, an Azure Storage linked service links a storage account to the service. An Azure Blob dataset represents the blob container and the folder within that Azure Storage account that contains the input blobs to be processed.

Here is a sample scenario. To copy data from Blob storage to a SQL Database, you create two linked services: Azure Storage and Azure SQL Database. Then, create two datasets: Azure Blob dataset (which refers to the Azure Storage linked service) and Azure SQL Table dataset (which refers to the Azure SQL Database linked service). The Azure Storage and Azure SQL Database linked services contain connection strings that the service uses at runtime to connect to your Azure Storage and Azure SQL Database, respectively. The Azure Blob dataset specifies the blob container and blob folder that contains the input blobs in your Blob storage. The Azure SQL Table dataset specifies the SQL table in your SQL Database to which the data is to be copied.

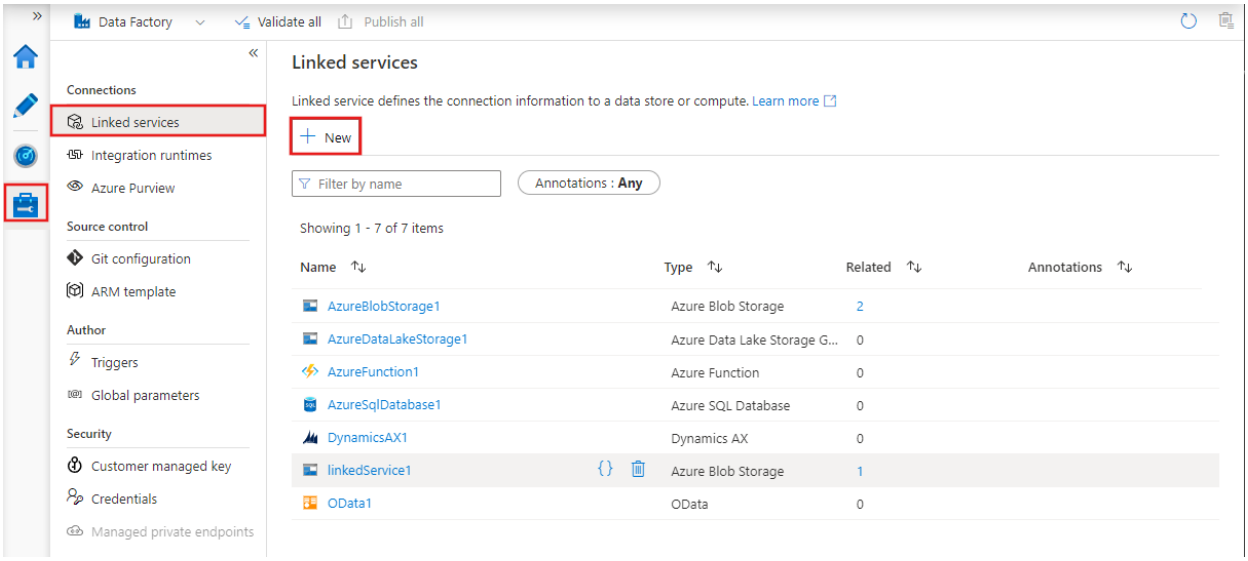
The following diagram shows the relationships among pipeline, activity, dataset, and linked service in the service:



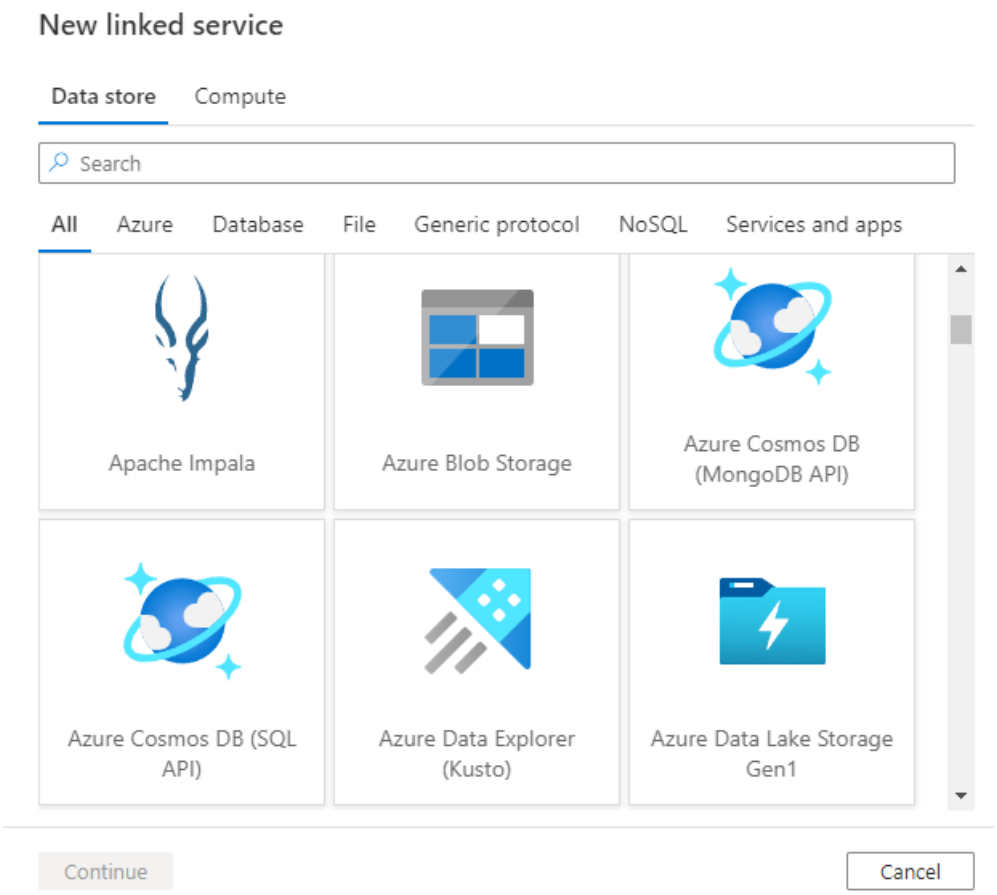
Linked service with UI

- Azure Data Factory
- Synapse Analytics

To create a new linked service in Azure Data Factory Studio, select the **Manage** tab and then **linked services**, where you can see any existing linked services you defined. Select **New** to create a new linked service.



After selecting New to create a new linked service you will be able to choose any of the supported connectors and configure its details accordingly. Thereafter you can use the linked service in any pipelines you create.



Linked service JSON

A linked service is defined in JSON format as follows:

```
{
    "name": "<Name of the linked service>",
    "properties": {
        "type": "<Type of the linked service>",
        "typeProperties": {
              "<data store or compute-specific type properties>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

The following table describes properties in the above JSON:

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the linked service. See Naming rules.	Yes
type	Type of the linked service. For example: AzureBlobStorage (data store) or AzureBatch (compute). See the description for typeProperties.	Yes

PROPERTY	DESCRIPTION	REQUIRED
typeProperties	The type properties are different for each data store or compute. For the supported data store types and their type properties, see the connector overview article. Navigate to the data store connector article to learn about type properties specific to a data store. For the supported compute types and their type properties, see Compute linked services.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is located in a private network). If not specified, it uses the default Azure Integration Runtime.	No

Linked service example

The following linked service is an Azure Blob storage linked service. Notice that the type is set to Azure Blob storage. The type properties for the Azure Blob storage linked service include a connection string. The service uses this connection string to connect to the data store at runtime.

```
{
    "name": "AzureBlobStorageLinkedService",
    "properties": {
        "type": "AzureBlobStorage",
        "typeProperties": {
            "connectionString": "DefaultEndpointsProtocol=https;AccountName=<accountname>;AccountKey=
<accountkey>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Create linked services

Linked services can be created in the Azure Data Factory UX via the management hub and any activities, datasets, or data flows that reference them.

You can create linked services by using one of these tools or SDKs: .NET API, PowerShell, REST API, Azure Resource Manager Template, and Azure portal.

Data store linked services

You can find the list of supported data stores in the connector overview article. Click a data store to learn the supported connection properties.

Compute linked services

Reference compute environments supported for details about different compute environments you can connect to from your service as well as the different configurations.

Next steps

- Learn how to use credentials from a user-assigned managed identity in a linked service.

See the following tutorials for step-by-step instructions for creating pipelines and datasets by using one of these tools or SDKs.

- Quickstart: create a Data Factory using .NET
- Quickstart: create a Data Factory using PowerShell
- Quickstart: create a Data Factory using REST API
- Quickstart: create a Data Factory using Azure portal

Datasets in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article describes what datasets are, how they're defined in JSON format, and how they're used in Azure Data Factory and Synapse pipelines.

If you're new to Data Factory, see Introduction to Azure Data Factory for an overview. For more information about Azure Synapse, see What is Azure Synapse

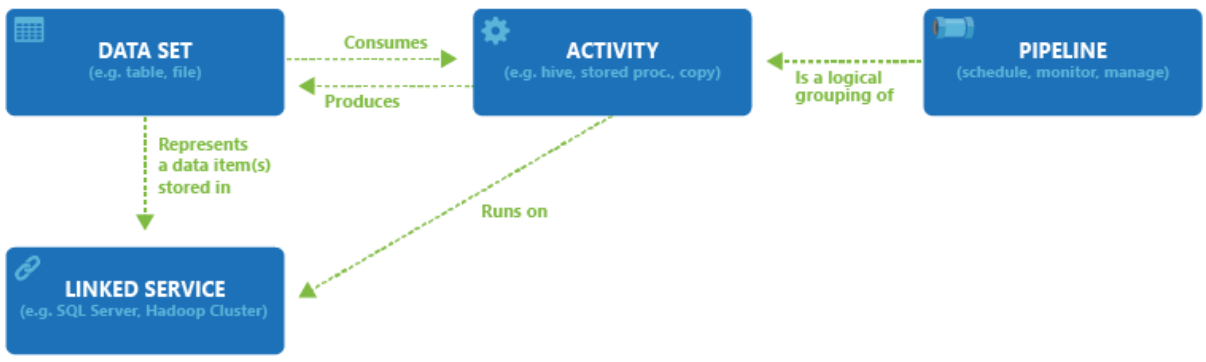
Overview

An Azure Data Factory or Synapse workspace can have one or more pipelines. A **pipeline** is a logical grouping of **activities** that together perform a task. The activities in a pipeline define actions to perform on your data. Now, a **dataset** is a named view of data that simply points or references the data you want to use in your **activities** as inputs and outputs. Datasets identify data within different data stores, such as tables, files, folders, and documents. For example, an Azure Blob dataset specifies the blob container and folder in Blob Storage from which the activity should read the data.

Before you create a dataset, you must create a **linked service** to link your data store to the service. Linked services are much like connection strings, which define the connection information needed for the service to connect to external resources. Think of it this way; the dataset represents the structure of the data within the linked data stores, and the linked service defines the connection to the data source. For example, an Azure Storage linked service links a storage account. An Azure Blob dataset represents the blob container and the folder within that Azure Storage account that contains the input blobs to be processed.

Here's a sample scenario. To copy data from Blob storage to a SQL Database, you create two linked services: Azure Blob Storage and Azure SQL Database. Then, create two datasets: Delimited Text dataset (which refers to the Azure Blob Storage linked service, assuming you have text files as source) and Azure SQL Table dataset (which refers to the Azure SQL Database linked service). The Azure Blob Storage and Azure SQL Database linked services contain connection strings that the service uses at runtime to connect to your Azure Storage and Azure SQL Database, respectively. The Delimited Text dataset specifies the blob container and blob folder that contains the input blobs in your Blob Storage, along with format-related settings. The Azure SQL Table dataset specifies the SQL table in your SQL Database to which the data is to be copied.

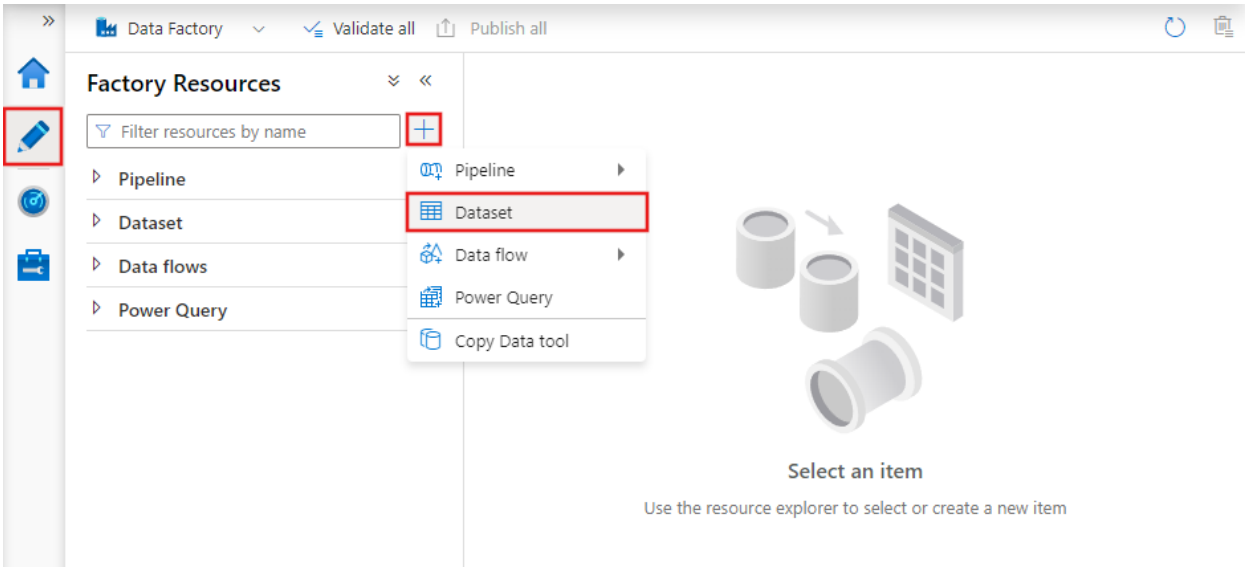
The following diagram shows the relationships among pipeline, activity, dataset, and linked services:



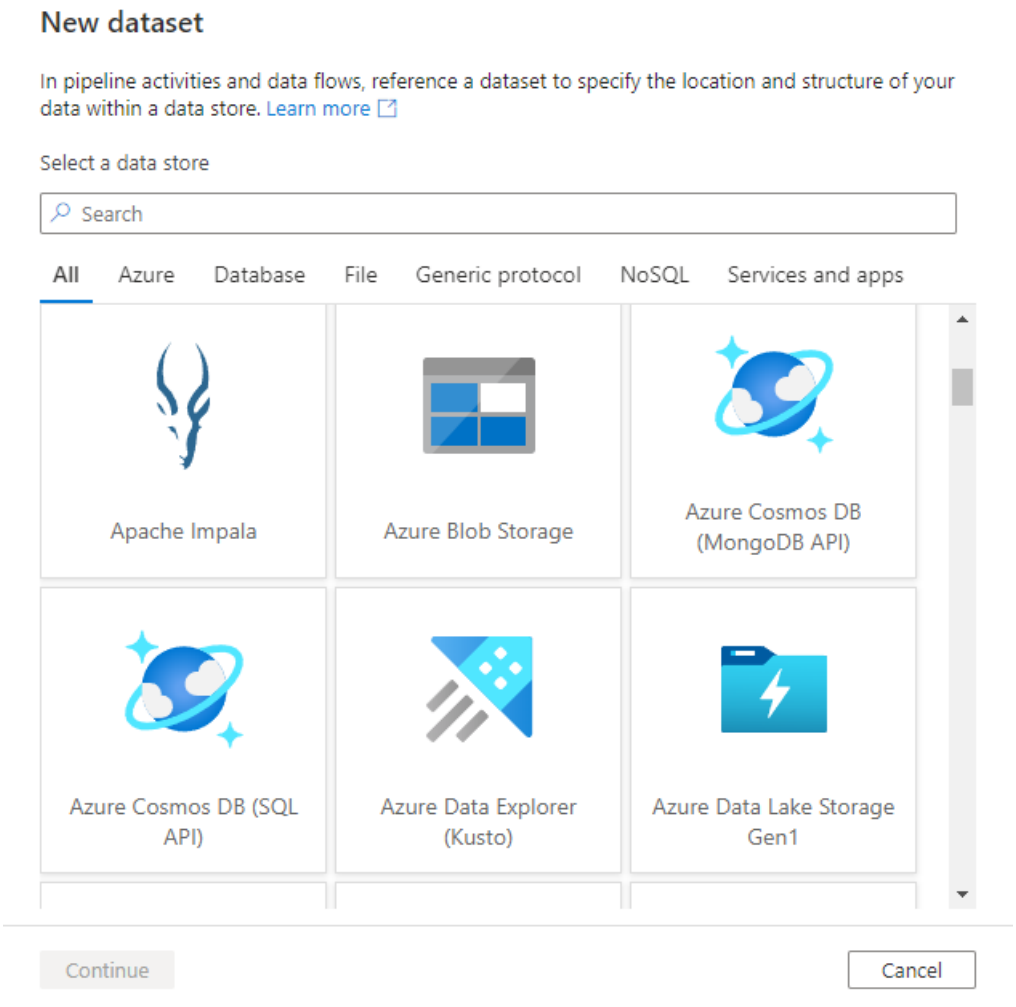
Create a dataset with UI

- Azure Data Factory
- Synapse Analytics

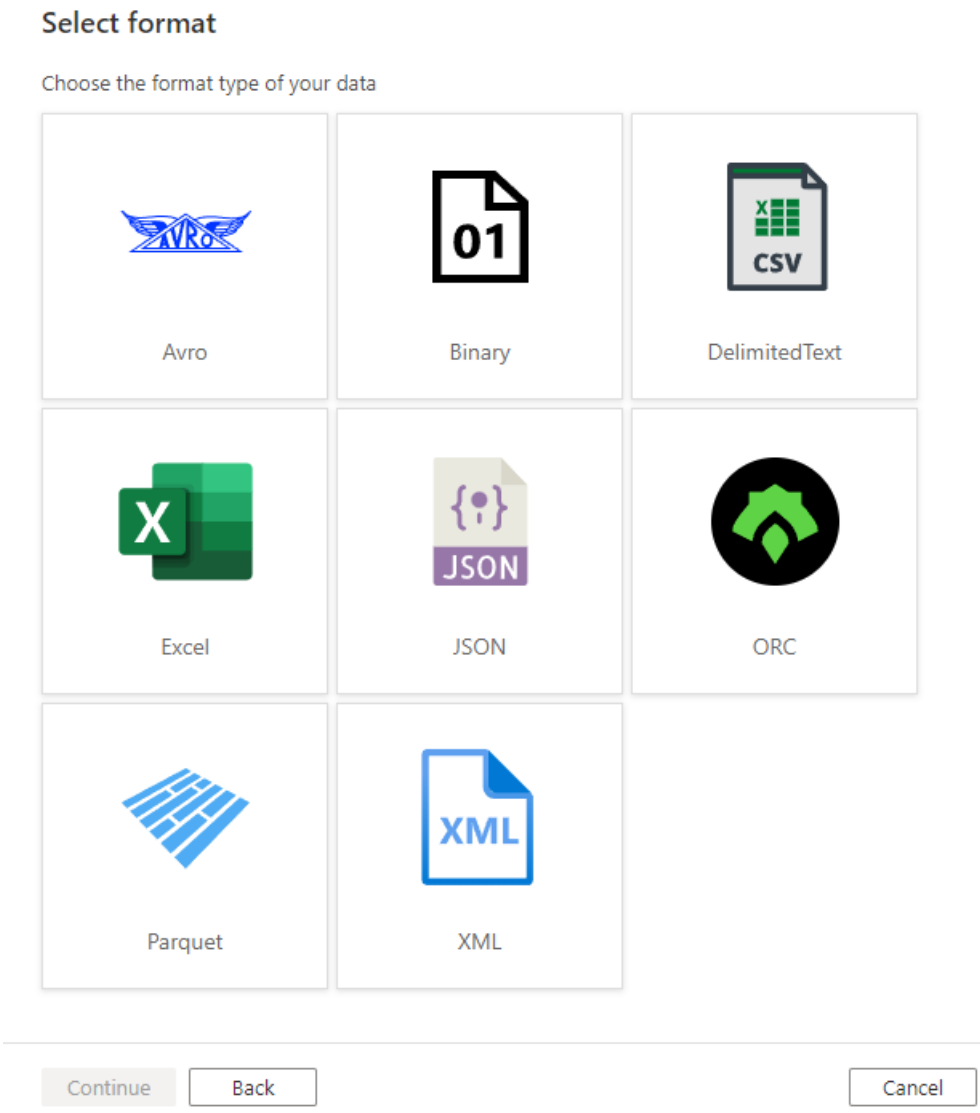
To create a dataset with the Azure Data Factory Studio, select the Author tab (with the pencil icon), and then the plus sign icon, to choose **Dataset**.



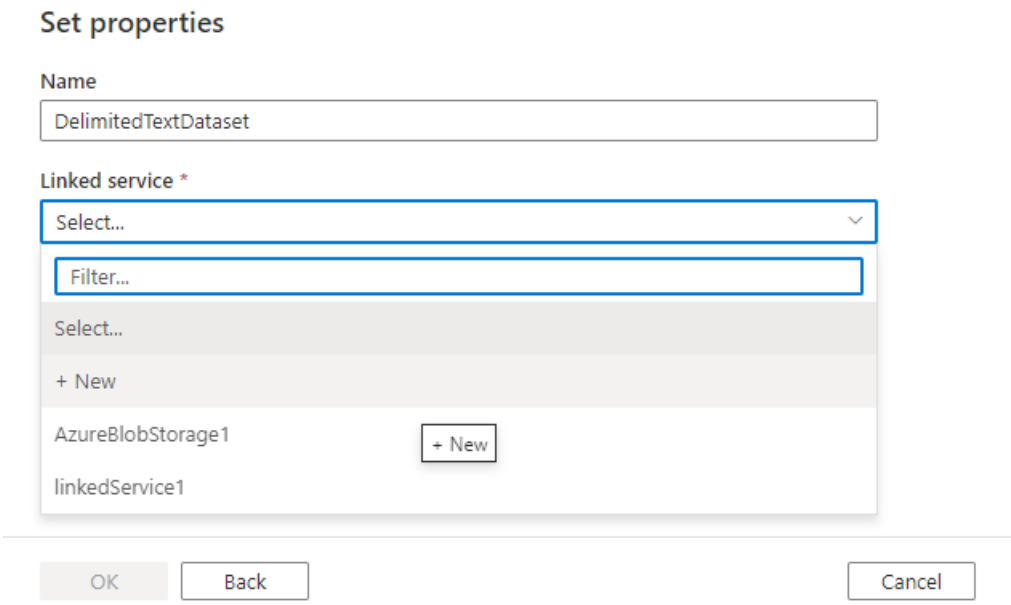
You'll see the new dataset window to choose any of the connectors available in Azure Data Factory, to set up an existing or new linked service.



Next you'll be prompted to choose the dataset format.



Finally, you can choose an existing linked service of the type you selected for the dataset, or create a new one if one isn't already defined.



Once you create the dataset, you can use it within any pipelines in the Azure Data Factory.

Dataset JSON

A dataset is defined in the following JSON format:

```
{
    "name": "<name of dataset>",
    "properties": {
        "type": "<type of dataset: DelimitedText, AzureSqlTable etc...>",
        "linkedServiceName": {
                "referenceName": "<name of linked service>",
                "type": "LinkedServiceReference",
        },
        "schema":[

        ],
        "typeProperties": {
            "<type specific property>": "<value>",
            "<type specific property 2>": "<value 2>",
        }
    }
}
```

The following table describes properties in the above JSON:

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the dataset. See Naming rules.	Yes
type	Type of the dataset. Specify one of the types supported by Data Factory (for example: DelimitedText, AzureSqlTable). For details, see Dataset types.	Yes
schema	Schema of the dataset, represents the physical data type and shape.	No
typeProperties	The type properties are different for each type. For details on the supported types and their properties, see Dataset type.	Yes

When you import the schema of dataset, select the **Import Schema** button and choose to import from the source or from a local file. In most cases, you'll import the schema directly from the source. But if you already have a local schema file (a Parquet file or CSV with headers), you can direct the service to base the schema on that file.

In copy activity, datasets are used in source and sink. Schema defined in dataset is optional as reference. If you want to apply column/field mapping between source and sink, refer to Schema and type mapping.

In Data Flow, datasets are used in source and sink transformations. The datasets define the basic data schemas. If your data has no schema, you can use schema drift for your source and sink. Metadata from the datasets appears in your source transformation as the source projection. The projection in the source transformation represents the Data Flow data with defined names and types.

Dataset type

The service supports many different types of datasets, depending on the data stores you use. You can find the list of supported data stores from Connector overview article. Select a data store to learn how to create a linked service and a dataset for it.

For example, for a Delimited Text dataset, the dataset type is set to **DelimitedText** as shown in the following JSON sample:

```
{
    "name": "DelimitedTextInput",
    "properties": {
        "linkedServiceName": {
            "referenceName": "AzureBlobStorage",
            "type": "LinkedServiceReference"
        },
        "annotations": [],
        "type": "DelimitedText",
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "fileName": "input.log",
                "folderPath": "inputdata",
                "container": "adfgetstarted"
            },
            "columnDelimiter": ",",
            "escapeChar": "\\",
            "quoteChar": "\""
        },
        "schema": []
    }
}
```

Create datasets

You can create datasets by using one of these tools or SDKs: .NET API, PowerShell, REST API, Azure Resource Manager Template, and Azure portal

Current version vs. version 1 datasets

Here are some differences between datasets in Data Factory current version (and Azure Synapse), and the legacy Data Factory version 1:

- The external property isn't supported in the current version. It's replaced by a trigger.
- The policy and availability properties aren't supported in the current version. The start time for a pipeline depends on triggers.
- Scoped datasets (datasets defined in a pipeline) aren't supported in the current version.

Next steps

See the following tutorial for step-by-step instructions for creating pipelines and datasets by using one of these tools or SDKs.

- Quickstart: create a data factory using .NET
- Quickstart: create a data factory using PowerShell
- Quickstart: create a data factory using REST API
- Quickstart: create a data factory using Azure portal

Pipeline execution and triggers in Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 16 minutes to read • Edit Online

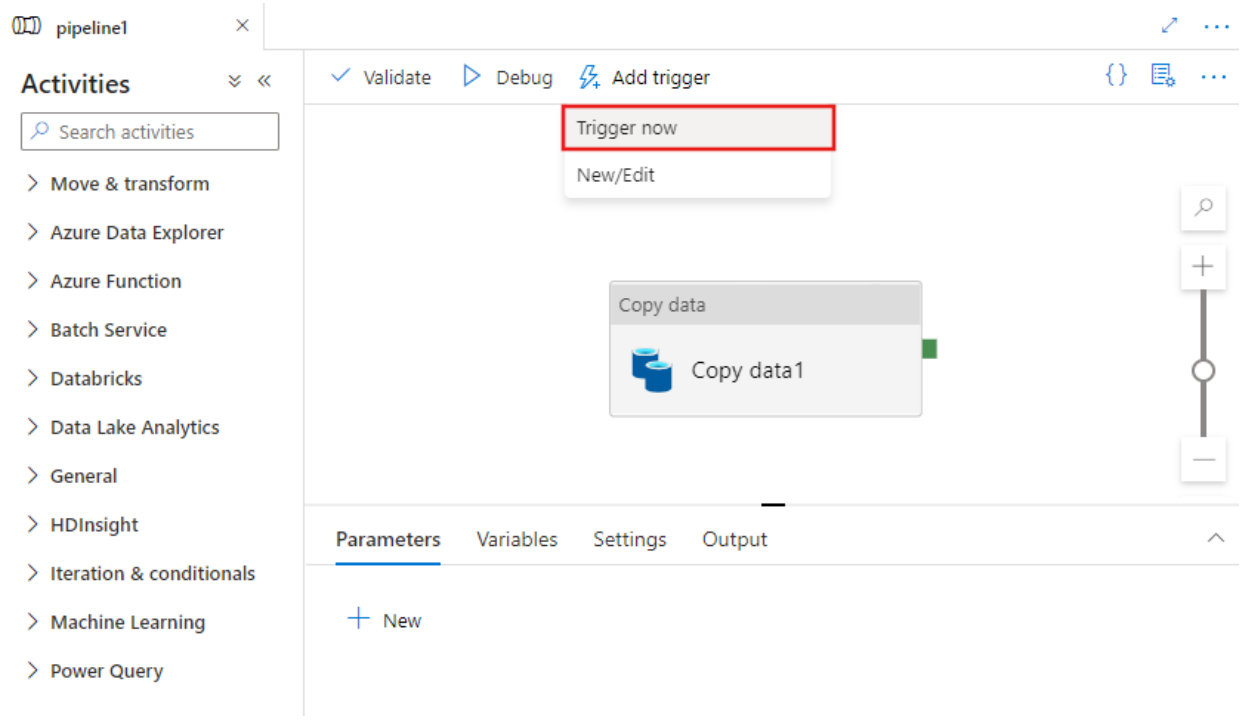
APPLIES TO: Azure Data Factory Azure Synapse Analytics

A *pipeline run* in Azure Data Factory and Azure Synapse defines an instance of a pipeline execution. For example, say you have a pipeline that executes at 8:00 AM, 9:00 AM, and 10:00 AM. In this case, there are three separate runs of the pipeline or pipeline runs. Each pipeline run has a unique pipeline run ID. A run ID is a GUID that uniquely defines that particular pipeline run.

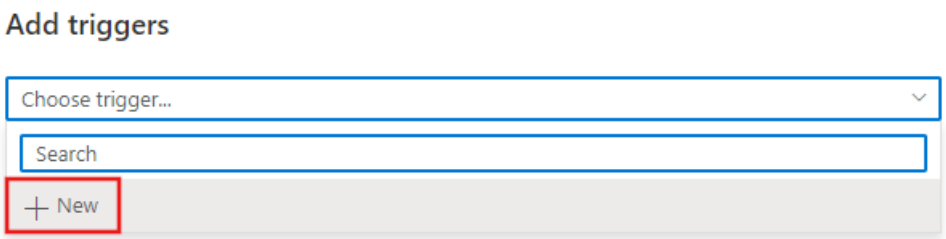
Pipeline runs are typically instantiated by passing arguments to parameters that you define in the pipeline. You can execute a pipeline either manually or by using a *trigger*. This article provides details about both ways of executing a pipeline.

Create triggers with UI

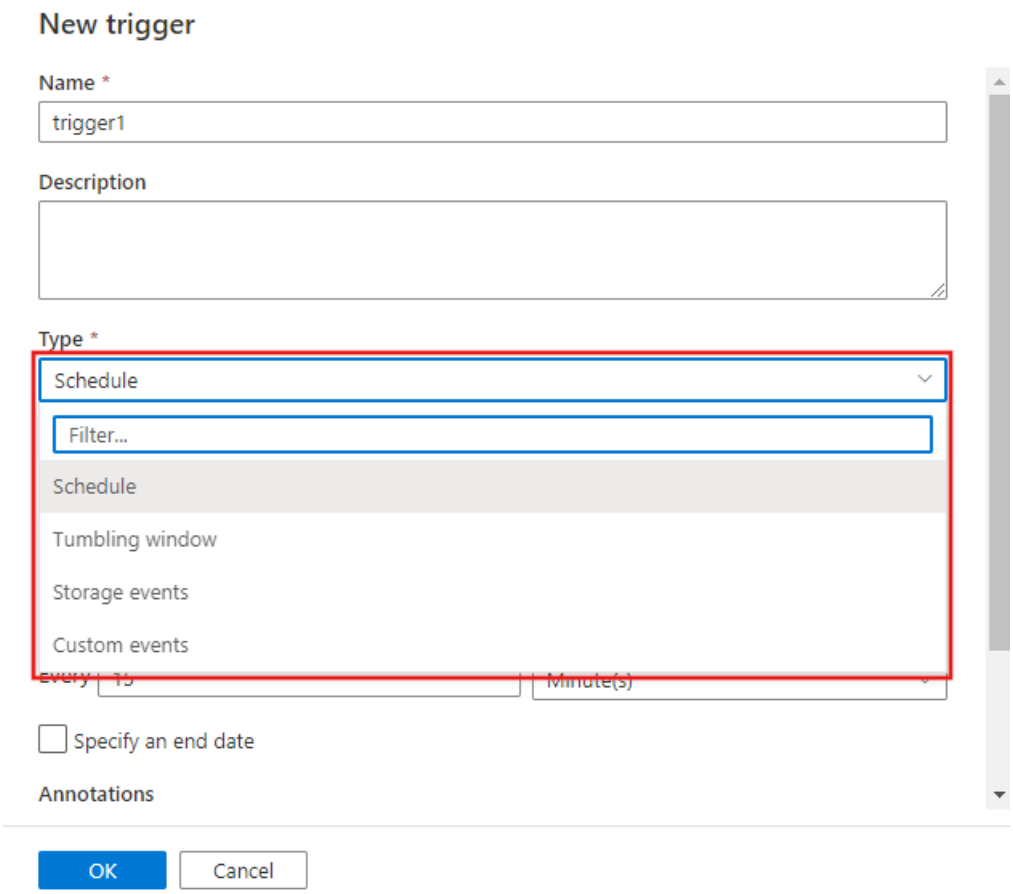
To manually trigger a pipeline or configure a new scheduled, tumbling window, storage event, or custom event trigger, select Add trigger at the top of the pipeline editor.



If you choose to manually trigger the pipeline, it will execute immediately. Otherwise if you choose New/Edit, you will be prompted with the add triggers window to either choose an existing trigger to edit, or create a new trigger.



You will see the trigger configuration window, allowing you to choose the trigger type.



Read more about scheduled, tumbling window, storage event, and custom event triggers below.

Manual execution (on-demand) with JSON

The manual execution of a pipeline is also referred to as *on-demand* execution.

For example, say you have a basic pipeline named **copyPipeline** that you want to execute. The pipeline has a single activity that copies from an Azure Blob storage source folder to a destination folder in the same storage. The following JSON definition shows this sample pipeline:

```
{
    "name": "copyPipeline",
    "properties": {
        "activities": [
            {
                "type": "Copy",
                "typeProperties": {
                    "source": {
                        "type": "BlobSource"
                    },
                    "sink": {
                        "type": "BlobSink"
                    }
                },
                "name": "CopyBlobtoBlob",
                "inputs": [
                    {
                        "referenceName": "sourceBlobDataset",
                        "type": "DatasetReference"
                    }
                ],
                "outputs": [
                    {
                        "referenceName": "sinkBlobDataset",
                        "type": "DatasetReference"
                    }
                ]
            }
        ],
        "parameters": {
            "sourceBlobContainer": {
                "type": "String"
            },
            "sinkBlobContainer": {
                "type": "String"
            }
        }
    }
}
```

In the JSON definition, the pipeline takes two parameters: **sourceBlobContainer** and **sinkBlobContainer**. You pass values to these parameters at runtime.

You can manually run your pipeline by using one of the following methods:

- .NET SDK
- Azure PowerShell module
- REST API
- Python SDK

REST API

The following sample command shows you how to run your pipeline by using the REST API manually:

```
POST
https://management.azure.com/subscriptions/mySubId/resourceGroups/myResourceGroup/providers/Microsoft.DataFa
ctory/factories/myDataFactory/pipelines/copyPipeline/createRun?api-version=2017-03-01-preview
```

For a complete sample, see Quickstart: Create a data factory by using the REST API.

Azure PowerShell

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

The following sample command shows you how to manually run your pipeline by using Azure PowerShell:

```
Invoke-AzDataFactoryV2Pipeline -DataFactory $df -PipelineName "Adfv2QuickStartPipeline" -ParameterFile
.\PipelineParameters.json
```

You pass parameters in the body of the request payload. In the .NET SDK, Azure PowerShell, and the Python SDK, you pass values in a dictionary that's passed as an argument to the call:

```
{
  "sourceBlobContainer": "MySourceFolder",
  "sinkBlobContainer": "MySinkFolder"
}
```

The response payload is a unique ID of the pipeline run:

```
{
  "runId": "0448d45a-a0bd-23f3-90a5-bfeea9264aed"
}
```

For a complete sample, see Quickstart: Create a data factory by using Azure PowerShell.

.NET SDK

The following sample call shows you how to run your pipeline by using the .NET SDK manually:

```
client.Pipelines.CreateRunWithHttpMessagesAsync(resourceGroup, dataFactoryName, pipelineName, parameters)
```

For a complete sample, see Quickstart: Create a data factory by using the .NET SDK.

> **NOTE**
> You can use the .NET SDK to invoke pipelines from Azure Functions, from your web services, and so on.

Trigger execution with JSON

Triggers are another way that you can execute a pipeline run. Triggers represent a unit of processing that determines when a pipeline execution needs to be kicked off. Currently, the service supports three types of triggers:

- Schedule trigger: A trigger that invokes a pipeline on a wall-clock schedule.
- Tumbling window trigger: A trigger that operates on a periodic interval, while also retaining state.
- Event-based trigger: A trigger that responds to an event.

Pipelines and triggers have a many-to-many relationship (except for the tumbling window trigger). Multiple triggers can kick off a single pipeline, or a single trigger can kick off multiple pipelines. In the following trigger definition, the **pipelines** property refers to a list of pipelines that are triggered by the particular trigger. The

property definition includes values for the pipeline parameters.

Basic trigger definition

```
{
    "properties": {
        "name": "MyTrigger",
        "type": "<type of trigger>",
        "typeProperties": {...},
        "pipelines": [
            {
                "pipelineReference": {
                    "type": "PipelineReference",
                    "referenceName": "<Name of your pipeline>"
                },
                "parameters": {
                    "<parameter 1 Name>": {
                        "type": "Expression",
                        "value": "<parameter 1 Value>"
                    },
                    "<parameter 2 Name>": "<parameter 2 Value>"
                }
            }
        ]
    }
}
```

Schedule trigger with JSON

A schedule trigger runs pipelines on a wall-clock schedule. This trigger supports periodic and advanced calendar options. For example, the trigger supports intervals like "weekly" or "Monday at 5:00 PM and Thursday at 9:00 PM." The schedule trigger is flexible because the dataset pattern is agnostic, and the trigger doesn't discern between time-series and non-time-series data.

For more information about schedule triggers and, for examples, see Create a trigger that runs a pipeline on a schedule.

Schedule trigger definition

When you create a schedule trigger, you specify scheduling and recurrence by using a JSON definition.

To have your schedule trigger kick off a pipeline run, include a pipeline reference of the particular pipeline in the trigger definition. Pipelines and triggers have a many-to-many relationship. Multiple triggers can kick off a single pipeline. A single trigger can kick off multiple pipelines.

```
{
  "properties": {
    "type": "ScheduleTrigger",
    "typeProperties": {
      "recurrence": {
        "frequency": <<Minute, Hour, Day, Week, Year>>,
        "interval": <<int>>, // How often to fire
        "startTime": <<datetime>>,
        "endTime": <<datetime>>,
        "timeZone": "UTC",
        "schedule": { // Optional (advanced scheduling specifics)
          "hours": [<<0-24>>],
          "weekDays": [<<Monday-Sunday>>],
          "minutes": [<<0-60>>],
          "monthDays": [<<1-31>>],
          "monthlyOccurrences": [
            {
              "day": <<Monday-Sunday>>,
              "occurrence": <<1-5>>
            }
          ]
        }
      }
    },
  "pipelines": [
    {
      "pipelineReference": {
        "type": "PipelineReference",
        "referenceName": "<Name of your pipeline>"
      },
      "parameters": {
        "<parameter 1 Name>": {
          "type": "Expression",
          "value": "<parameter 1 Value>"
        },
        "<parameter 2 Name>": "<parameter 2 Value>"
      }
    }
  ]}
}
```

IMPORTANT

The **parameters** property is a mandatory property of the **pipelines** element. If your pipeline doesn't take any parameters, you must include an empty JSON definition for the **parameters** property.

Schema overview

The following table provides a high-level overview of the major schema elements that are related to recurrence and scheduling a trigger:

JSON PROPERTY	DESCRIPTION
startTime	A date-time value. For basic schedules, the value of the **startTime** property applies to the first occurrence. For complex schedules, the trigger starts no sooner than the specified **startTime** value.
endTime	The end date and time for the trigger. The trigger doesn't execute after the specified end date and time. The value for the property can't be in the past.

JSON PROPERTY	DESCRIPTION
timeZone	The time zone. For a list of supported time zones, see Create a trigger that runs a pipeline on a schedule.
recurrence	A recurrence object that specifies the recurrence rules for the trigger. The recurrence object supports the **frequency**, **interval**, **endTime**, **count**, and **schedule** elements. When a recurrence object is defined, the **frequency** element is required. The other elements of the recurrence object are optional.
frequency	The unit of frequency at which the trigger recurs. The supported values include "minute", "hour", "day", "week", and "month".
interval	A positive integer that denotes the interval for the **frequency** value. The **frequency** value determines how often the trigger runs. For example, if the **interval** is 3 and the **frequency** is "week", the trigger recurs every three weeks.
schedule	The recurrence schedule for the trigger. A trigger with a specified **frequency** value alters its recurrence based on a recurrence schedule. The **schedule** property contains modifications for the recurrence that are based on minutes, hours, weekdays, month days, and week number.

Schedule trigger example

```
{
  "properties": {
    "name": "MyTrigger",
    "type": "ScheduleTrigger",
    "typeProperties": {
      "recurrence": {
        "frequency": "Hour",
        "interval": 1,
        "startTime": "2017-11-01T09:00:00-08:00",
        "endTime": "2017-11-02T22:00:00-08:00"
      }
    },
    "pipelines": [{
        "pipelineReference": {
          "type": "PipelineReference",
          "referenceName": "SQLServerToBlobPipeline"
        },
        "parameters": {}
      },
      {
        "pipelineReference": {
          "type": "PipelineReference",
          "referenceName": "SQLServerToAzureSQLPipeline"
        },
        "parameters": {}
      }
    ]
  }
}
```

Schema defaults, limits, and examples

JSON PROPERTY	TYPE	REQUIRED	DEFAULT VALUE	VALID VALUES	EXAMPLE
startTime	string	Yes	None	ISO 8601 date-times	`"startTime" : "2013-01-09T09:30:00-08:00"`
recurrence	object	Yes	None	A recurrence object	`"recurrence" : { "frequency" : "monthly", "interval" : 1 }`
interval	number	No	1	1 to 1000	`"interval":10`
endTime	string	Yes	None	A date-time value that represents a time in the future	`"endTime" : "2013-02-09T09:30:00-08:00"`
schedule	object	No	None	A schedule object	`"schedule" : { "minute" : [30], "hour" : [8,17] }`

startTime property

The following table shows you how the **startTime** property controls a trigger run:

STARTTIME VALUE	RECURRENCE WITHOUT SCHEDULE	RECURRENCE WITH SCHEDULE
Start time is in the past	Calculates the first future execution time after the start time, and runs at that time. Runs subsequent executions calculated from the last execution time. See the example that follows this table.	The trigger starts *no sooner than* the specified start time. The first occurrence is based on the schedule, calculated from the start time. Runs subsequent executions based on the recurrence schedule.
Start time is in the future or the current time	Runs once at the specified start time. Runs subsequent executions calculated from the last execution time.	The trigger starts *no sooner* than the specified start time. The first occurrence is based on the schedule, calculated from the start time. Runs subsequent executions based on the recurrence schedule.

Let's look at an example of what happens when the start time is in the past, with a recurrence, but no schedule. Assume that the current time is 2017-04-08 13:00, the start time is 2017-04-07 14:00, and the recurrence is every two days. (The **recurrence** value is defined by setting the **frequency** property to "day" and the **interval** property to 2.) Notice that the **startTime** value is in the past and occurs before the current time.

Under these conditions, the first execution is 2017-04-09 at 14:00. The Scheduler engine calculates execution occurrences from the start time. Any instances in the past are discarded. The engine uses the next instance that occurs in the future. In this scenario, the start time is 2017-04-07 at 2:00 PM. The next instance is two days from that time, which is on 2017-04-09 at 2:00 PM.

The first execution time is the same even whether **startTime** is 2017-04-05 14:00 or 2017-04-01 14:00. After the first execution, subsequent executions are calculated by using the schedule. Therefore, the subsequent

executions are on 2017-04-11 at 2:00 PM, then on 2017-04-13 at 2:00 PM, then on 2017-04-15 at 2:00 PM, and so on.

Finally, when hours or minutes aren't set in the schedule for a trigger, the hours or minutes of the first execution are used as defaults.

schedule property

You can use **schedule** to *limit* the number of trigger executions. For example, if a trigger with a monthly frequency is scheduled to run only on day 31, the trigger runs only in those months that have a thirty-first day.

You can also use **schedule** to *expand* the number of trigger executions. For example, a trigger with a monthly frequency that's scheduled to run on month days 1 and 2, runs on the first and second days of the month, rather than once a month.

If multiple **schedule** elements are specified, the order of evaluation is from the largest to the smallest schedule setting: week number, month day, weekday, hour, minute.

The following table describes the **schedule** elements in detail:

JSON ELEMENT	DESCRIPTION	VALID VALUES
minutes	Minutes of the hour at which the trigger runs.	- Integer - Array of integers
hours	Hours of the day at which the trigger runs.	- Integer - Array of integers
weekDays	Days of the week the trigger runs. The value can be specified only with a weekly frequency.	- Monday - Tuesday - Wednesday - Thursday - Friday - Saturday - Sunday - Array of day values (maximum array size is 7) Day values aren't case-sensitive
monthlyOccurrences	Days of the month on which the trigger runs. The value can be specified with a monthly frequency only.	- Array of **monthlyOccurrence** objects: `{ "day": day, "occurrence": occurrence }` - The **day** attribute is the day of the week on which the trigger runs. For example, a **monthlyOccurrences** property with a **day** value of `{Sunday}` means every Sunday of the month. The **day** attribute is required. - The **occurrence** attribute is the occurrence of the specified **day** during the month. For example, a **monthlyOccurrences** property with **day** and **occurrence** values of `{Sunday, -1}` means the last Sunday of the month. The **occurrence** attribute is optional.

JSON ELEMENT	DESCRIPTION	VALID VALUES
monthDays	Day of the month on which the trigger runs. The value can be specified with a monthly frequency only.	- Any value <= -1 and >= -31 - Any value >= 1 and <= 31 - Array of values

Tumbling window trigger

Tumbling window triggers are a type of trigger that fires at a periodic time interval from a specified start time, while retaining state. Tumbling windows are a series of fixed-sized, non-overlapping, and contiguous time intervals.

For more information about tumbling window triggers and, for examples, see Create a tumbling window trigger.

Examples of trigger recurrence schedules

This section provides examples of recurrence schedules. It focuses on the **schedule** object and its elements.

The examples assume that the **interval** value is 1 and that the **frequency** value is correct according to the schedule definition. For example, you can't have a **frequency** value of "day" and also have a **monthDays** modification in the **schedule** object. These kinds of restrictions are described in the table in the preceding section.

EXAMPLE	DESCRIPTION
`{"hours":[5]}`	Run at 5:00 AM every day.
`{"minutes":[15], "hours":[5]}`	Run at 5:15 AM every day.
`{"minutes":[15], "hours":[5,17]}`	Run at 5:15 AM and 5:15 PM every day.
`{"minutes":[15,45], "hours":[5,17]}`	Run at 5:15 AM, 5:45 AM, 5:15 PM, and 5:45 PM every day.
`{"minutes":[0,15,30,45]}`	Run every 15 minutes.
`{hours":[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]}`	Run every hour. This trigger runs every hour. The minutes are controlled by the **startTime** value, when a value is specified. If a value isn't specified, the minutes are controlled by the creation time. For example, if the start time or creation time (whichever applies) is 12:25 PM, the trigger runs at 00:25, 01:25, 02:25, ..., and 23:25. This schedule is equivalent to having a trigger with a **frequency** value of "hour", an **interval** value of 1, and no **schedule**. This schedule can be used with different **frequency** and **interval** values to create other triggers. For example, when the **frequency** value is "month", the schedule runs only once a month, rather than every day when the **frequency** value is "day".

EXAMPLE	DESCRIPTION
`{"minutes":[0]}`	Run every hour on the hour. This trigger runs every hour on the hour starting at 12:00 AM, 1:00 AM, 2:00 AM, and so on. This schedule is equivalent to a trigger with a **frequency** value of "hour" and a **startTime** value of zero minutes, and no **schedule** but a **frequency** value of "day". If the **frequency** value is "week" or "month", the schedule executes one day a week or one day a month only, respectively.
`{"minutes":[15]}`	Run at 15 minutes past every hour. This trigger runs every hour at 15 minutes past the hour starting at 00:15 AM, 1:15 AM, 2:15 AM, and so on, and ending at 11:15 PM.
`{"hours":[17], "weekDays":["saturday"]}`	Run at 5:00 PM on Saturdays every week.
`{"hours":[17], "weekDays":["monday", "wednesday", "friday"]}`	Run at 5:00 PM on Monday, Wednesday, and Friday every week.
`{"minutes":[15,45], "hours":[17], "weekDays": ["monday", "wednesday", "friday"]}`	Run at 5:15 PM and 5:45 PM on Monday, Wednesday, and Friday every week.
`{"minutes":[0,15,30,45], "weekDays":["monday", "tuesday", "wednesday", "thursday", "friday"]}`	Run every 15 minutes on weekdays.
`{"minutes":[0,15,30,45], "hours": [9, 10, 11, 12, 13, 14, 15, 16] "weekDays":["monday", "tuesday", "wednesday", "thursday", "friday"]}`	Run every 15 minutes on weekdays between 9:00 AM and 4:45 PM.
`{"weekDays":["tuesday", "thursday"]}`	Run on Tuesdays and Thursdays at the specified start time.
`{"minutes":[0], "hours":[6], "monthDays":[28]}`	Run at 6:00 AM on the twenty-eighth day of every month (assuming a **frequency** value of "month").
`{"minutes":[0], "hours":[6], "monthDays":[-1]}`	Run at 6:00 AM on the last day of the month. To run a trigger on the last day of a month, use -1 instead of day 28, 29, 30, or 31.
`{"minutes":[0], "hours":[6], "monthDays":[1,-1]}`	Run at 6:00 AM on the first and last day of every month.
`{monthDays":[1,14]}`	Run on the first and fourteenth day of every month at the specified start time.
`{"minutes":[0], "hours":[5], "monthlyOccurrences": [{"day":"friday", "occurrence":1}]}`	Run on the first Friday of every month at 5:00 AM.
`{"monthlyOccurrences":[{"day":"friday", "occurrence":1}]}`	Run on the first Friday of every month at the specified start time.

EXAMPLE	DESCRIPTION
`{"monthlyOccurrences":[{"day":"friday", "occurrence":-3}]}`	Run on the third Friday from the end of the month, every month, at the specified start time.
`{"minutes":[15], "hours":[5], "monthlyOccurrences": [{"day":"friday", "occurrence":1},{"day":"friday", "occurrence":-1}]}`	Run on the first and last Friday of every month at 5:15 AM.
`{"monthlyOccurrences":[{"day":"friday", "occurrence":1},{"day":"friday", "occurrence":-1}]}`	Run on the first and last Friday of every month at the specified start time.
`{"monthlyOccurrences":[{"day":"friday", "occurrence":5}]}`	Run on the fifth Friday of every month at the specified start time. When there's no fifth Friday in a month, the pipeline doesn't run. To run the trigger on the last occurring Friday of the month, consider using -1 instead of 5 for the **occurrence** value.
`{"minutes":[0,15,30,45], "monthlyOccurrences": [{"day":"friday", "occurrence":-1}]}`	Run every 15 minutes on the last Friday of the month.
`{"minutes":[15,45], "hours":[5,17], "monthlyOccurrences":[{"day":"wednesday", "occurrence":3}]}`	Run at 5:15 AM, 5:45 AM, 5:15 PM, and 5:45 PM on the third Wednesday of every month.

Trigger type comparison

The tumbling window trigger and the schedule trigger both operate on time heartbeats. How are they different?

> **NOTE**
>
> The tumbling window trigger run *waits for the triggered pipeline run* to finish. Its run state reflects the state of the triggered pipeline run. For example, if a triggered pipeline run is cancelled, the corresponding tumbling window trigger run is marked cancelled. This is different from the "fire and forget" behavior of the schedule trigger, which is marked successful as long as a pipeline run started.

The following table provides a comparison of the tumbling window trigger and schedule trigger:

ITEM	TUMBLING WINDOW TRIGGER	SCHEDULE TRIGGER
Backfill scenarios	Supported. Pipeline runs can be scheduled for windows in the past.	Not supported. Pipeline runs can be executed only on time periods from the current time and the future.
Reliability	100% reliability. Pipeline runs can be scheduled for all windows from a specified start date without gaps.	Less reliable.

ITEM	TUMBLING WINDOW TRIGGER	SCHEDULE TRIGGER
Retry capability	Supported. Failed pipeline runs have a default retry policy of 0, or a policy that's specified by the user in the trigger definition. Automatically retries when the pipeline runs fail due to concurrency/server/throttling limits (that is, status codes 400: User Error, 429: Too many requests, and 500: Internal Server error).	Not supported.
Concurrency	Supported. Users can explicitly set concurrency limits for the trigger. Allows between 1 and 50 concurrent triggered pipeline runs.	Not supported.
System variables	Along with @trigger().scheduledTime and @trigger().startTime, it also supports the use of the **WindowStart** and **WindowEnd** system variables. Users can access `trigger().outputs.windowStartTime` and `trigger().outputs.windowEndTime` as trigger system variables in the trigger definition. The values are used as the window start time and window end time, respectively. For example, for a tumbling window trigger that runs every hour, for the window 1:00 AM to 2:00 AM, the definition is `trigger().outputs.windowStartTime = 2017-09-01T01:00:00Z` and `trigger().outputs.windowEndTime = 2017-09-01T02:00:00Z` .	Only supports default @trigger().scheduledTime and @trigger().startTime variables.
Pipeline-to-trigger relationship	Supports a one-to-one relationship. Only one pipeline can be triggered.	Supports many-to-many relationships. Multiple triggers can kick off a single pipeline. A single trigger can kick off multiple pipelines.

Event-based trigger

An event-based trigger runs pipelines in response to an event. There are two flavors of event-based triggers.

- *Storage event trigger* runs a pipeline against events happening in a Storage account, such as the arrival of a file, or the deletion of a file in Azure Blob Storage account.
- *Custom event trigger* processes and handles custom articles in Event Grid

For more information about event-based triggers, see Storage Event Trigger and Custom Event Trigger.

Next steps

See the following tutorials:

- Quickstart: Create a data factory by using the .NET SDK
- Create a schedule trigger

- Create a tumbling window trigger

Integration runtime in Azure Data Factory

8/19/2022 • 14 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

The Integration Runtime (IR) is the compute infrastructure used by Azure Data Factory and Azure Synapse pipelines to provide the following data integration capabilities across different network environments:

- **Data Flow**: Execute a Data Flow in a managed Azure compute environment.
- **Data movement**: Copy data across data stores in a public or private networks (for both on-premises or virtual private networks). The service provides support for built-in connectors, format conversion, column mapping, and performant and scalable data transfer.
- **Activity dispatch**: Dispatch and monitor transformation activities running on a variety of compute services such as Azure Databricks, Azure HDInsight, ML Studio (classic), Azure SQL Database, SQL Server, and more.
- **SSIS package execution**: Natively execute SQL Server Integration Services (SSIS) packages in a managed Azure compute environment.

In Data Factory and Synapse pipelines, an activity defines the action to be performed. A linked service defines a target data store or a compute service. An integration runtime provides the bridge between activities and linked services. It's referenced by the linked service or activity, and provides the compute environment where the activity is either run directly or dispatched. This allows the activity to be performed in the closest possible region to the target data store or compute service to maximize performance while also allowing flexibility to meet security and compliance requirements.

Integration runtimes can be created in the Azure Data Factory and Azure Synapse UI via the management hub directly, as well as from any activities, datasets, or data flows that reference them.

Integration runtime types

Data Factory offers three types of Integration Runtime (IR), and you should choose the type that best serves your data integration capabilities and network environment requirements. The three types of IR are:

- Azure
- Self-hosted
- Azure-SSIS

> **NOTE**
> Synapse pipelines currently only support Azure or self-hosted integration runtimes.

The following table describes the capabilities and network support for each of the integration runtime types:

IR TYPE	PUBLIC NETWORK SUPPORT	PRIVATE LINK SUPPORT
Azure	Data Flow Data movement Activity dispatch	Data Flow Data movement Activity dispatch
Self-hosted	Data movement Activity dispatch	Data movement Activity dispatch

IR TYPE	PUBLIC NETWORK SUPPORT	PRIVATE LINK SUPPORT
Azure-SSIS	SSIS package execution	SSIS package execution

> **NOTE**
> Outbound controls vary by service for Azure IR. In Synapse, workspaces have options to limit outbound traffic from the managed virtual network when utilizing Azure IR. In Data Factory, all ports are opened for outbound communications when utilizing Azure IR. Azure-SSIS IR can be integrated with your vNET to provide outbound communications controls.

Azure integration runtime

An Azure integration runtime can:

- Run Data Flows in Azure
- Run copy activities between cloud data stores
- Dispatch the following transform activities in a public network: Databricks Notebook/ Jar/ Python activity, HDInsight Hive activity, HDInsight Pig activity, HDInsight MapReduce activity, HDInsight Spark activity, HDInsight Streaming activity, ML Studio (classic) Batch Execution activity, ML Studio (classic) Update Resource activities, Stored Procedure activity, Data Lake Analytics U-SQL activity, .NET custom activity, Web activity, Lookup activity, and Get Metadata activity.

Azure IR network environment

Azure Integration Runtime supports connecting to data stores and computes services with public accessible endpoints. Enabling Managed Virtual Network, Azure Integration Runtime supports connecting to data stores using private link service in private network environment. In Synapse, workspaces have options to limit outbound traffic from the IR managed virtual network. In Data Factory, all ports are opened for outbound communications. The Azure-SSIS IR can be integrated with your vNET to provide outbound communications controls.

Azure IR compute resource and scaling

Azure integration runtime provides a fully managed, serverless compute in Azure. You don't have to worry about infrastructure provision, software installation, patching, or capacity scaling. In addition, you only pay for the duration of the actual utilization.

Azure integration runtime provides the native compute to move data between cloud data stores in a secure, reliable, and high-performance manner. You can set how many data integration units to use on the copy activity, and the compute size of the Azure IR is elastically scaled up accordingly without requiring you to explicitly adjust the size of the Azure Integration Runtime.

Activity dispatch is a lightweight operation to route the activity to the target compute service, so there isn't need to scale up the compute size for this scenario.

For information about creating and configuring an Azure IR, see How to create and configure Azure Integration Runtime.

> **NOTE**
> Azure Integration runtime has properties related to Data Flow runtime, which defines the underlying compute infrastructure that would be used to run the data flows.

Self-hosted integration runtime

A self-hosted IR is capable of:

- Running copy activity between a cloud data stores and a data store in private network.
- Dispatching the following transform activities against compute resources in on-premises or Azure Virtual Network: HDInsight Hive activity (BYOC-Bring Your Own Cluster), HDInsight Pig activity (BYOC), HDInsight MapReduce activity (BYOC), HDInsight Spark activity (BYOC), HDInsight Streaming activity (BYOC), ML Studio (classic) Batch Execution activity, ML Studio (classic) Update Resource activities, Stored Procedure activity, Data Lake Analytics U-SQL activity, Custom activity (runs on Azure Batch), Lookup activity, and Get Metadata activity.

> **NOTE**
> Use self-hosted integration runtime to support data stores that require bring-your-own driver, such as SAP Hana, MySQL, etc. For more information, see supported data stores.

> **NOTE**
> The Java Runtime Environment (JRE) is a dependency of the Self Hosted IR. Please make sure you have the JRE installed on the same host.

Self-hosted IR network environment

If you want to perform data integration securely in a private network environment that doesn't have a direct line-of-sight from the public cloud environment, you can install a self-hosted IR in your on-premises environment behind a firewall, or inside a virtual private network. The self-hosted integration runtime only makes outbound HTTP-based connections to the internet.

Self-hosted IR compute resource and scaling

Install a Self-hosted IR on an on-premises machine or a virtual machine inside a private network. Currently, the self-hosted IR is only supported on a Windows operating system.
For high availability and scalability, you can scale out the self-hosted IR by associating the logical instance with multiple on-premises machines in active-active mode. For more information, see the article on how to create and configure a self-hosted IR for details.

Azure-SSIS integration runtime

To lift and shift existing SSIS workload, you can create an Azure-SSIS IR to natively execute SSIS packages.

Azure-SSIS IR network environment

The Azure-SSIS IR can be provisioned in either public network or private network. On-premises data access is supported by joining Azure-SSIS IR to a virtual network that is connected to your on-premises network.

Azure-SSIS IR compute resource and scaling

The Azure-SSIS IR is a fully managed cluster of Azure VMs dedicated to run your SSIS packages. You can bring your own Azure SQL Database or SQL Managed Instance for the catalog of SSIS projects/packages (SSISDB). You can scale up the power of the compute by specifying node size and scale it out by specifying the number of nodes in the cluster. You can manage the cost of running your Azure-SSIS Integration Runtime by stopping and starting it as your requirements demand.

For more information, see How to create and configure the Azure-SSIS IR. Once created, you can deploy and manage your existing SSIS packages with little to no change using familiar tools such as SQL Server Data Tools (SSDT) and SQL Server Management Studio (SSMS), just like using SSIS on-premises.

For more information about the Azure-SSIS runtime, see the following articles:

- Tutorial: deploy SSIS packages to Azure. This article provides step-by-step instructions to create an Azure-SSIS IR and uses an Azure SQL Database to host the SSIS catalog.
- How to: Create an Azure-SSIS integration runtime. This article expands on the tutorial and provides instructions on using SQL Managed Instance and joining the IR to a virtual network.
- Monitor an Azure-SSIS IR. This article shows you how to retrieve information about an Azure-SSIS IR and provides descriptions of statuses in the returned information.
- Manage an Azure-SSIS IR. This article shows you how to stop, start, or remove an Azure-SSIS IR. It also shows you how to scale out your Azure-SSIS IR by adding more nodes to the IR.
- Join an Azure-SSIS IR to a virtual network. This article provides conceptual information about joining an Azure-SSIS IR to an Azure virtual network. It also provides steps to use the Azure portal to configure a virtual network and join an Azure-SSIS IR to it.

Integration runtime location

Relationship between factory location and IR location

When you create an instance of Data Factory or a Synapse Workspace, you need to specify its location. The metadata for the instance is stored here, and triggering of the pipeline is initiated from here. Metadata is only stored in the chosen region and will not be stored in other regions.

Meanwhile, a pipeline can access data stores and compute services in other Azure regions to move data between data stores or process data using compute services. This behavior is realized through the globally available IR to ensure data compliance, efficiency, and reduced network egress costs.

The IR Location defines the location of its back-end compute, and where the data movement, activity dispatching, and SSIS package execution are performed. The IR location can be different from the location of the Data Factory it belongs to.

Azure IR location

You can set the location region of an Azure IR, in which case the activity execution or dispatch will happen in the selected region.

The default is to auto-resolve the Azure IR in the public network. With this option:

- For copy activity, a best effort is made to automatically detect your sink data store's location, then use the IR in either the same region, if available, or the closest one in the same geography, otherwise; if the sink data store's region is not detectable, the IR in the instance's region is used instead.

  For example, a Data Factory or Synapse Workspace was created in East US,

  - When copying data to an Azure Blob in West US, if the blob is detected to be in the West US region, the copy activity is executed on the IR in West US; if region detection fails, the copy activity is executed on the IR in East US.
  - When copying data to Salesforce, for which the region is not detectable, the copy activity is executed on the IR in East US.

  > **TIP**
  > If you have strict data compliance requirements and need to ensure that data do not leave a certain geography, you can explicitly create an Azure IR in a certain region and point the Linked Service to this IR using the ConnectVia property. For example, if you want to copy data from a blob in UK South to an Azure Synapse workspace in UK South and want to ensure data does not leave the UK, create an Azure IR in UK South and link both Linked Services to this IR.

- For Lookup/GetMetadata/Delete activity execution (Pipeline activities), transformation activity dispatching (External activities), and authoring operations (test connection, browse folder list and table list, and

preview data), the IR in the same region as the Data Factory or Synapse Workspace is used.

- For Data Flow, the IR in the Data Factory or Synapse Workspace region is used.

  > **TIP**
  > A best practice is to ensure data flows run in the same region as your corresponding data stores when possible. You can either achieve this with auto-resolve for the Azure IR (if the data store location is the same as the Data Factory or Synapse Workspace location), or by creating a new Azure IR instance in the same region as your data stores and then executing the data flows on it.

If you enable Managed Virtual Network with auto-resolve for the Azure IR, the IR in the Data Factory or Synapse Workspace region is used.

You can monitor which IR location takes effect during activity execution in pipeline activity monitoring view in the Data Factory Studio or Synapse Studio, or in the activity monitoring payload.

Self-hosted IR location

The self-hosted IR is logically registered to the Data Factory or Synapse Workspace and the compute used to support its functionalities is provided by you. Therefore there is no explicit location property for self-hosted IR.

When used to perform data movement, the self-hosted IR extracts data from the source and writes into the destination.

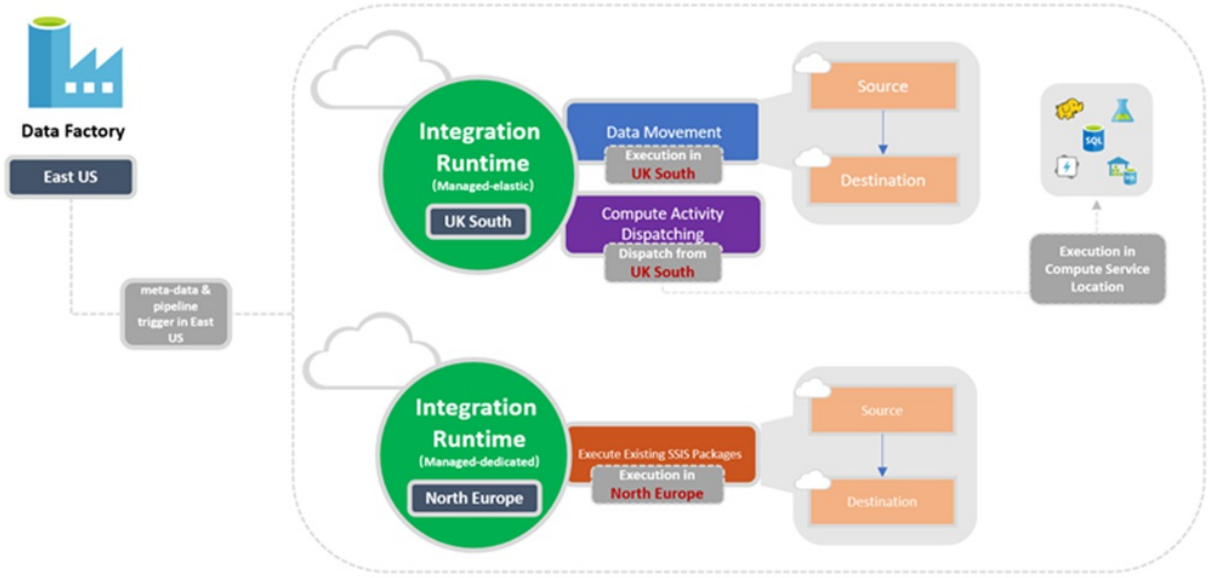
Azure-SSIS IR location

> **NOTE**
> Azure-SSIS integration runtimes are not currently supported in Synapse pipelines.

Selecting the right location for your Azure-SSIS IR is essential to achieve high performance in your extract-transform-load (ETL) workflows.

- The location of your Azure-SSIS IR does not need to be the same as the location of your Data Factory, but it should be the same as the location of your own Azure SQL Database or SQL Managed Instance where SSISDB is located. This way, your Azure-SSIS Integration Runtime can easily access the SSISDB without incurring excessive traffic between different locations.
- If you do not have an existing SQL Database or SQL Managed Instance, but you have on-premises data sources/destinations, you should create a new Azure SQL Database or SQL Managed Instance in the same location of a virtual network connected to your on-premises network. This way, you can create your Azure-SSIS IR using the new Azure SQL Database or SQL Managed Instance, and join that virtual network. Everything will be in the same location, minimizing data movement and associated costs, while maximizing performance.
- If the location of your existing Azure SQL Database or SQL Managed Instance is not the same as the location of a virtual network connected to your on-premises network, first create your Azure-SSIS IR using an existing Azure SQL Database or SQL Managed Instance and join another virtual network in the same location. Then configure a virtual network to virtual network connection between the different locations.

The following diagram shows the location settings for Data Factory and its integration runtimes:



Determining which IR to use

If an activity associates with more than one type of integration runtime, it will resolve to one of them. The self-hosted integration runtime takes precedence over the Azure integration runtime in Azure Data Factory or Synapse Workspace instances using a managed virtual network. And the latter takes precedence over the global Azure integration runtime.

For example, one copy activity is used to copy data from source to sink. The global Azure integration runtime is associated with the linked service to source and an Azure integration runtime in an Azure Data Factory managed virtual network associates with the linked service for sink, then the result is that both source and sink linked services use the Azure integration runtime in the Azure Data Factory managed virtual network. But if a self-hosted integration runtime associates the linked service for source, then both source and sink linked service use the self-hosted integration runtime.

Copy activity

The Copy activity requires both source and sink linked services to define the direction of data flow. The following logic is used to determine which integration runtime instance is used to perform the copy:

- **Copying between two cloud data sources**: if both source and sink linked services are using Azure IR, the regional Azure IR is used if it was specified, or the location of the Azure IR is automatically determined if the auto-resolve IR (default) option was chosen as described in the Integration runtime location section.
- **Copying between a cloud data source and a data source in a private network**: if either the source or sink linked service points to a self-hosted IR, the copy activity is executed on the self-hosted IR.
- **Copying between two data sources in a private network**: both the source and sink linked service must point to the same instance of the integration runtime, and that IR is used to execute the copy activity.

Lookup and GetMetadata activity

The Lookup and GetMetadata activity is executed on the integration runtime associated to the data store linked service.

External transformation activity

Each external transformation activity that utilizes an external compute engine has a target compute linked service, which points to an integration runtime. This IR instance determines the location from where that external hand-coded transformation activity is dispatched.

Data Flow activity

Data Flow activities are executed on their associated Azure integration runtime. The Spark compute utilized by

Data Flows are determined by the data flow properties in your Azure IR, and are fully managed by the service.

Integration Runtime in CI/CD

Integration runtimes don't change often and are similar across all stages in your CI/CD. Data Factory requires you to have the same name and type of integration runtime across all stages of CI/CD. If you want to share integration runtimes across all stages, consider using a dedicated factory just to contain the shared integration runtimes. You can then use this shared factory in all of your environments as a linked integration runtime type.

Next steps

See the following articles:

- Create Azure integration runtime
- Create self-hosted integration runtime
- Create an Azure-SSIS integration runtime. This article expands on the tutorial and provides instructions on using SQL Managed Instance and joining the IR to a virtual network.

Data integration in Azure Synapse Analytics versus Azure Data Factory

8/19/2022 • 2 minutes to read • Edit Online

In Azure Synapse Analytics, the data integration capabilities such as Synapse pipelines and data flows are based upon those of Azure Data Factory. For more information, see what is Azure Data Factory.

Available features in ADF & Azure Synapse Analytics

Check below table for features availability:

CATEGORY	FEATURE	AZURE DATA FACTORY	AZURE SYNAPSE ANALYTICS
Integration Runtime	Using SSIS and SSIS Integration Runtime	✓	✓ *Public preview*
	Support for Cross-region Integration Runtime (Data Flows)	✓	✗
	Integration Runtime Sharing	✓ *Can be shared across different data factories*	✗
Pipelines Activities	SSIS Package Activity	✓	✓ *Public preview*
	Support for Power Query Activity	✓	✗
	Support for global parameters	✓	✗
Template Gallery and Knowledge center	Solution Templates	✓ *Azure Data Factory Template Gallery*	✓ *Synapse Workspace Knowledge center*
GIT Repository Integration	GIT Integration	✓	✓
Monitoring	Monitoring of Spark Jobs for Data Flow	✗	✓ *Leverage the Synapse Spark pools*

Next steps

Get started with data integration in your Synapse workspace by learning how to ingest data into a Azure Data Lake Storage gen2 account.

Data flows in Azure Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

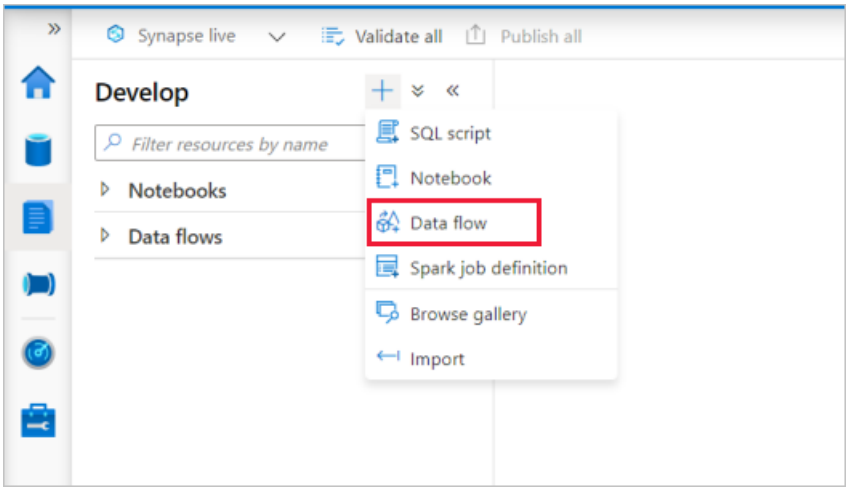
What are data flows?

Data flows are visually designed data transformations in Azure Synapse Analytics. Data flows allow data engineers to develop data transformation logic without writing code. The resulting data flows are executed as activities within Azure Synapse Analytics pipelines that use scaled-out Apache Spark clusters. Data flow activities can be operationalized using existing Azure Synapse Analytics scheduling, control, flow, and monitoring capabilities.

Data flows provide an entirely visual experience with no coding required. Your data flows run on Synapse-managed execution clusters for scaled-out data processing. Azure Synapse Analytics handles all the code translation, path optimization, and execution of your data flow jobs.

Getting started

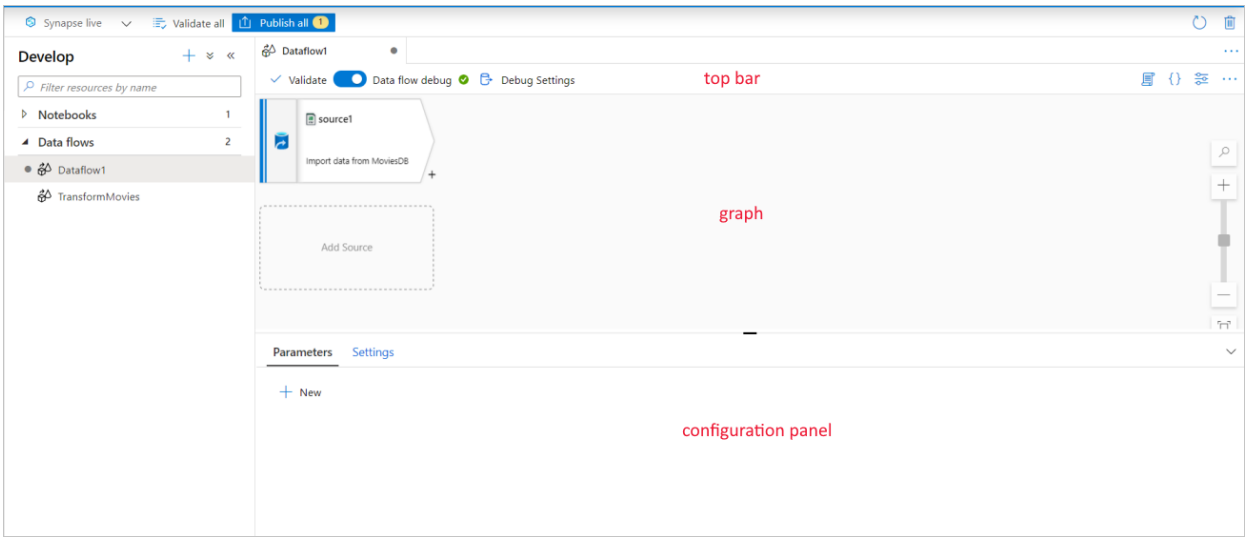
Data flows are created from the Develop pane in Synapse studio. To create a data flow, select the plus sign next to **Develop**, and then select **Data Flow**.



This action takes you to the data flow canvas, where you can create your transformation logic. Select **Add source** to start configuring your source transformation. For more information, see Source transformation.

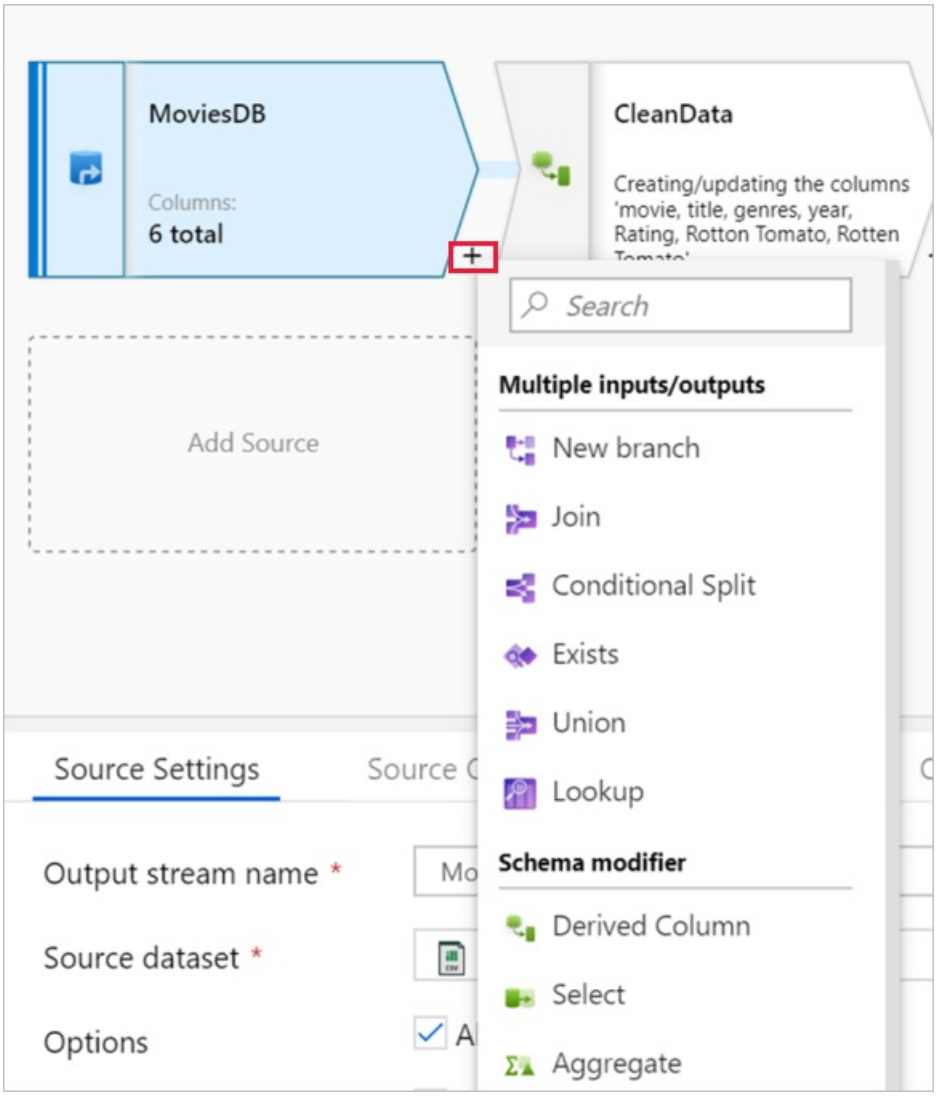
Authoring data flows

Data flow has a unique authoring canvas designed to make building transformation logic easy. The data flow canvas is separated into three parts: the top bar, the graph, and the configuration panel.



Graph

The graph displays the transformation stream. It shows the lineage of source data as it flows into one or more sinks. To add a new source, select **Add source**. To add a new transformation, select the plus sign on the lower right of an existing transformation. Learn more on how to manage the data flow graph.



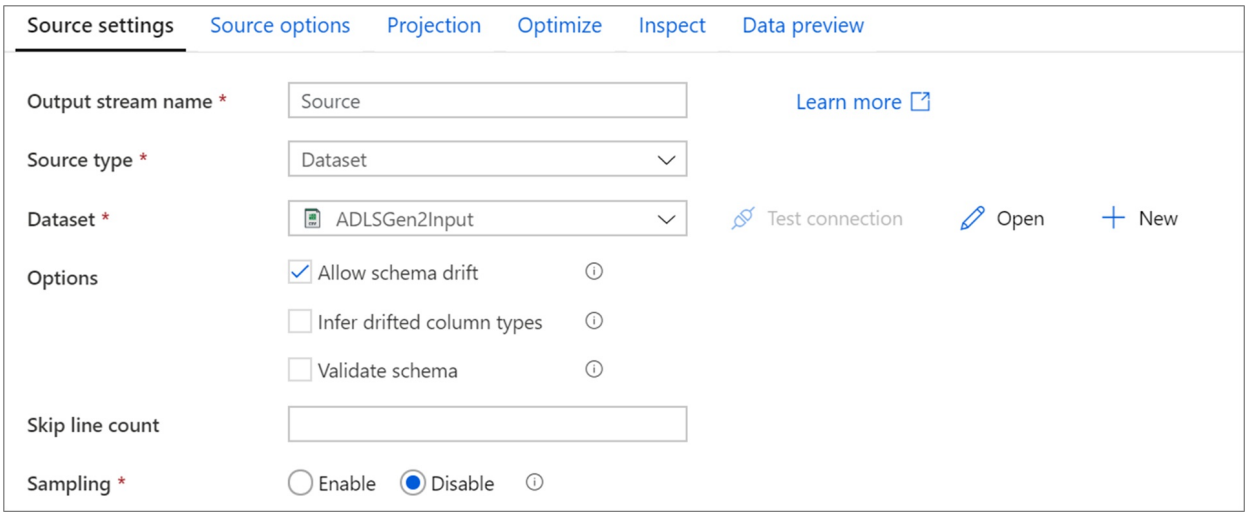
Configuration panel

The configuration panel shows the settings specific to the currently selected transformation. If no transformation is selected, it shows the data flow. In the overall data flow configuration, you can add parameters via the **Parameters** tab. For more information, see Data flow parameters.

Each transformation contains at least four configuration tabs.

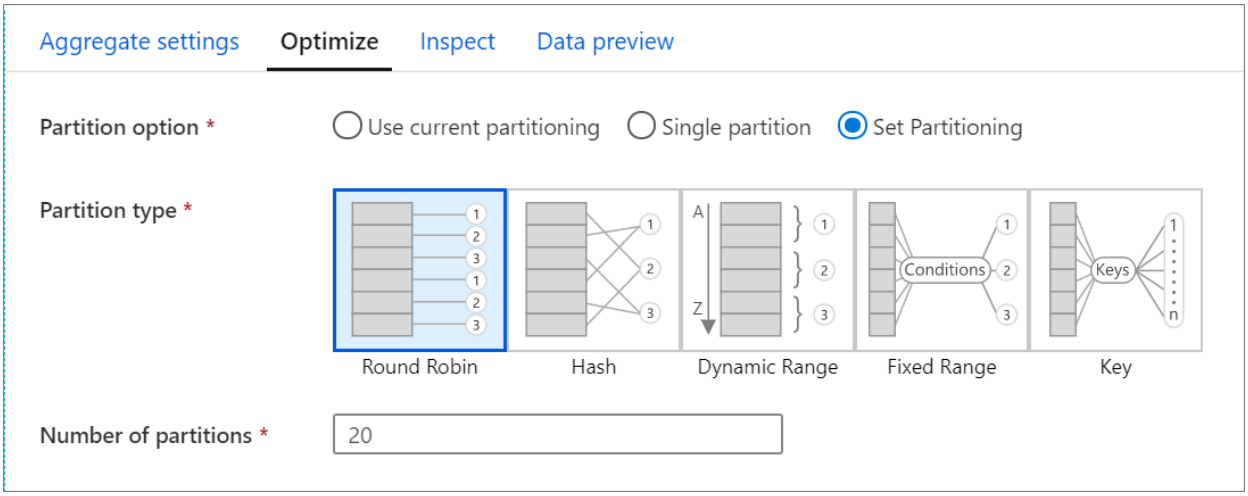
Transformation settings

The first tab in each transformation's configuration pane contains the settings specific to that transformation. For more information, see that transformation's documentation page.



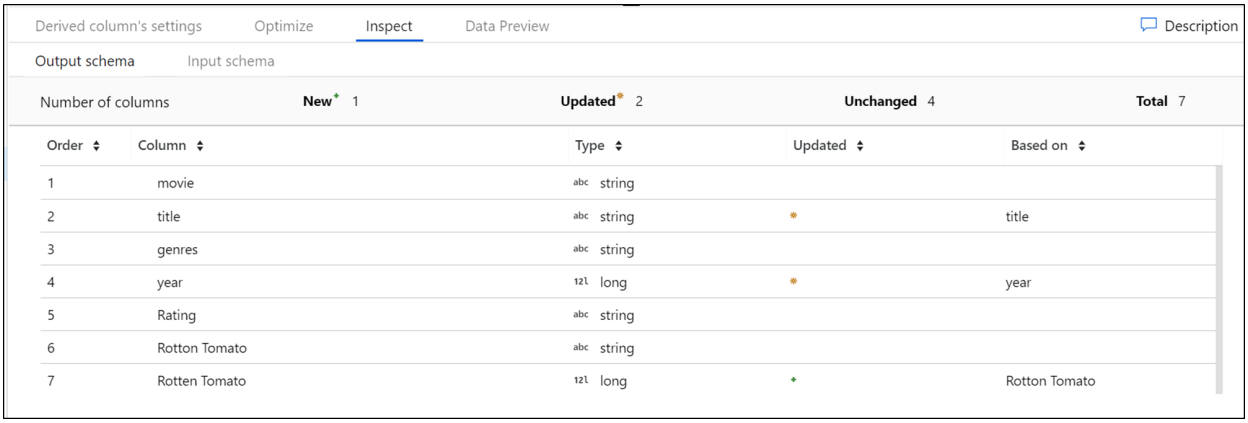
Optimize

The **Optimize** tab contains settings to configure partitioning schemes. To learn more about how to optimize your data flows, see the mapping data flow performance guide.



Inspect

The **Inspect** tab provides a view into the metadata of the data stream that you're transforming. You can see column counts, the columns changed, the columns added, data types, the column order, and column references. **Inspect** is a read-only view of your metadata. You don't need to have debug mode enabled to see metadata in the **Inspect** pane.



As you change the shape of your data through transformations, you'll see the metadata changes flow in the **Inspect** pane. If there isn't a defined schema in your source transformation, then metadata won't be visible in the **Inspect** pane. Lack of metadata is common in schema drift scenarios.

Data preview

If debug mode is on, the **Data Preview** tab gives you an interactive snapshot of the data at each transform. For more information, see Data preview in debug mode.

Top bar

The top bar contains actions that affect the whole data flow, like validation and debug settings. You can view the underlying JSON code and data flow script of your transformation logic as well.

Available transformations

View the mapping data flow transformation overview to get a list of available transformations.

Data flow activity

Data flows are operationalized within Azure Synapse Analytics pipelines using the data flow activity. All a user has to do is specify which integration runtime to use and pass in parameter values. For more information, learn about the Azure integration runtime.

Debug mode

Debug mode allows you to interactively see the results of each transformation step while you build and debug your data flows. The debug session can be used both in when building your data flow logic and running pipeline debug runs with data flow activities. To learn more, see the debug mode documentation.

Monitoring data flows

Data flow integrates with existing Azure Synapse Analytics monitoring capabilities. To learn how to understand data flow monitoring output, see monitoring mapping data flows.

The Azure Synapse Analytics team has created a performance tuning guide to help you optimize the execution time of your data flows after building your business logic.

Next steps

- Learn how to create a source transformation.
- Learn how to build your data flows in debug mode.

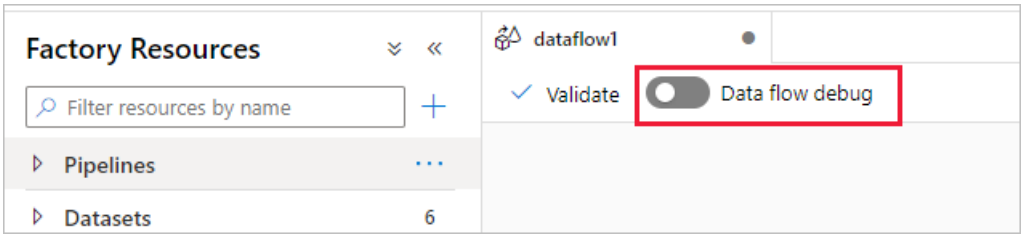
Mapping data flow Debug Mode

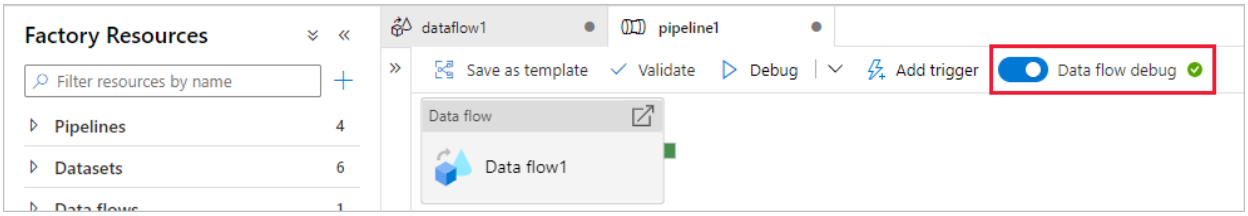
8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

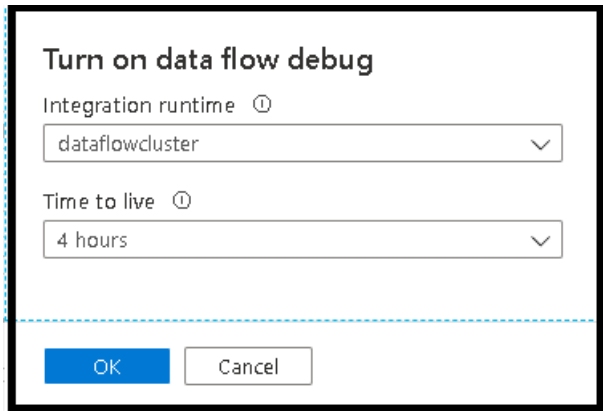
Overview

Azure Data Factory and Synapse Analytics mapping data flow's debug mode allows you to interactively watch the data shape transform while you build and debug your data flows. The debug session can be used both in Data Flow design sessions as well as during pipeline debug execution of data flows. To turn on debug mode, use the **Data Flow Debug** button in the top bar of data flow canvas or pipeline canvas when you have data flow activities.





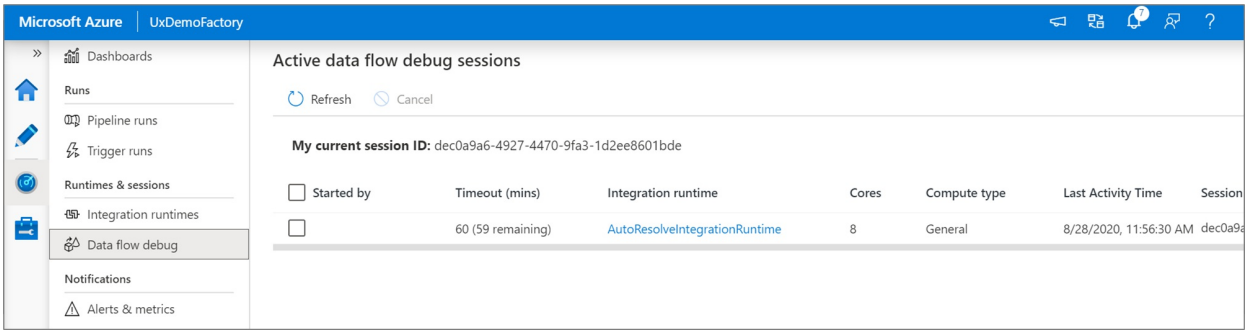
Once you turn on the slider, you will be prompted to select which integration runtime configuration you wish to use. If AutoResolveIntegrationRuntime is chosen, a cluster with eight cores of general compute with a default 60-minute time to live will be spun up. If you'd like to allow for more idle team before your session times out, you can choose a higher TTL setting. For more information on data flow integration runtimes, see Integration Runtime performance.



When Debug mode is on, you'll interactively build your data flow with an active Spark cluster. The session will close once you turn debug off. You should be aware of the hourly charges incurred by Data Factory during the time that you have the debug session turned on.

In most cases, it's a good practice to build your Data Flows in debug mode so that you can validate your business logic and view your data transformations before publishing your work. Use the "Debug" button on the pipeline panel to test your data flow in a pipeline.

- Azure Data Factory
- Synapse Analytics



> **NOTE**
>
> Every debug session that a user starts from their browser UI is a new session with its own Spark cluster. You can use the monitoring view for debug sessions above to view and manage debug sessions. You are charged for every hour that each debug session is executing including the TTL time.

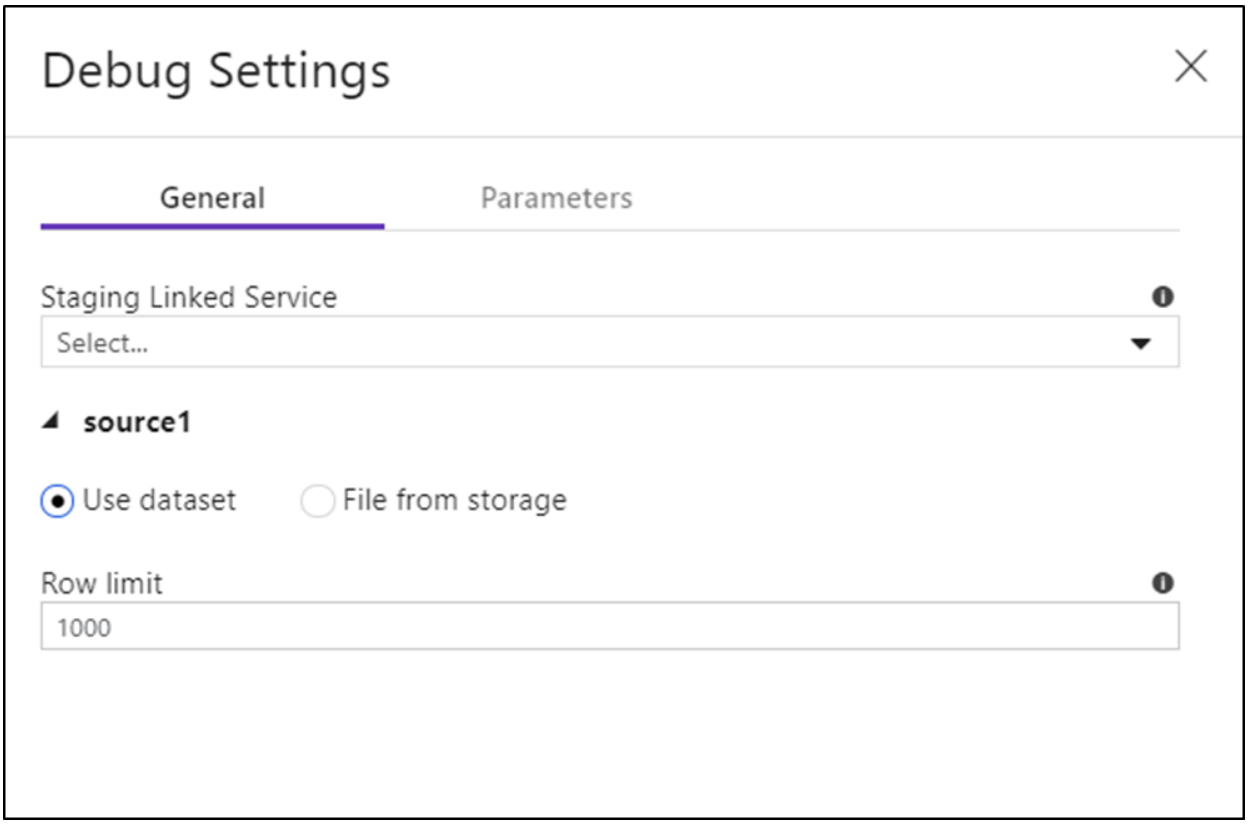
Cluster status

The cluster status indicator at the top of the design surface turns green when the cluster is ready for debug. If your cluster is already warm, then the green indicator will appear almost instantly. If your cluster wasn't already running when you entered debug mode, then the Spark cluster will perform a cold boot. The indicator will spin until the environment is ready for interactive debugging.

When you are finished with your debugging, turn the Debug switch off so that your Spark cluster can terminate and you'll no longer be billed for debug activity.

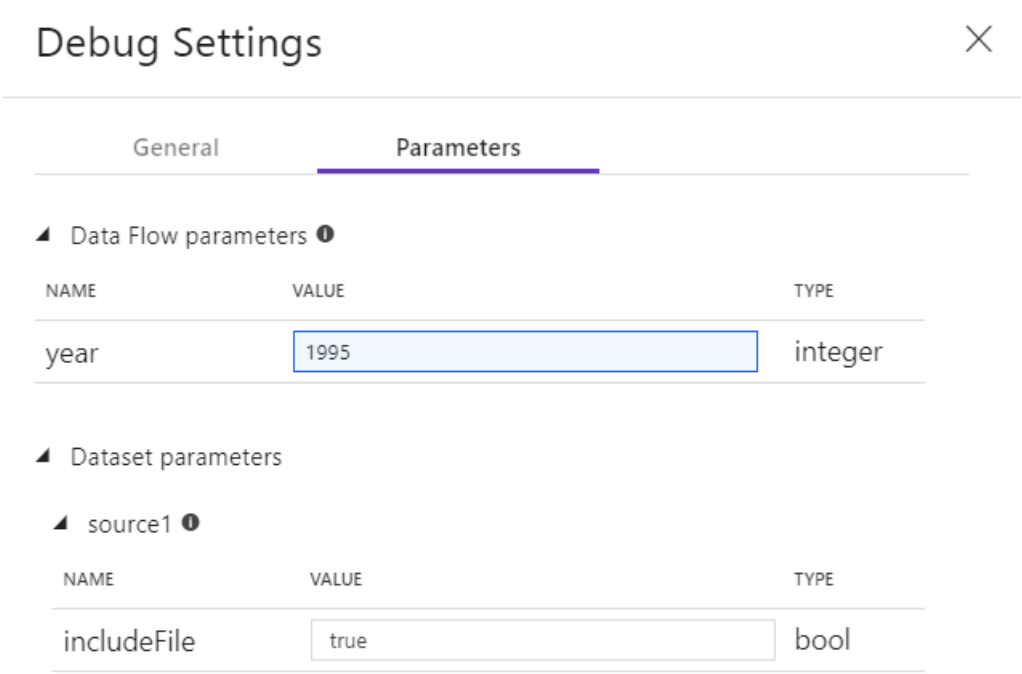
Debug settings

Once you turn on debug mode, you can edit how a data flow previews data. Debug settings can be edited by clicking "Debug Settings" on the Data Flow canvas toolbar. You can select the row limit or file source to use for each of your Source transformations here. The row limits in this setting are only for the current debug session. You can also select the staging linked service to be used for an Azure Synapse Analytics source.



If you have parameters in your Data Flow or any of its referenced datasets, you can specify what values to use during debugging by selecting the **Parameters** tab.

Use the sampling settings here to point to sample files or sample tables of data so that you do not have to change your source datasets. By using a sample file or table here, you can maintain the same logic and property settings in your data flow while testing against a subset of data.

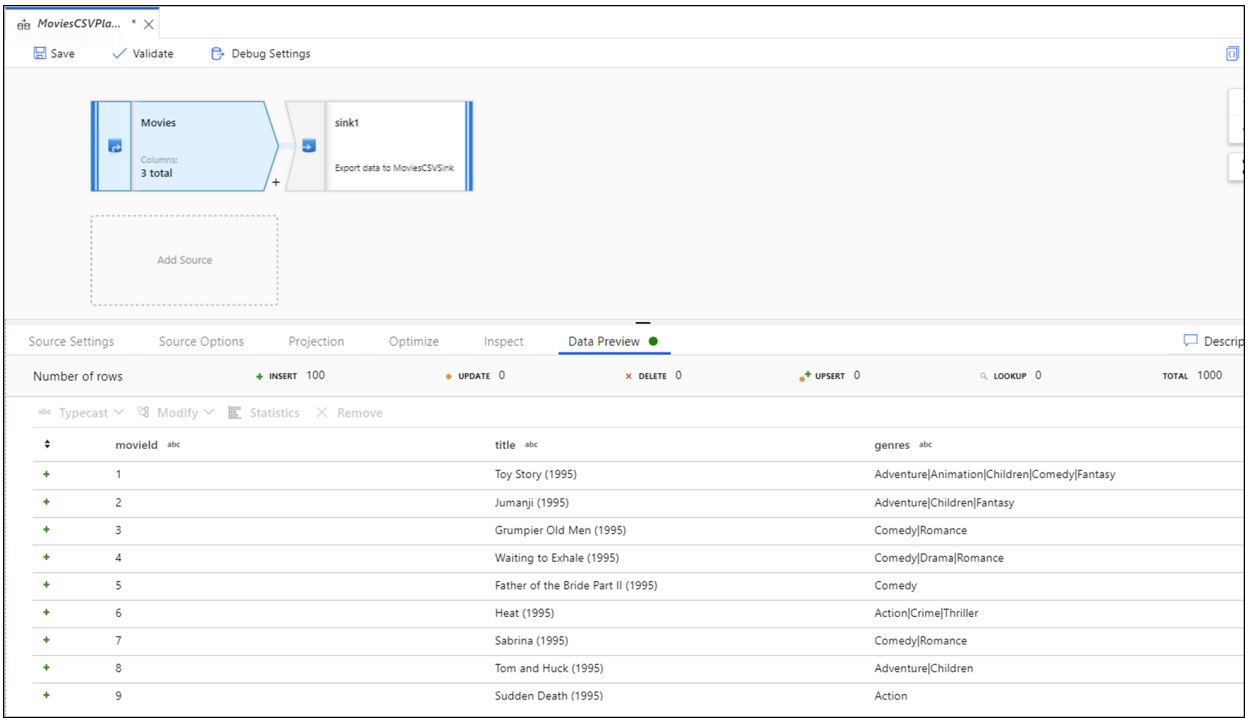


The default IR used for debug mode in data flows is a small 4-core single worker node with a 4-core single driver node. This works fine with smaller samples of data when testing your data flow logic. If you expand the row limits in your debug settings during data preview or set a higher number of sampled rows in your source during pipeline debug, then you may wish to consider setting a larger compute environment in a new Azure

Integration Runtime. Then you can restart your debug session using the larger compute environment.

Data preview

With debug on, the Data Preview tab will light-up on the bottom panel. Without debug mode on, Data Flow will show you only the current metadata in and out of each of your transformations in the Inspect tab. The data preview will only query the number of rows that you have set as your limit in your debug settings. Click **Refresh** to update the data preview based on your current transformations. If your source data has changed, then click the Refresh > Refetch from source.



> **NOTE**
> File sources only limit the rows that you see, not the rows being read. For very large datasets, it is recommended that you take a small portion of that file and use it for your testing. You can select a temporary file in Debug Settings for each source that is a file dataset type.

When running in Debug Mode in Data Flow, your data will not be written to the Sink transform. A Debug session is intended to serve as a test harness for your transformations. Sinks are not required during debug and are ignored in your data flow. If you wish to test writing the data in your Sink, execute the Data Flow from a pipeline and use the Debug execution from a pipeline.

Data Preview is a snapshot of your transformed data using row limits and data sampling from data frames in Spark memory. Therefore, the sink drivers are not utilized or tested in this scenario.

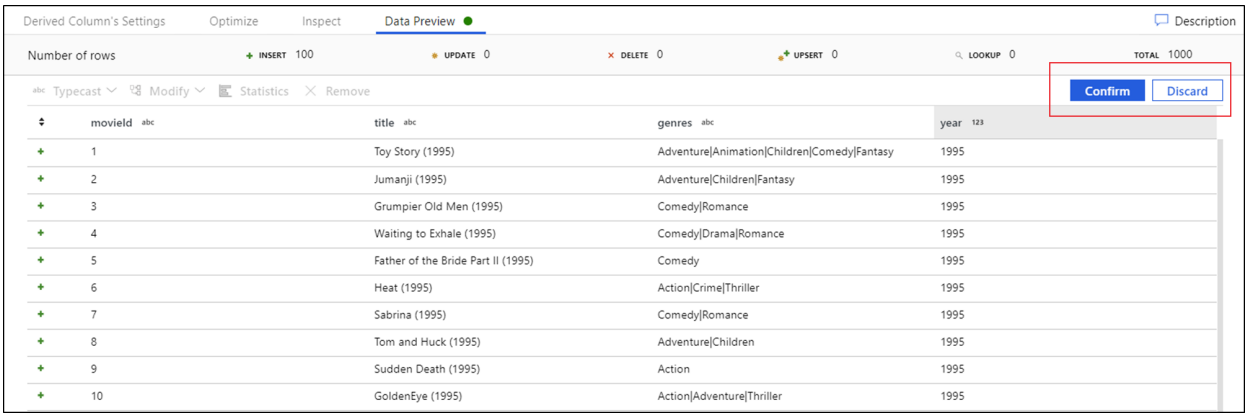
Testing join conditions

When unit testing Joins, Exists, or Lookup transformations, make sure that you use a small set of known data for your test. You can use the Debug Settings option above to set a temporary file to use for your testing. This is needed because when limiting or sampling rows from a large dataset, you cannot predict which rows and which keys will be read into the flow for testing. The result is non-deterministic, meaning that your join conditions may fail.

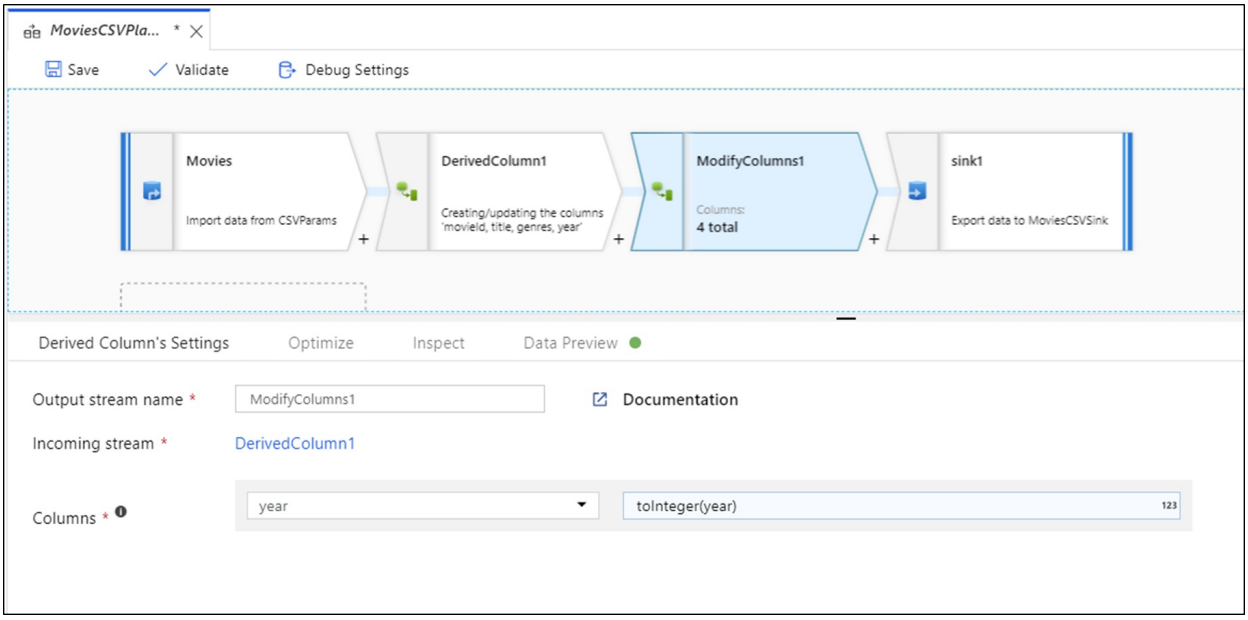
Quick actions

Once you see the data preview, you can generate a quick transformation to typecast, remove, or do a modification on a column. Click on the column header and then select one of the options from the data preview toolbar.

Once you select a modification, the data preview will immediately refresh. Click **Confirm** in the top-right corner to generate a new transformation.



Typecast and **Modify** will generate a Derived Column transformation and **Remove** will generate a Select transformation.

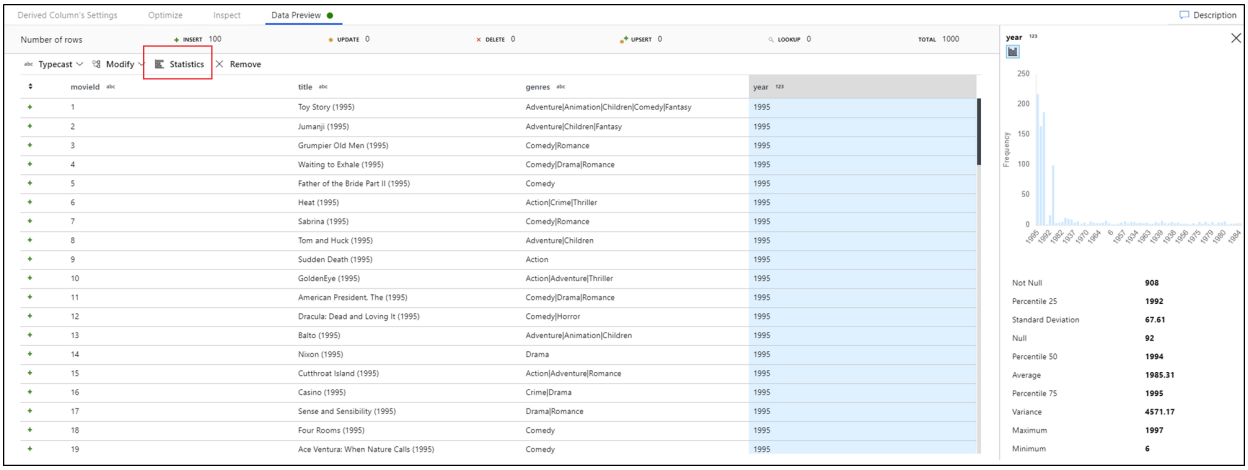


> **NOTE**
>
> If you edit your Data Flow, you need to re-fetch the data preview before adding a quick transformation.

Data profiling

Selecting a column in your data preview tab and clicking **Statistics** in the data preview toolbar will pop up a chart on the far-right of your data grid with detailed statistics about each field. The service will make a determination based upon the data sampling of which type of chart to display. High-cardinality fields will default

to NULL/NOT NULL charts while categorical and numeric data that has low cardinality will display bar charts showing data value frequency. You'll also see max/len length of string fields, min/max values in numeric fields, standard dev, percentiles, counts, and average.



Next steps

- Once you're finished building and debugging your data flow, execute it from a pipeline.
- When testing your pipeline with a data flow, use the pipeline Debug run execution option.

Schema drift in mapping data flow

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Schema drift is the case where your sources often change metadata. Fields, columns, and, types can be added, removed, or changed on the fly. Without handling for schema drift, your data flow becomes vulnerable to upstream data source changes. Typical ETL patterns fail when incoming columns and fields change because they tend to be tied to those source names.

To protect against schema drift, it's important to have the facilities in a data flow tool to allow you, as a Data Engineer, to:

- Define sources that have mutable field names, data types, values, and sizes
- Define transformation parameters that can work with data patterns instead of hard-coded fields and values
- Define expressions that understand patterns to match incoming fields, instead of using named fields

Azure Data Factory natively supports flexible schemas that change from execution to execution so that you can build generic data transformation logic without the need to recompile your data flows.

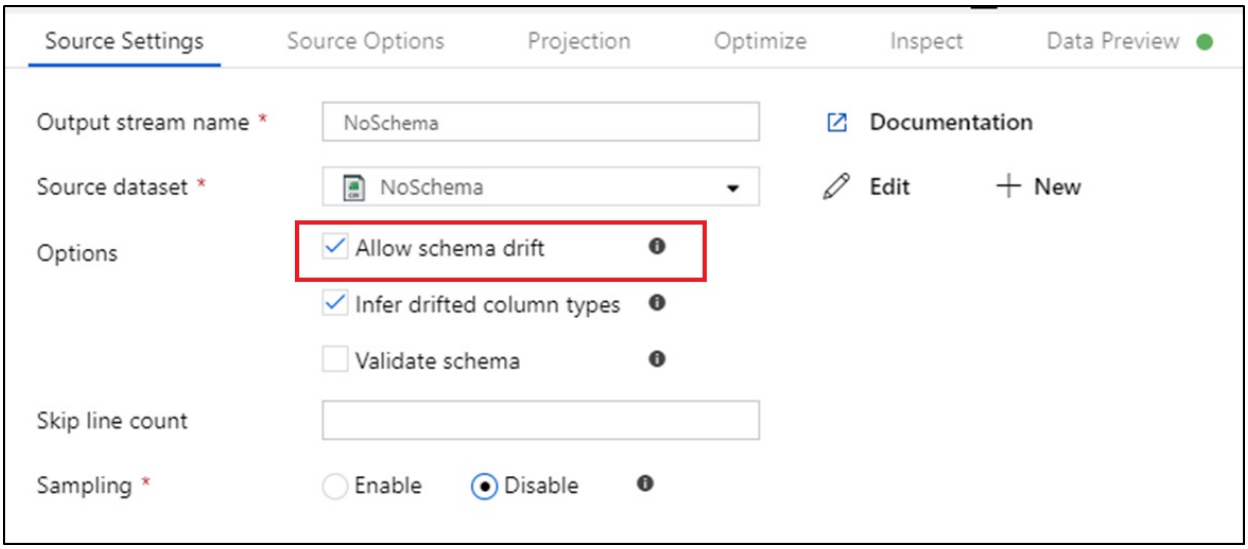
You need to make an architectural decision in your data flow to accept schema drift throughout your flow. When you do this, you can protect against schema changes from the sources. However, you'll lose early-binding of your columns and types throughout your data flow. Azure Data Factory treats schema drift flows as late-binding flows, so when you build your transformations, the drifted column names won't be available to you in the schema views throughout the flow.

This video provides an introduction to some of the complex solutions that you can build easily in Azure Data Factory or Synapse Analytics pipelines with data flow's **schema drift** feature. In this example, we build reusable patterns based on flexible database schemas:

Schema drift in source

Columns coming into your data flow from your source definition are defined as "drifted" when they are not present in your source projection. You can view your source projection from the projection tab in the source transformation. When you select a dataset for your source, the service will automatically take the schema from the dataset and create a projection from that dataset schema definition.

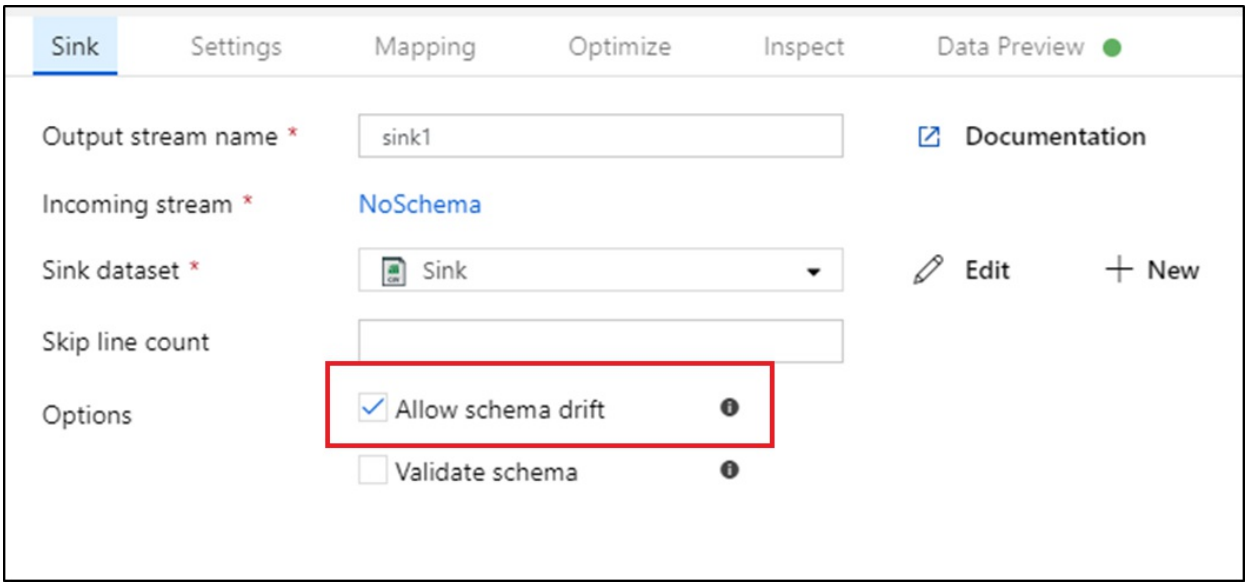
In a source transformation, schema drift is defined as reading columns that aren't defined your dataset schema. To enable schema drift, check **Allow schema drift** in your source transformation.



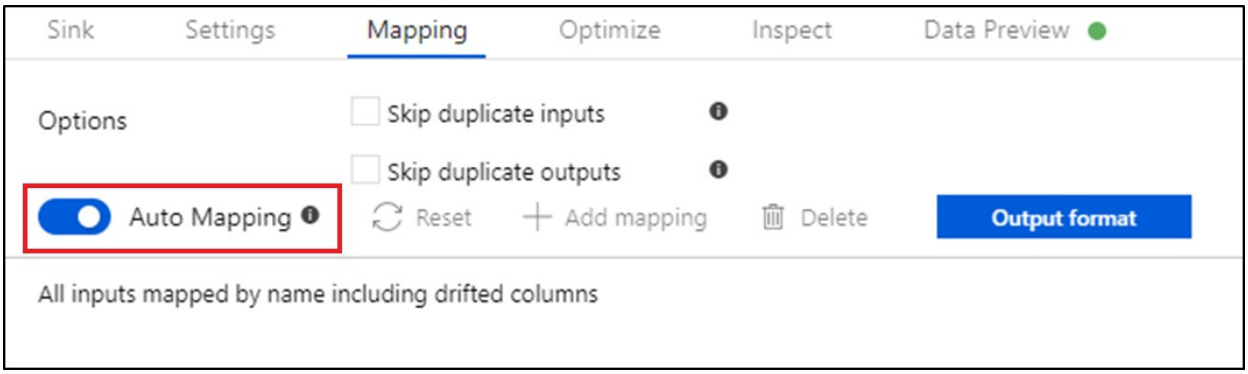
When schema drift is enabled, all incoming fields are read from your source during execution and passed through the entire flow to the Sink. By default, all newly detected columns, known as *drifted columns*, arrive as a string data type. If you wish for your data flow to automatically infer data types of drifted columns, check **Infer drifted column types** in your source settings.

Schema drift in sink

In a sink transformation, schema drift is when you write additional columns on top of what is defined in the sink data schema. To enable schema drift, check **Allow schema drift** in your sink transformation.



If schema drift is enabled, make sure the **Auto-mapping** slider in the Mapping tab is turned on. With this slider on, all incoming columns are written to your destination. Otherwise you must use rule-based mapping to write drifted columns.



Transforming drifted columns

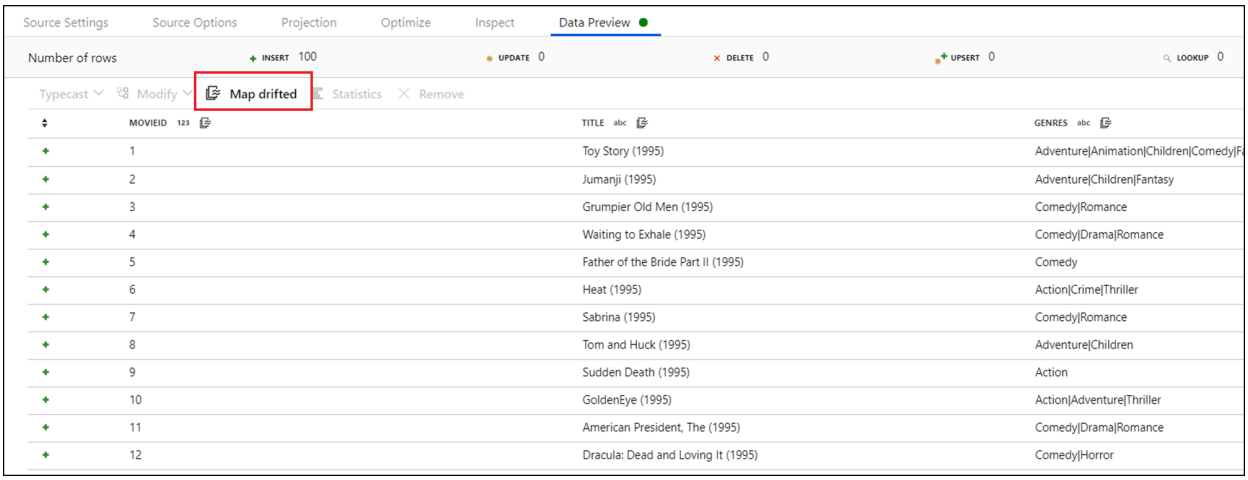
When your data flow has drifted columns, you can access them in your transformations with the following methods:

- Use the `byPosition` and `byName` expressions to explicitly reference a column by name or position number.
- Add a column pattern in a Derived Column or Aggregate transformation to match on any combination of name, stream, position, origin, or type
- Add rule-based mapping in a Select or Sink transformation to match drifted columns to columns aliases via a pattern

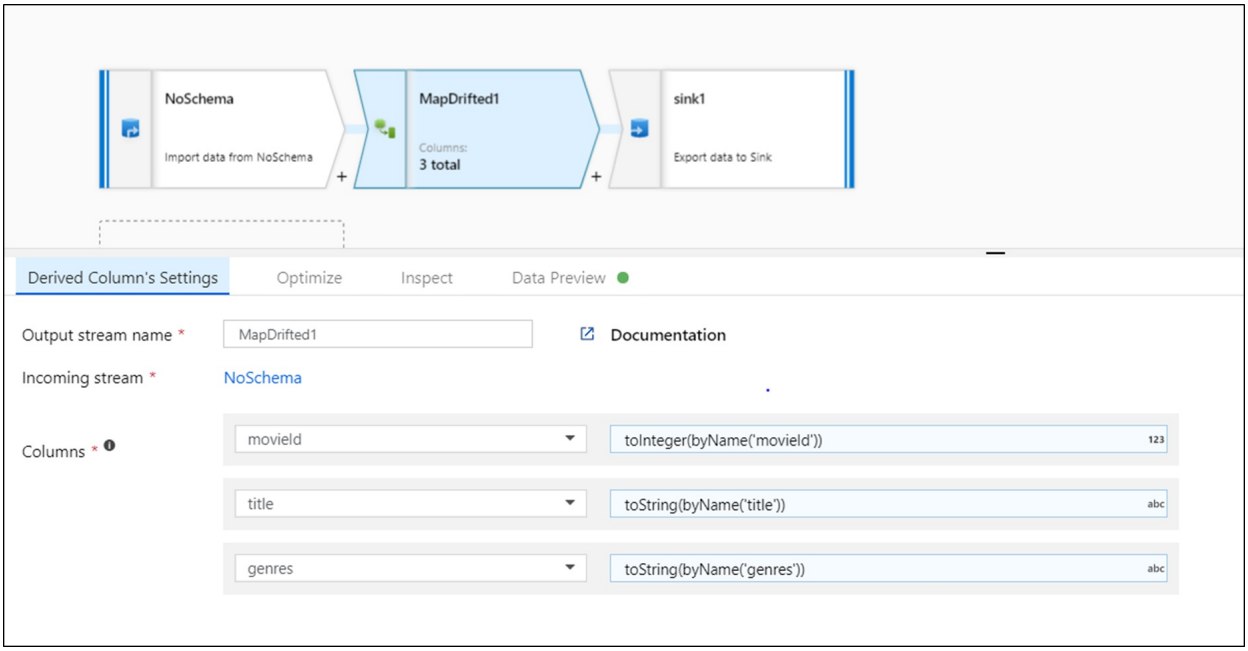
For more information on how to implement column patterns, see Column patterns in mapping data flow.

Map drifted columns quick action

To explicitly reference drifted columns, you can quickly generate mappings for these columns via a data preview quick action. Once debug mode is on, go to the Data Preview tab and click **Refresh** to fetch a data preview. If data factory detects that drifted columns exist, you can click **Map Drifted** and generate a derived column that allows you to reference all drifted columns in schema views downstream.



In the generated Derived Column transformation, each drifted column is mapped to its detected name and data type. In the above data preview, the column 'movieId' is detected as an integer. After **Map Drifted** is clicked, movieId is defined in the Derived Column as `toInteger(byName('movieId'))` and included in schema views in downstream transformations.



Next steps

In the Data Flow Expression Language, you'll find additional facilities for column patterns and schema drift including "byName" and "byPosition".

Using column patterns in mapping data flow

8/19/2022 • 4 minutes to read • Edit Online

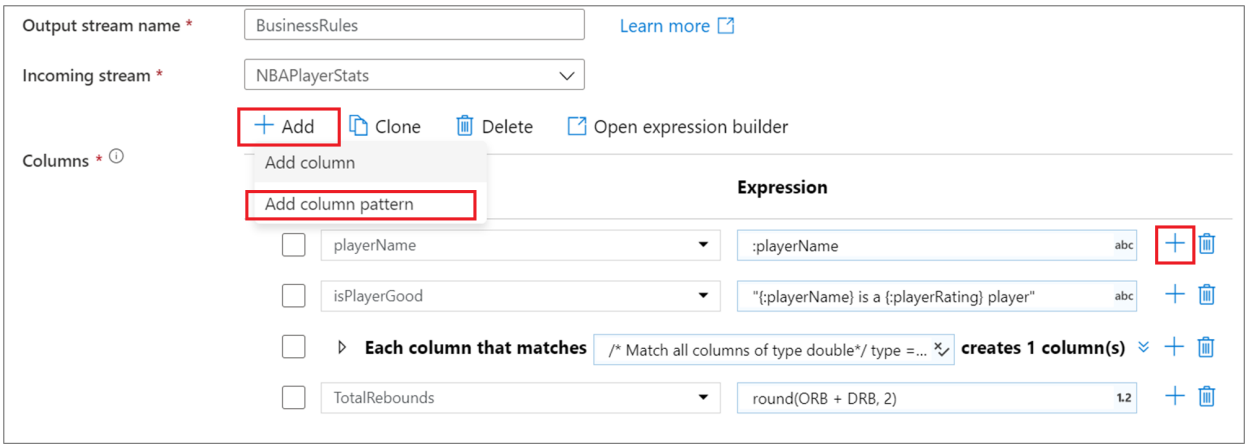
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Several mapping data flow transformations allow you to reference template columns based on patterns instead of hard-coded column names. This matching is known as *column patterns*. You can define patterns to match columns based on name, data type, stream, origin, or position instead of requiring exact field names. There are two scenarios where column patterns are useful:

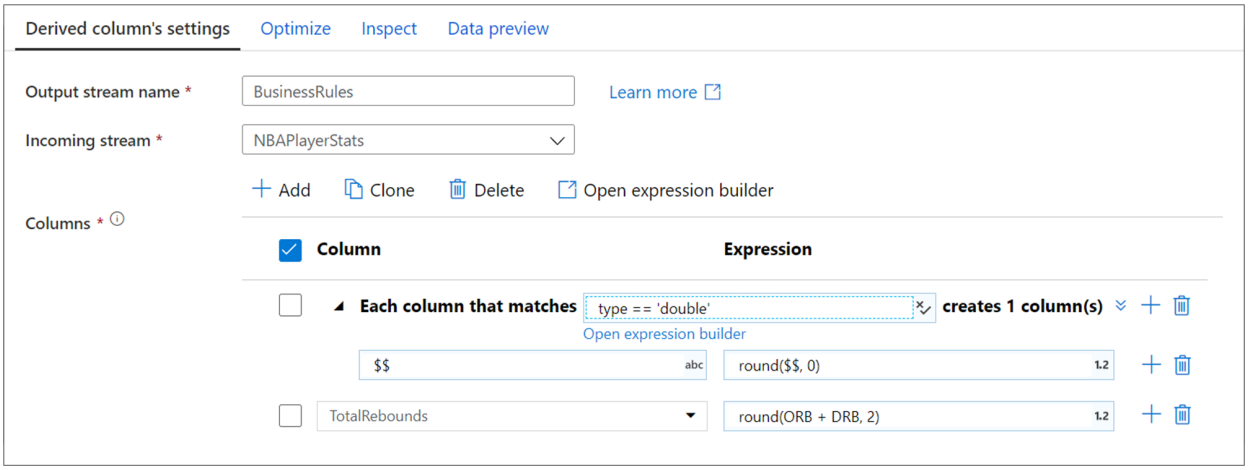
- If incoming source fields change often such as the case of changing columns in text files or NoSQL databases. This scenario is known as schema drift.
- If you wish to do a common operation on a large group of columns. For example, wanting to cast every column that has 'total' in its column name into a double.

Column patterns in derived column and aggregate

To add a column pattern in a derived column, aggregate, or window transformation, click on **Add** above the column list or the plus icon next to an existing derived column. Choose **Add column pattern**.



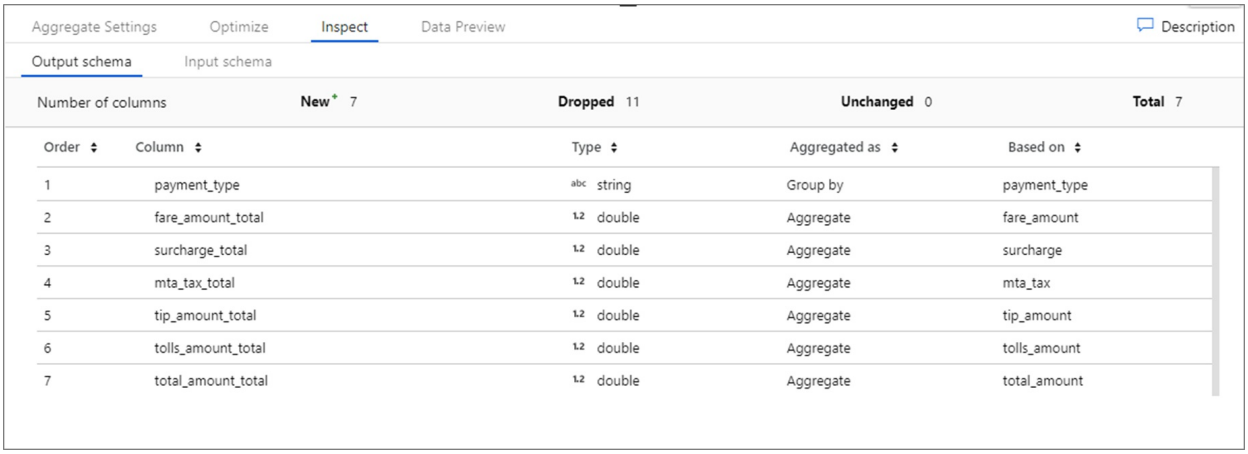
Use the expression builder to enter the match condition. Create a boolean expression that matches columns based on the `name`, `type`, `stream`, `origin`, and `position` of the column. The pattern will affect any column, drifted or defined, where the condition returns true.



The above column pattern matches every column of type double and creates one derived column per match. By

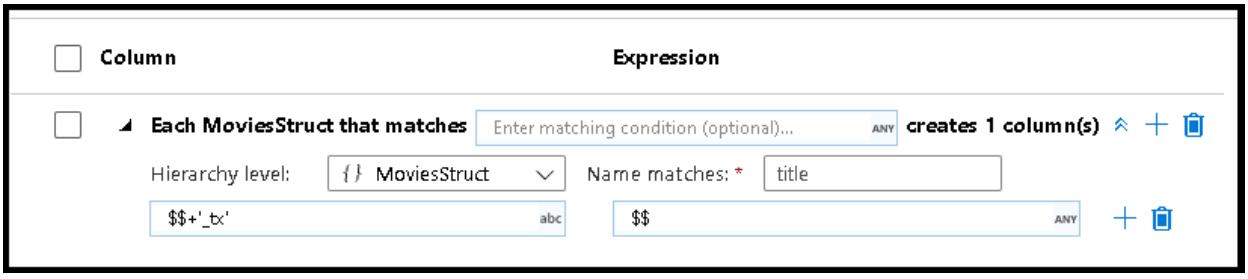
stating `$$` as the column name field, each matched column is updated with the same name. The value of the each column is the existing value rounded to two decimal points.

To verify your matching condition is correct, you can validate the output schema of defined columns in the **Inspect** tab or get a snapshot of the data in the **Data preview** tab.



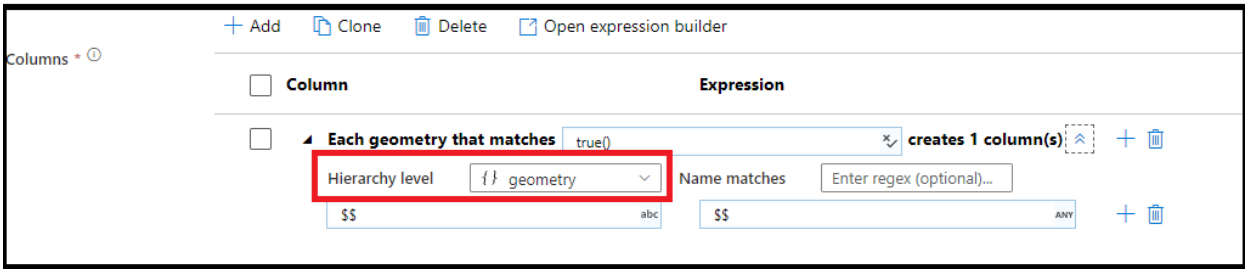
Hierarchical pattern matching

You can build pattern matching inside of complex hierarchical structures as well. Expand the section `Each MoviesStruct that matches` where you will be prompted for each hierarchy in your data stream. You can then build matching patterns for properties within that chosen hierarchy.



Flattening structures

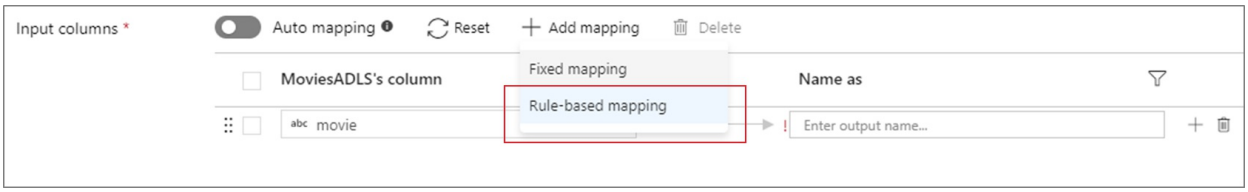
When your data has complex structures like arrays, hierarchical structures, and maps, you can use the Flatten transformation to unroll arrays and denormalize your data. For structures and maps, use the derived column transformation with column patterns to form your flattened relational table from the hierarchies. You can use the column patterns that would look like this sample, which flattens the geography hierarchy into a relational table form:



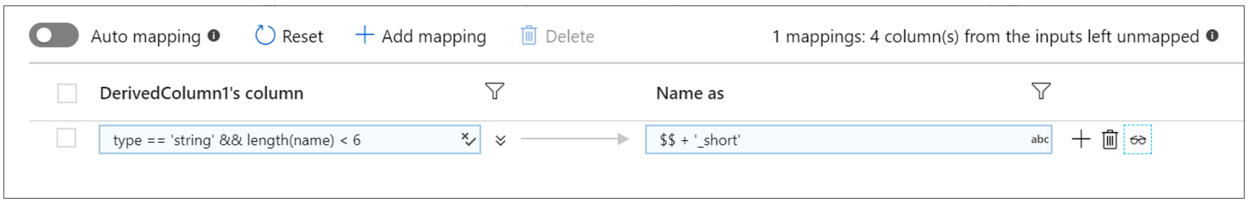
Rule-based mapping in select and sink

When mapping columns in source and select transformations, you can add either fixed mapping or rule-based mappings. Match based on the `name`, `type`, `stream`, `origin`, and `position` of columns. You can have any combination of fixed and rule-based mappings. By default, all projections with greater than 50 columns will default to a rule-based mapping that matches on every column and outputs the inputted name.

To add a rule-based mapping, click **Add mapping** and select **Rule-based mapping**.



Each rule-based mapping requires two inputs: the condition on which to match by and what to name each mapped column. Both values are inputted via the expression builder. In the left expression box, enter your boolean match condition. In the right expression box, specify what the matched column will be mapped to.

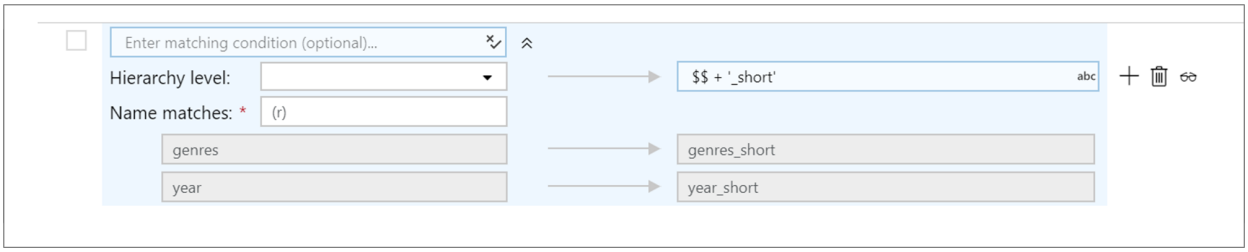


Use `$$` syntax to reference the input name of a matched column. Using the above image as an example, say a user wants to match on all string columns whose names are shorter than six characters. If one incoming column was named `test`, the expression `$$ + '_short'` will rename the column `test_short`. If that's the only mapping that exists, all columns that don't meet the condition will be dropped from the outputted data.

Patterns match both drifted and defined columns. To see which defined columns are mapped by a rule, click the eyeglasses icon next to the rule. Verify your output using data preview.

Regex mapping

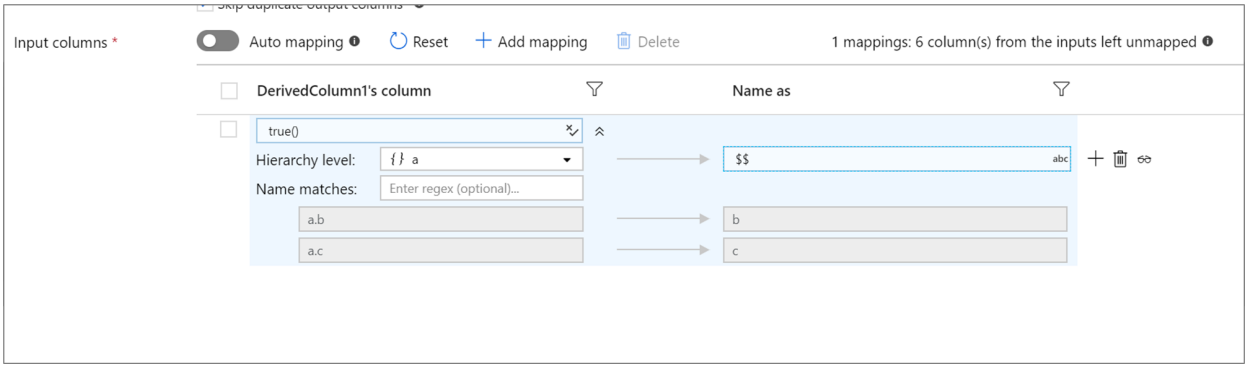
If you click the downward chevron icon, you can specify a regex-mapping condition. A regex-mapping condition matches all column names that match the specified regex condition. This can be used in combination with standard rule-based mappings.



The above example matches on regex pattern `(r)` or any column name that contains a lower case r. Similar to standard rule-based mapping, all matched columns are altered by the condition on the right using `$$` syntax.

Rule-based hierarchies

If your defined projection has a hierarchy, you can use rule-based mapping to map the hierarchies subcolumns. Specify a matching condition and the complex column whose subcolumns you wish to map. Every matched subcolumn will be outputted using the 'Name as' rule specified on the right.



The above example matches on all subcolumns of complex column `a`. `a` contains two subcolumns `b` and `c`.

The output schema will include two columns `b` and `c` as the 'Name as' condition is `$$`.

Pattern matching expression values

- `$$` translates to the name or value of each match at run time. Think of `$$` as equivalent to `this`
- `$0` translates to the current column name match at run time for scalar types. For hierarchical types, `$0` represents the current matched column hierarchy path.
- `name` represents the name of each incoming column
- `type` represents the data type of each incoming column. The list of data types in the data flow type system can be found here.
- `stream` represents the name associated with each stream, or transformation in your flow
- `position` is the ordinal position of columns in your data flow
- `origin` is the transformation where a column originated or was last updated

Next steps

- Learn more about the mapping data flow expression language for data transformations
- Use column patterns in the sink transformation and select transformation with rule-based mapping

Monitor Data Flows

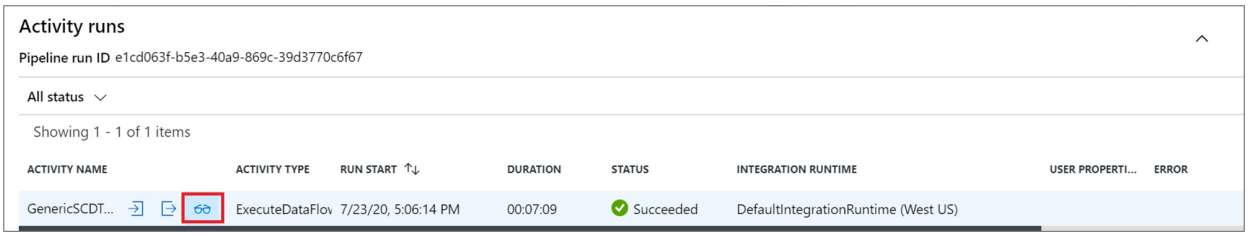
8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

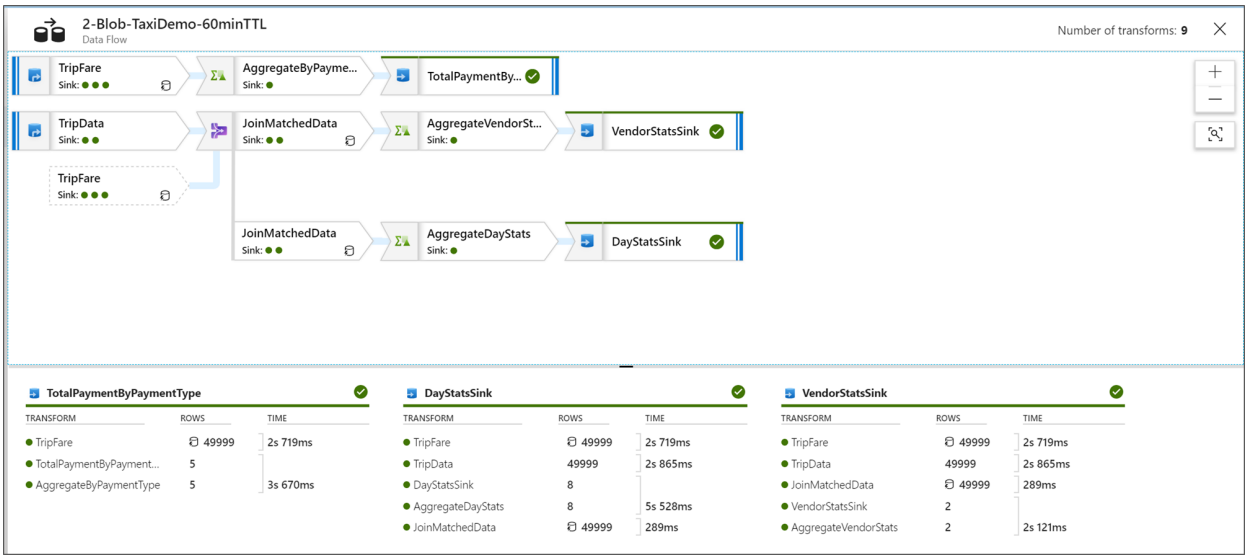
After you have completed building and debugging your data flow, you want to schedule your data flow to execute on a schedule within the context of a pipeline. You can schedule the pipeline using Triggers. For testing and debugging you data flow from a pipeline, you can use the Debug button on the toolbar ribbon or Trigger Now option from the Pipeline Builder to execute a single-run execution to test your data flow within the pipeline context.

When you execute your pipeline, you can monitor the pipeline and all of the activities contained in the pipeline including the Data Flow activity. Click on the monitor icon in the left-hand UI panel. You can see a screen similar to the one below. The highlighted icons allow you to drill into the activities in the pipeline, including the Data Flow activity.

You see statistics at this level as well including the run times and status. The Run ID at the activity level is different than the Run ID at the pipeline level. The Run ID at the previous level is for the pipeline. Selecting the eyeglasses gives you deep details on your data flow execution.

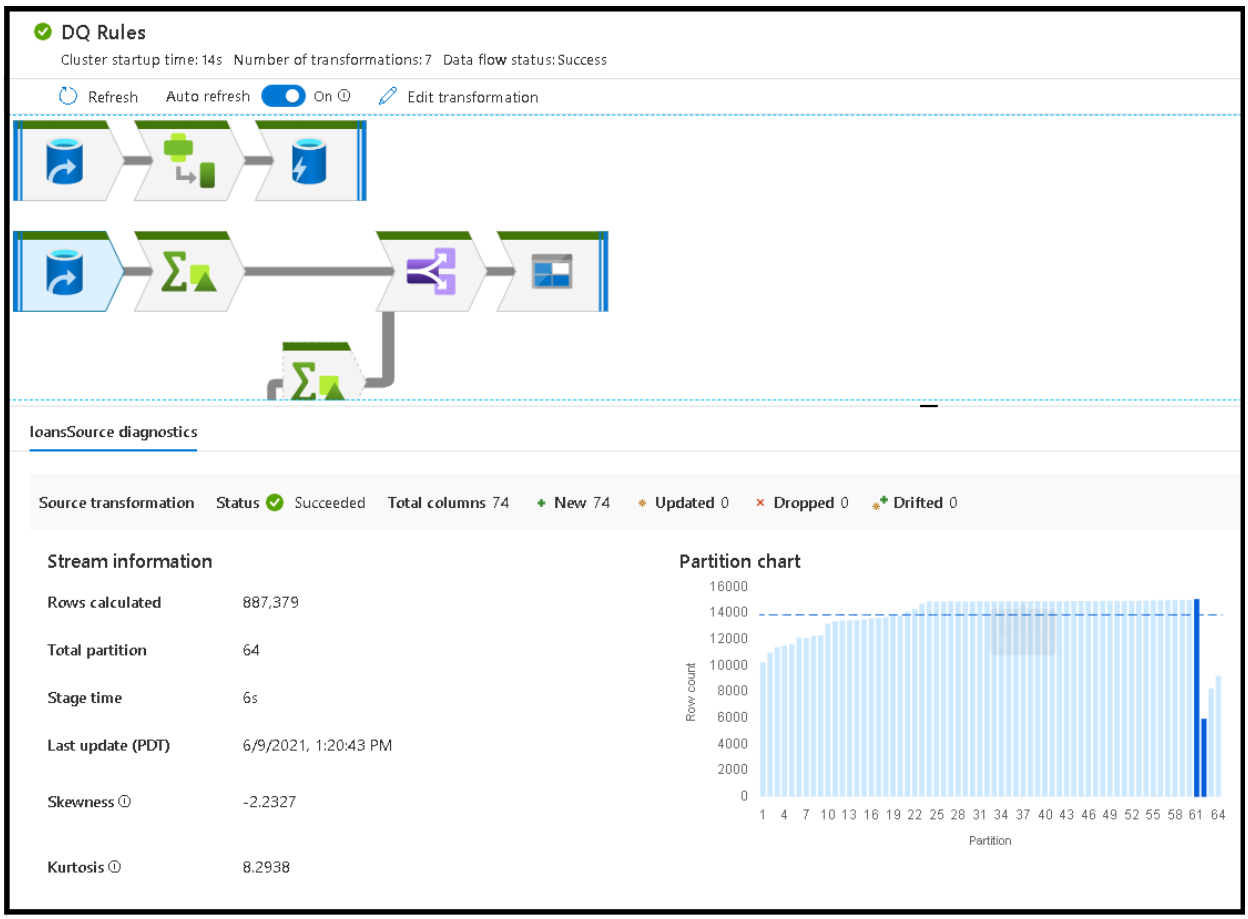


When you're in the graphical node monitoring view, you can see a simplified view-only version of your data flow graph. To see the details view with larger graph nodes that include transformation stage labels, use the zoom slider on the right side of your canvas. You can also use the search button on the right side to find parts of your data flow logic in the graph.



View Data Flow Execution Plans

When your Data Flow is executed in Spark, the service determines optimal code paths based on the entirety of your data flow. Additionally, the execution paths may occur on different scale-out nodes and data partitions. Therefore, the monitoring graph represents the design of your flow, taking into account the execution path of your transformations. When you select individual nodes, you can see "stages" that represent code that was executed together on the cluster. The timings and counts that you see represent those groups or stages as opposed to the individual steps in your design.



- When you select the open space in the monitoring window, the stats in the bottom pane display timing and row counts for each Sink and the transformations that led to the sink data for transformation lineage.
- When you select individual transformations, you receive additional feedback on the right-hand panel that shows partition stats, column counts, skewness (how evenly is the data distributed across partitions), and

kurtosis (how spiky is the data).

- Sorting by *processing time* will help you to identify which stages in your data flow took the most time.
- To find which transformations inside each stage took the most time, sort on *highest processing time*.
- The *rows written* is also sortable as a way to identify which streams inside your data flow are writing the most data.
- When you select the Sink in the node view, you can see column lineage. There are three different methods that columns are accumulated throughout your data flow to land in the Sink. They are:
  - Computed: You use the column for conditional processing or within an expression in your data flow, but don't land it in the Sink
  - Derived: The column is a new column that you generated in your flow, that is, it was not present in the Source
  - Mapped: The column originated from the source and your are mapping it to a sink field
  - Data flow status: The current status of your execution
  - Cluster startup time: Amount of time to acquire the JIT Spark compute environment for your data flow execution
  - Number of transforms: How many transformation steps are being executed in your flow

Total Sink Processing Time vs. Transformation Processing Time

Each transformation stage includes a total time for that stage to complete with each partition execution time totaled together. When you click on the Sink you will see "Sink Processing Time". This time includes the total of the transformation time *plus* the I/O time it took to write your data to your destination store. The difference between the Sink Processing Time and the total of the transformation is the I/O time to write the data.

You can also see detailed timing for each partition transformation step if you open the JSON output from your data flow activity in the pipeline monitoring view. The JSON contains millisecond timing for each partition, whereas the UX monitoring view is an aggregate timing of partitions added together:

```
{
    "stage": 4,
    "partitionTimes": [
         14353,
         14914,
         14246,
         14912,
         ...
        ]
}
```

Sink processing time

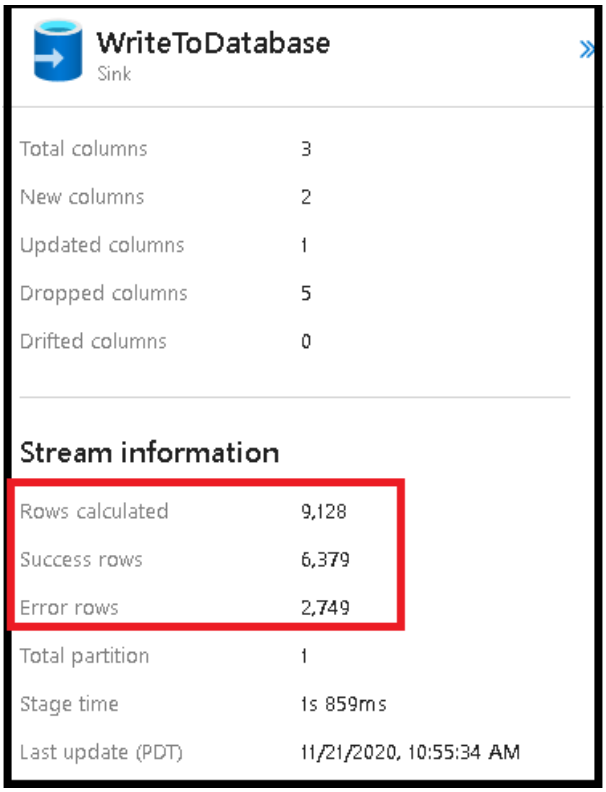
When you select a sink transformation icon in your map, the slide-in panel on the right will show an additional data point called "post processing time" at the bottom. This is the amount time spent executing your job on the Spark cluster *after* your data has been loaded, transformed, and written. This time can include closing connection pools, driver shutdown, deleting files, coalescing files, etc. When you perform actions in your flow like "move files" and "output to single file", you will likely see an increase in the post processing time value.

- Write stage duration: The time to write the data to a staging location for Synapse SQL
- Table operation SQL duration: The time spent moving data from temp tables to target table

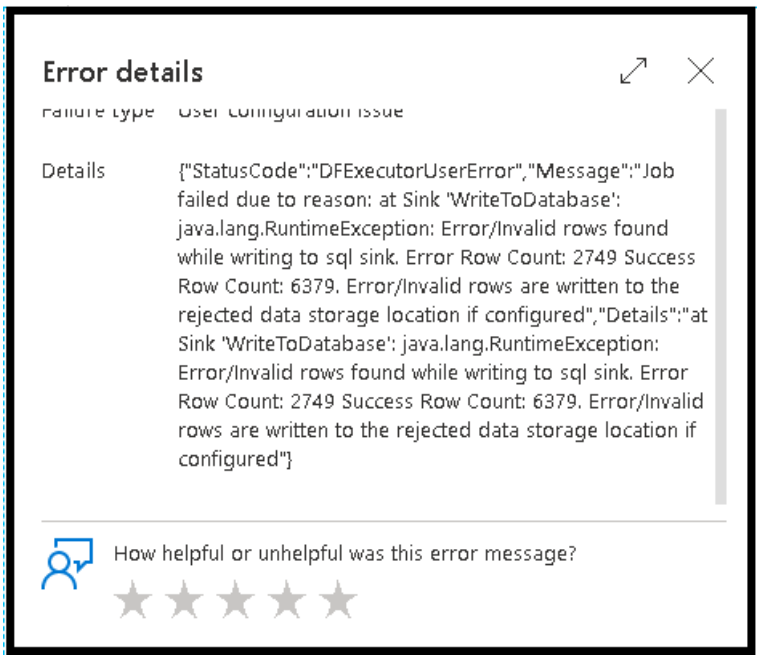
- Pre SQL duration & Post SQL duration: The time spent running pre/post SQL commands
- Pre commands duration & post commands duration: The time spent running any pre/post operations for file based source/sinks. For example move or delete files after processing.
- Merge duration: The time spent merging the file, merge files are used for file based sinks when writing to single file or when "File name as column data" is used. If significant time is spent in this metric, you should avoid using these options.
- Stage time: Total amount of time spent inside of Spark to complete the operation as a stage.
- Temporary staging stable: Name of the temporary table used by data flows to stage data in the database.

Error rows

Enabling error row handling in your data flow sink will be reflected in the monitoring output. When you set the sink to "report success on error", the monitoring output will show the number of success and failed rows when you click on the sink monitoring node.

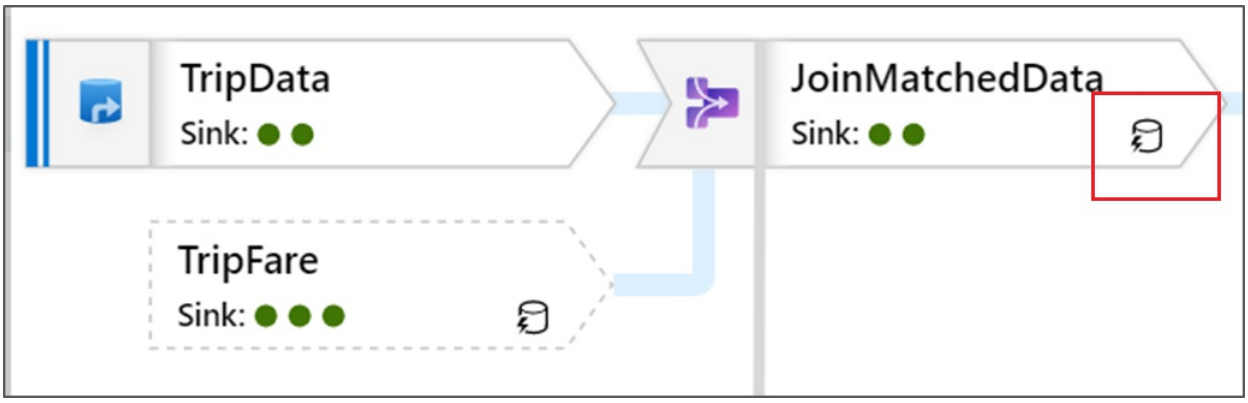


When you select "report failure on error", the same output will be shown only in the activity monitoring output text. This is because the data flow activity will return failure for execution and the detailed monitoring view will be unavailable.



Monitor Icons

This icon means that the transformation data was already cached on the cluster, so the timings and execution path have taken that into account:



You also see green circle icons in the transformation. They represent a count of the number of sinks that data is flowing into.

Mapping data flows performance and tuning guide

8/19/2022 • 5 minutes to read • Edit Online

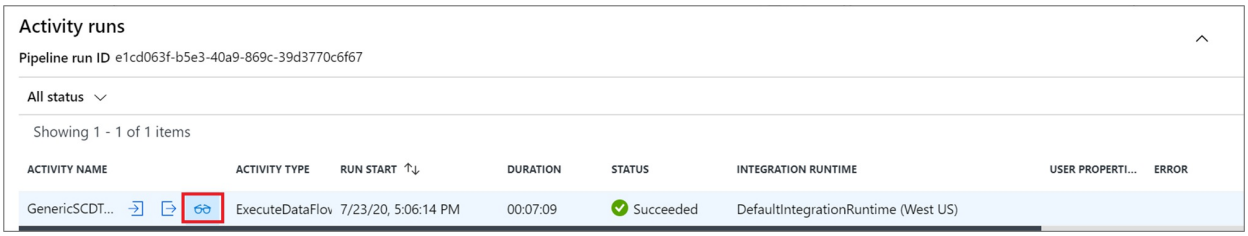
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Mapping data flows in Azure Data Factory and Synapse pipelines provide a code-free interface to design and run data transformations at scale. If you're not familiar with mapping data flows, see the Mapping Data Flow Overview. This article highlights various ways to tune and optimize your data flows so that they meet your performance benchmarks.

Watch the below video to see shows some sample timings transforming data with data flows.

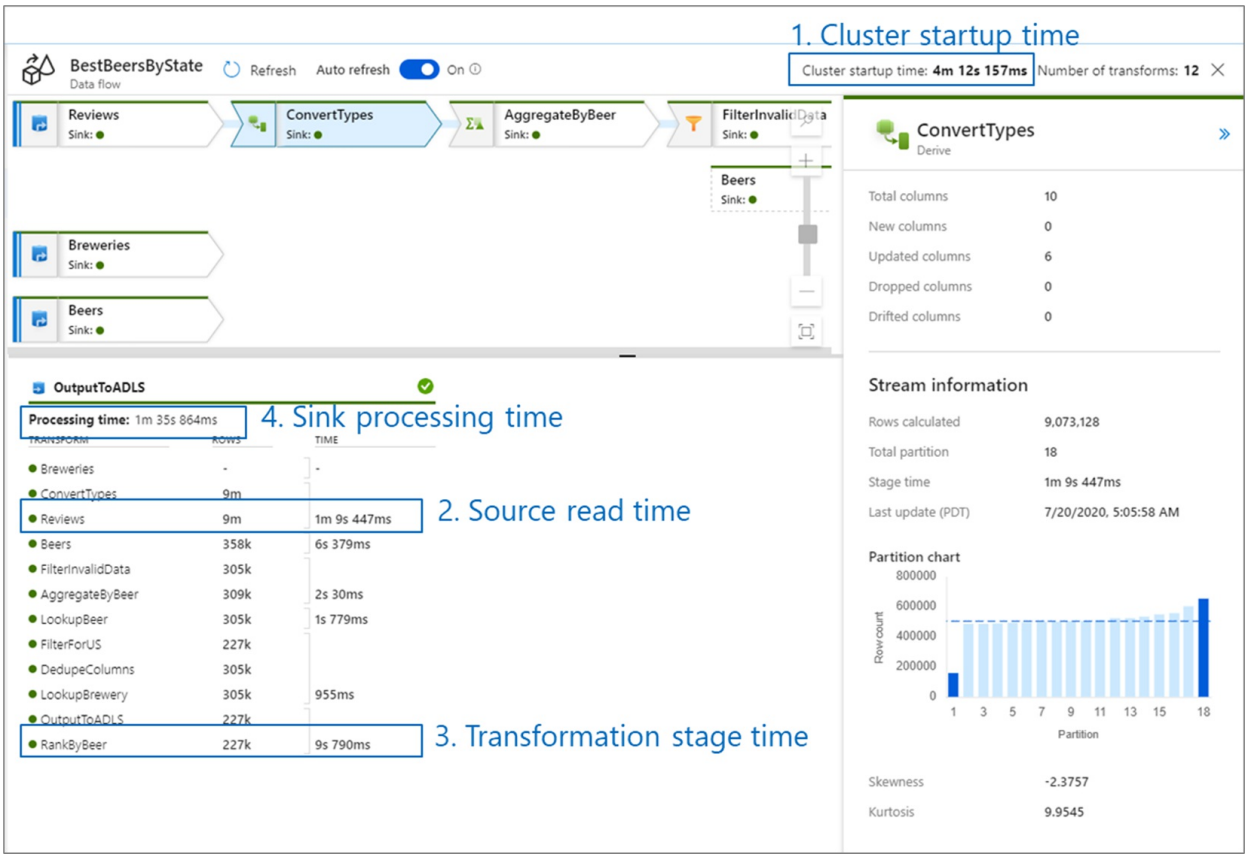
Monitoring data flow performance

Once you verify your transformation logic using debug mode, run your data flow end-to-end as an activity in a pipeline. Data flows are operationalized in a pipeline using the execute data flow activity. The data flow activity has a unique monitoring experience compared to other activities that displays a detailed execution plan and performance profile of the transformation logic. To view detailed monitoring information of a data flow, click on the eyeglasses icon in the activity run output of a pipeline. For more information, see Monitoring mapping data flows.



When monitoring data flow performance, there are four possible bottlenecks to look out for:

- Cluster start-up time
- Reading from a source
- Transformation time
- Writing to a sink



Cluster start-up time is the time it takes to spin up an Apache Spark cluster. This value is located in the top-right corner of the monitoring screen. Data flows run on a just-in-time model where each job uses an isolated cluster. This start-up time generally takes 3-5 minutes. For sequential jobs, this can be reduced by enabling a time to live value. For more information, refer to the **Time to live** section in Integration Runtime performance.

Data flows utilize a Spark optimizer that reorders and runs your business logic in 'stages' to perform as quickly as possible. For each sink that your data flow writes to, the monitoring output lists the duration of each transformation stage, along with the time it takes to write data into the sink. The time that is the largest is likely the bottleneck of your data flow. If the transformation stage that takes the largest contains a source, then you may want to look at further optimizing your read time. If a transformation is taking a long time, then you may need to repartition or increase the size of your integration runtime. If the sink processing time is large, you may need to scale up your database or verify you are not outputting to a single file.

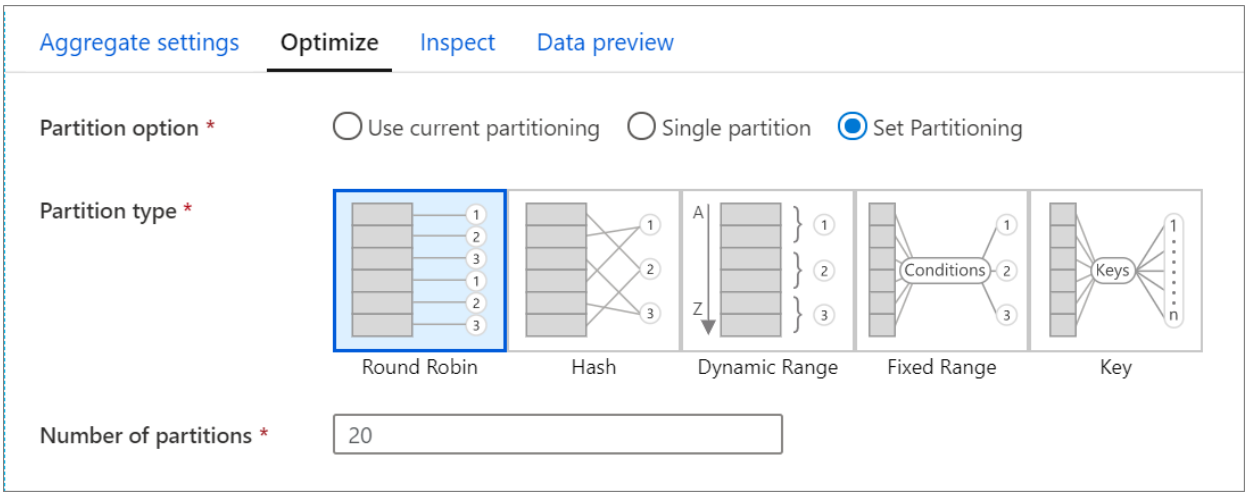
Once you have identified the bottleneck of your data flow, use the below optimizations strategies to improve performance.

Testing data flow logic

When designing and testing data flows from UI, debug mode allows you to interactively test against a live Spark cluster. This allows you to preview data and execute your data flows without waiting for a cluster to warm up. For more information, see Debug Mode.

Optimize tab

The **Optimize** tab contains settings to configure the partitioning scheme of the Spark cluster. This tab exists in every transformation of data flow and specifies whether you want to repartition the data **after** the transformation has completed. Adjusting the partitioning provides control over the distribution of your data across compute nodes and data locality optimizations that can have both positive and negative effects on your overall data flow performance.



By default, *Use current partitioning* is selected which instructs the service keep the current output partitioning of the transformation. As repartitioning data takes time, *Use current partitioning* is recommended in most scenarios. Scenarios where you may want to repartition your data include after aggregates and joins that significantly skew your data or when using Source partitioning on a SQL DB.

To change the partitioning on any transformation, select the **Optimize** tab and select the **Set Partitioning** radio button. You are presented with a series of options for partitioning. The best method of partitioning differs based on your data volumes, candidate keys, null values, and cardinality.

> **IMPORTANT**
> Single partition combines all the distributed data into a single partition. This is a very slow operation that also significantly affects all downstream transformation and writes. This option is strongly discouraged unless there is an explicit business reason to use it.

The following partitioning options are available in every transformation:

Round robin

Round robin distributes data equally across partitions. Use round-robin when you don't have good key candidates to implement a solid, smart partitioning strategy. You can set the number of physical partitions.

Hash

The service produces a hash of columns to produce uniform partitions such that rows with similar values fall in the same partition. When you use the Hash option, test for possible partition skew. You can set the number of physical partitions.

Dynamic range

The dynamic range uses Spark dynamic ranges based on the columns or expressions that you provide. You can set the number of physical partitions.

Fixed range

Build an expression that provides a fixed range for values within your partitioned data columns. To avoid partition skew, you should have a good understanding of your data before you use this option. The values you enter for the expression are used as part of a partition function. You can set the number of physical partitions.

Key

If you have a good understanding of the cardinality of your data, key partitioning might be a good strategy. Key partitioning creates partitions for each unique value in your column. You can't set the number of partitions because the number is based on unique values in the data.

TIP

Manually setting the partitioning scheme reshuffles the data and can offset the benefits of the Spark optimizer. A best practice is to not manually set the partitioning unless you need to.

Logging level

If you do not require every pipeline execution of your data flow activities to fully log all verbose telemetry logs, you can optionally set your logging level to "Basic" or "None". When executing your data flows in "Verbose" mode (default), you are requesting the service to fully log activity at each individual partition level during your data transformation. This can be an expensive operation, so only enabling verbose when troubleshooting can improve your overall data flow and pipeline performance. "Basic" mode will only log transformation durations while "None" will only provide a summary of durations.

Next steps

- Optimizing sources
- Optimizing sinks
- Optimizing transformations
- Using data flows in pipelines

See other Data Flow articles related to performance:

- Data Flow activity
- Monitor Data Flow performance
- Integration Runtime performance

Optimizing sources

8/19/2022 • 2 minutes to read • Edit Online

For every source except Azure SQL Database, it is recommended that you keep **Use current partitioning** as the selected value. When reading from all other source systems, data flows automatically partitions data evenly based upon the size of the data. A new partition is created for about every 128 MB of data. As your data size increases, the number of partitions increase.

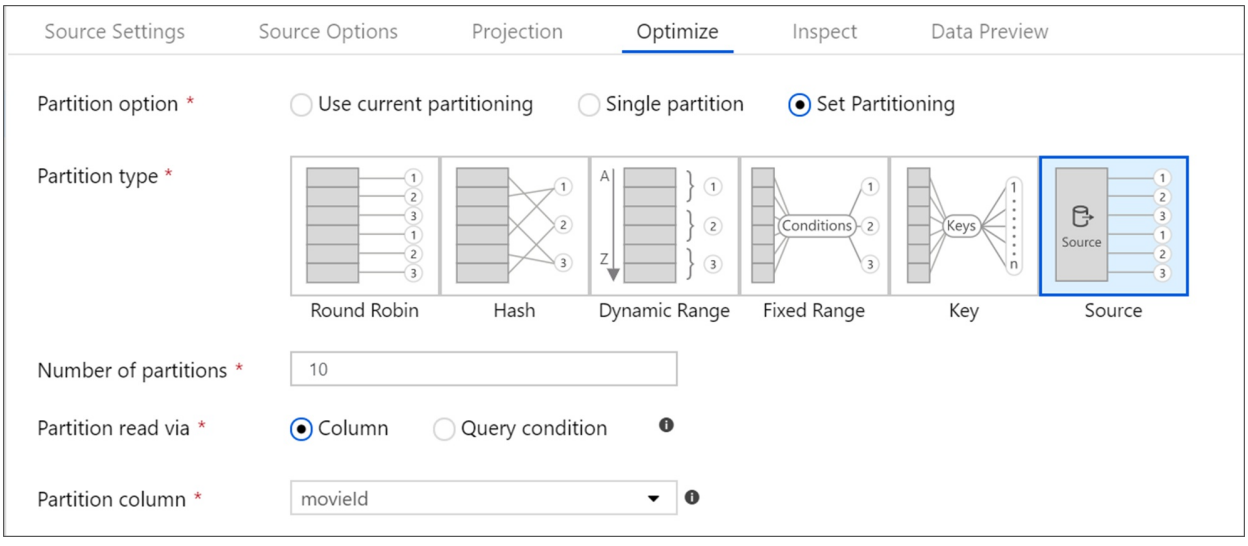
Any custom partitioning happens *after* Spark reads in the data and will negatively impact your data flow performance. As the data is evenly partitioned on read, this is not recommended.

> **NOTE**
> Read speeds can be limited by the throughput of your source system.

Azure SQL Database sources

Azure SQL Database has a unique partitioning option called 'Source' partitioning. Enabling source partitioning can improve your read times from Azure SQL DB by enabling parallel connections on the source system. Specify the number of partitions and how to partition your data. Use a partition column with high cardinality. You can also enter a query that matches the partitioning scheme of your source table.

> **TIP**
> For source partitioning, the I/O of the SQL Server is the bottleneck. Adding too many partitions may saturate your source database. Generally four or five partitions is ideal when using this option.



Isolation level

The isolation level of the read on an Azure SQL source system has an impact on performance. Choosing 'Read uncommitted' will provide the fastest performance and prevent any database locks. To learn more about SQL Isolation levels, please see Understanding isolation levels.

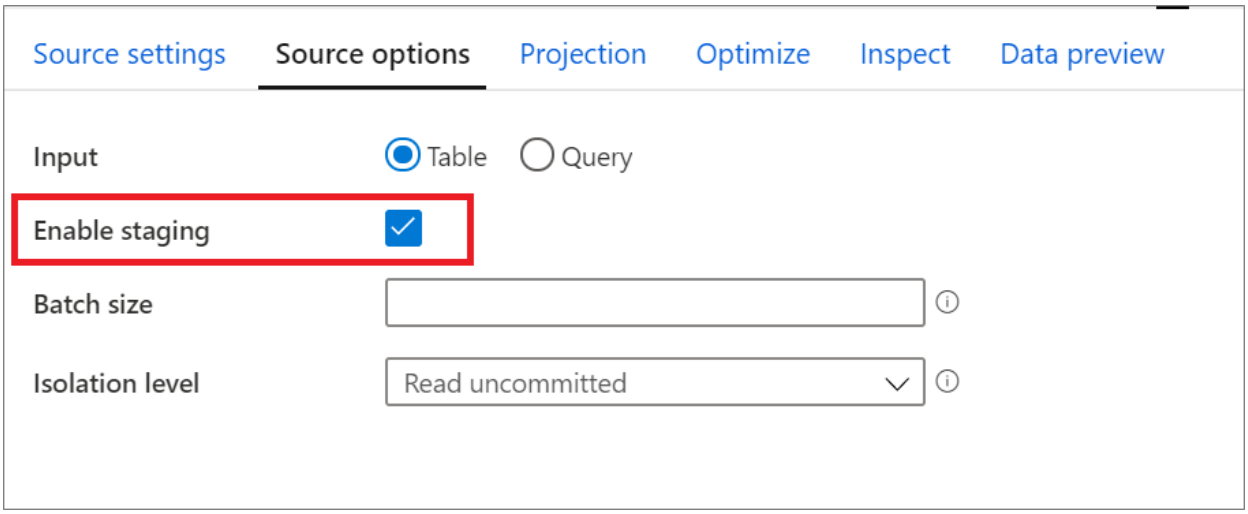
Read using query

You can read from Azure SQL Database using a table or a SQL query. If you are executing a SQL query, the

query must complete before transformation can start. SQL Queries can be useful to push down operations that may execute faster and reduce the amount of data read from a SQL Server such as SELECT, WHERE, and JOIN statements. When pushing down operations, you lose the ability to track lineage and performance of the transformations before the data comes into the data flow.

Azure Synapse Analytics sources

When using Azure Synapse Analytics, a setting called **Enable staging** exists in the source options. This allows the service to read from Synapse using `Staging` which greatly improves read performance by using the most performant bulk loading capability such as CETAS and COPY command. Enabling `Staging` requires you to specify an Azure Blob Storage or Azure Data Lake Storage gen2 staging location in the data flow activity settings.



File-based sources

While data flows support a variety of file types, the Spark-native Parquet format is recommended for optimal read and write times.

If you're running the same data flow on a set of files, we recommend reading from a folder, using wildcard paths or reading from a list of files. A single data flow activity run can process all of your files in batch. More information on how to configure these settings can be found in the **Source transformation** section of the Azure Blob Storage connector documentation.

If possible, avoid using the For-Each activity to run data flows over a set of files. This will cause each iteration of the for-each to spin up its own Spark cluster, which is often not necessary and can be expensive.

Next steps

- Data flow performance overview
- Optimizing sinks
- Optimizing transformations
- Using data flows in pipelines

See other Data Flow articles related to performance:

- Data Flow activity
- Monitor Data Flow performance
- Integration Runtime performance

Optimizing sinks

8/19/2022 • 4 minutes to read • Edit Online

When data flows write to sinks, any custom partitioning will happen immediately before the write. Like the source, in most cases it is recommended that you keep **Use current partitioning** as the selected partition option. Partitioned data will write significantly quicker than unpartitioned data, even your destination is not partitioned. Below are the individual considerations for various sink types.

Azure SQL Database sinks

With Azure SQL Database, the default partitioning should work in most cases. There is a chance that your sink may have too many partitions for your SQL database to handle. If you are running into this, reduce the number of partitions outputted by your SQL Database sink.

Best practice for deleting rows in sink based on missing rows in source

Here is a video walk through of how to use data flows with exists, alter row, and sink transformations to achieve this common pattern:

Impact of error row handling to performance

When you enable error row handling ("continue on error") in the sink transformation, the service will take an additional step before writing the compatible rows to your destination table. This additional step will have a small performance penalty that can be in the range of 5% added for this step with an additional small performance hit also added if you set the option to also write the incompatible rows to a log file.

Disabling indexes using a SQL Script

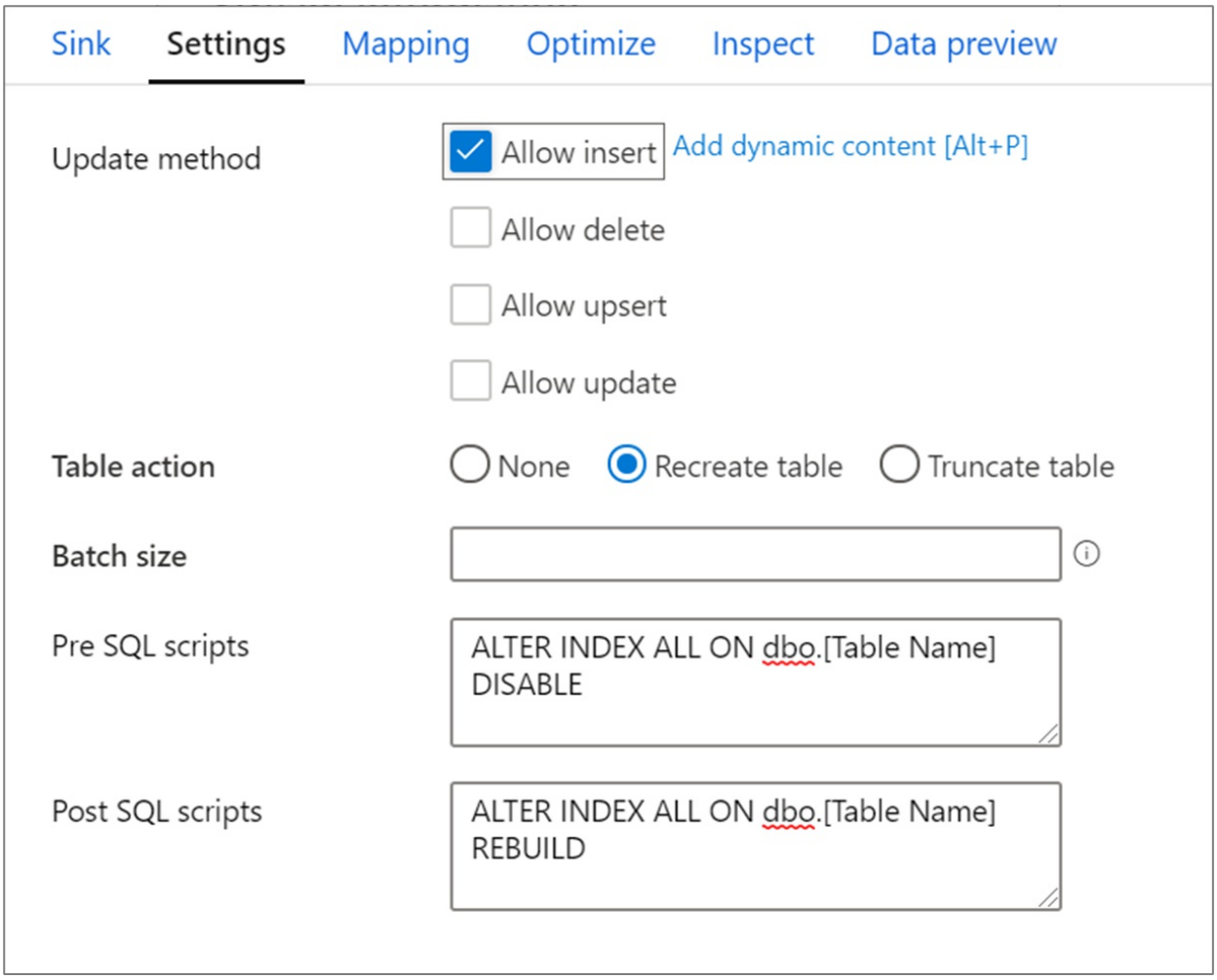
Disabling indexes before a load in a SQL database can greatly improve performance of writing to the table. Run the below command before writing to your SQL sink.

```
ALTER INDEX ALL ON dbo.[Table Name] DISABLE
```

After the write has completed, rebuild the indexes using the following command:

```
ALTER INDEX ALL ON dbo.[Table Name] REBUILD
```

These can both be done natively using Pre and Post-SQL scripts within an Azure SQL DB or Synapse sink in mapping data flows.



WARNING

When disabling indexes, the data flow is effectively taking control of a database and queries are unlikely to succeed at this time. As a result, many ETL jobs are triggered in the middle of the night to avoid this conflict. For more information, learn about the constraints of disabling SQL indexes

Scaling up your database

Schedule a resizing of your source and sink Azure SQL DB and DW before your pipeline run to increase the throughput and minimize Azure throttling once you reach DTU limits. After your pipeline execution is complete, resize your databases back to their normal run rate.

Azure Synapse Analytics sinks

When writing to Azure Synapse Analytics, make sure that **Enable staging** is set to true. This enables the service to write using the SQL COPY Command which effectively loads the data in bulk. You will need to reference an Azure Data Lake Storage gen2 or Azure Blob Storage account for staging of the data when using Staging.

Other than Staging, the same best practices apply to Azure Synapse Analytics as Azure SQL Database.

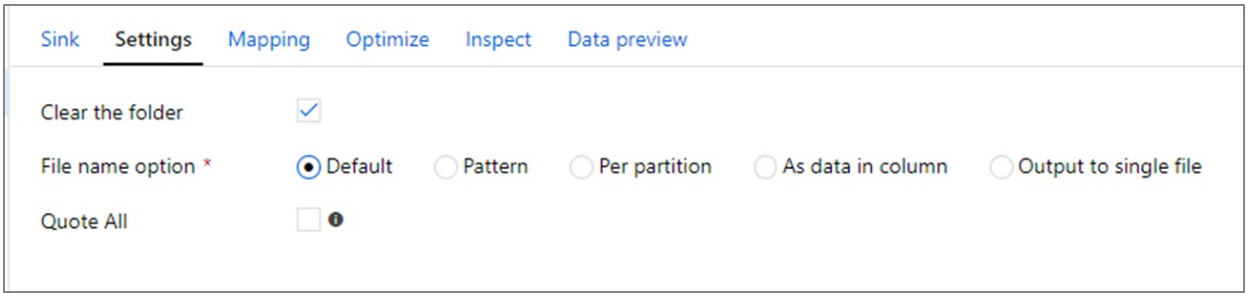
File-based sinks

While data flows support a variety of file types, the Spark-native Parquet format is recommended for optimal read and write times.

If the data is evenly distributed, **Use current partitioning** will be the fastest partitioning option for writing files.

File name options

When writing files, you have a choice of naming options that each have a performance impact.



Selecting the **Default** option will write the fastest. Each partition will equate to a file with the Spark default name. This is useful if you are just reading from the folder of data.

Setting a naming **Pattern** will rename each partition file to a more user-friendly name. This operation happens after write and is slightly slower than choosing the default.

Per partition allows you to name each individual partition manually.

If a column corresponds to how you wish to output the data, you can select **Name file as column data**. This reshuffles the data and can impact performance if the columns are not evenly distributed.

If a column corresponds to how you wish to generate folder names, select **Name folder as column data**.

Output to single file combines all the data into a single partition. This leads to long write times, especially for large datasets. This option is strongly discouraged unless there is an explicit business reason to use it.

Azure Cosmos DB sinks

When writing to Azure Cosmos DB, altering throughput and batch size during data flow execution can improve performance. These changes only take effect during the data flow activity run and will return to the original collection settings after conclusion.

Batch size: Usually, starting with the default batch size is sufficient. To further tune this value, calculate the rough object size of your data, and make sure that object size * batch size is less than 2MB. If it is, you can increase the batch size to get better throughput.

Throughput: Set a higher throughput setting here to allow documents to write faster to Azure Cosmos DB. Keep in mind the higher RU costs based upon a high throughput setting.

Write throughput budget: Use a value which is smaller than total RUs per minute. If you have a data flow with a high number of Spark partitions, setting a budget throughput will allow more balance across those partitions.

Next steps

- Data flow performance overview
- Optimizing sources
- Optimizing transformations
- Using data flows in pipelines

See other Data Flow articles related to performance:

- Data Flow activity
- Monitor Data Flow performance
- Integration Runtime performance

Optimizing transformations

8/19/2022 • 4 minutes to read • Edit Online

Use the following strategies to optimize performance of transformations in mapping data flows in Azure Data Factory and Azure Synapse Analytics pipelines.

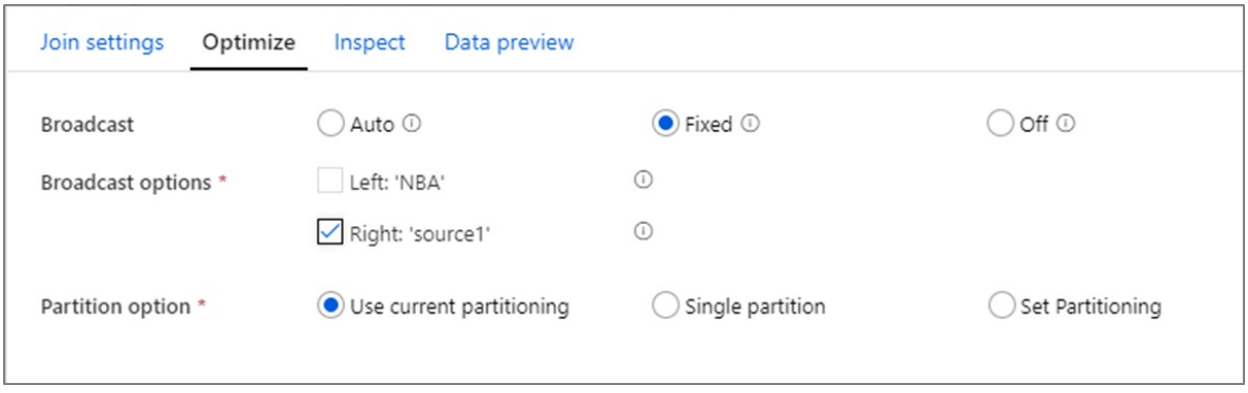
Optimizing Joins, Exists, and Lookups

Broadcasting

In joins, lookups, and exists transformations, if one or both data streams are small enough to fit into worker node memory, you can optimize performance by enabling **Broadcasting**. Broadcasting is when you send small data frames to all nodes in the cluster. This allows for the Spark engine to perform a join without reshuffling the data in the large stream. By default, the Spark engine will automatically decide whether or not to broadcast one side of a join. If you are familiar with your incoming data and know that one stream will be significantly smaller than the other, you can select **Fixed** broadcasting. Fixed broadcasting forces Spark to broadcast the selected stream.

If the size of the broadcasted data is too large for the Spark node, you may get an out of memory error. To avoid out of memory errors, use **memory optimized** clusters. If you experience broadcast timeouts during data flow executions, you can switch off the broadcast optimization. However, this will result in slower performing data flows.

When working with data sources that can take longer to query, like large database queries, it is recommended to turn broadcast off for joins. Source with long query times can cause Spark timeouts when the cluster attempts to broadcast to compute nodes. Another good choice for turning off broadcast is when you have a stream in your data flow that is aggregating values for use in a lookup transformation later. This pattern can confuse the Spark optimizer and cause timeouts.



Cross joins

If you use literal values in your join conditions or have multiple matches on both sides of a join, Spark will run the join as a cross join. A cross join is a full cartesian product that then filters out the joined values. This is significantly slower than other join types. Ensure that you have column references on both sides of your join conditions to avoid the performance impact.

Sorting before joins

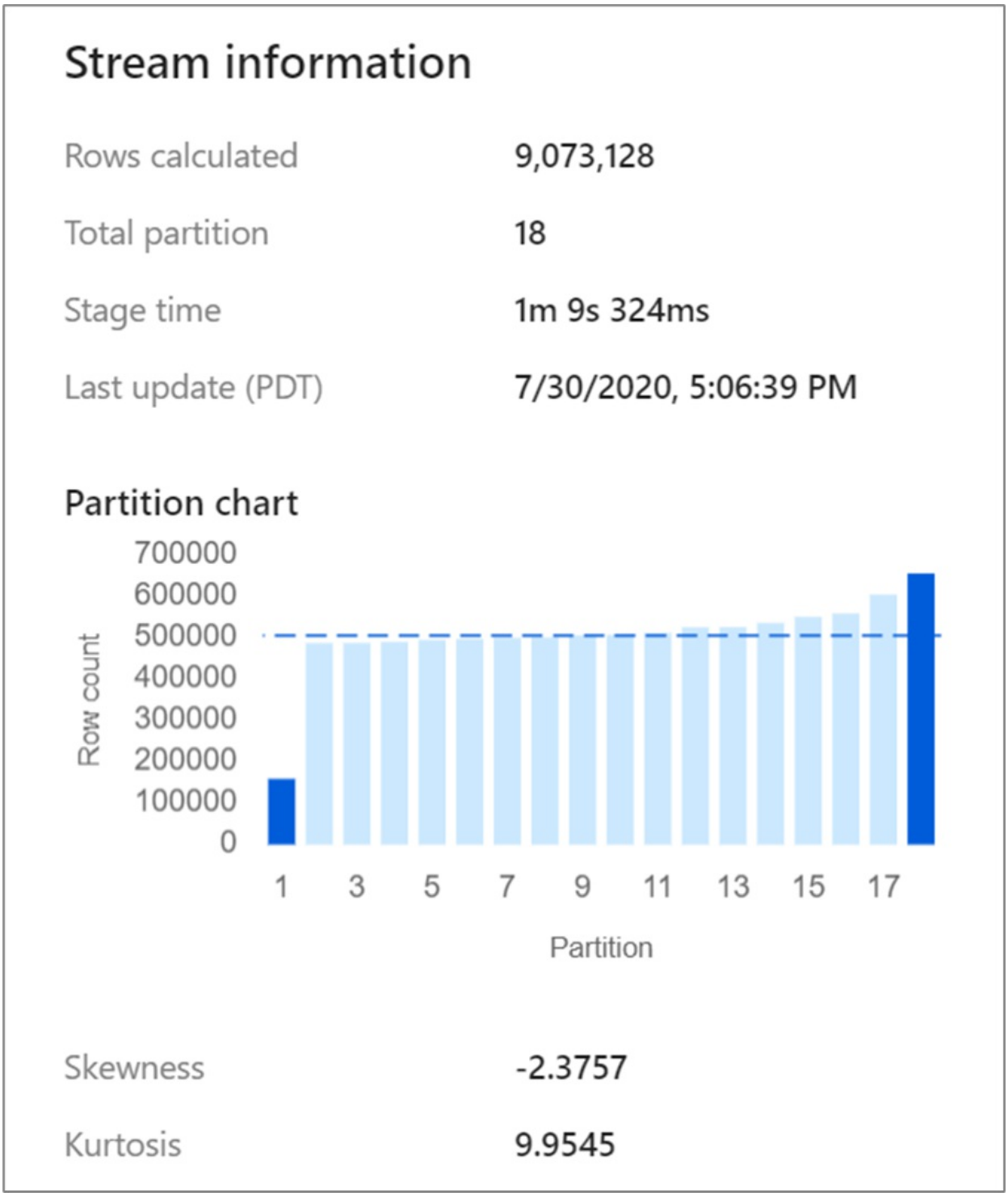
Unlike merge join in tools like SSIS, the join transformation isn't a mandatory merge join operation. The join keys don't require sorting prior to the transformation. Using Sort transformations in mapping data flows is not recommended.

Window transformation performance

The Window transformation in mapping data flow partitions your data by value in columns that you select as part of the `over()` clause in the transformation settings. There are a number of very popular aggregate and analytical functions that are exposed in the Windows transformation. However, if your use case is to generate a window over your entire dataset for the purpose of ranking `rank()` or row number `rowNumber()`, it is recommended that you instead use the Rank transformation and the Surrogate Key transformation. Those transformation will perform better again full dataset operations using those functions.

Repartitioning skewed data

Certain transformations such as joins and aggregates reshuffle your data partitions and can occasionally lead to skewed data. Skewed data means that data is not evenly distributed across the partitions. Heavily skewed data can lead to slower downstream transformations and sink writes. You can check the skewness of your data at any point in a data flow run by clicking on the transformation in the monitoring display.



The monitoring display will show how the data is distributed across each partition along with two metrics, skewness and kurtosis. **Skewness** is a measure of how asymmetrical the data is and can have a positive, zero, negative, or undefined value. Negative skew means the left tail is longer than the right. **Kurtosis** is the measure of whether the data is heavy-tailed or light-tailed. High kurtosis values are not desirable. Ideal ranges of skewness lie between -3 and 3 and ranges of kurtosis are less than 10. An easy way to interpret these numbers is looking at the partition chart and seeing if 1 bar is significantly larger than the rest.

If your data is not evenly partitioned after a transformation, you can use the optimize tab to repartition. Reshuffling data takes time and may not improve your data flow performance.

> **TIP**
> If you repartition your data, but have downstream transformations that reshuffle your data, use hash partitioning on a column used as a join key.

> **NOTE**
> Transformations inside your data flow (with the exception of the Sink transformation) do not modify the file and folder partitioning of data at rest. Partitioning in each transformation repartitions data inside the data frames of the temporary serverless Spark cluster that ADF manages for each of your data flow executions.

Next steps

- Data flow performance overview
- Optimizing sources
- Optimizing sinks
- Using data flows in pipelines

See other Data Flow articles related to performance:

- Data Flow activity
- Monitor Data Flow performance
- Integration Runtime performance

Using data flows in pipelines

8/19/2022 • 2 minutes to read • Edit Online

When building complex pipelines with multiple data flows, your logical flow can have a big impact on timing and cost. This section covers the impact of different architecture strategies.

Executing data flows in parallel

If you execute multiple data flows in parallel, the service spins up separate Spark clusters for each activity. This allows for each job to be isolated and run in parallel, but will lead to multiple clusters running at the same time.

If your data flows execute in parallel, we recommend that you don't enable the Azure IR time to the live property because it will lead to multiple unused warm pools.

> **TIP**
> Instead of running the same data flow multiple times in a for each activity, stage your data in a data lake and use wildcard paths to process the data in a single data flow.

Execute data flows sequentially

If you execute your data flow activities in sequence, it is recommended that you set a TTL in the Azure IR configuration. The service will reuse the compute resources, resulting in a faster cluster start-up time. Each activity will still be isolated and receive a new Spark context for each execution.

Overloading a single data flow

If you put all of your logic inside of a single data flow, the service will execute the entire job on a single Spark instance. While this may seem like a way to reduce costs, it mixes together different logical flows and can be difficult to monitor and debug. If one component fails, all other parts of the job will fail as well. Organizing data flows by independent flows of business logic is recommended. If your data flow becomes too large, splitting it into separate components will make monitoring and debugging easier. While there is no hard limit on the number of transformations in a data flow, having too many will make the job complex.

Execute sinks in parallel

The default behavior of data flow sinks is to execute each sink sequentially, in a serial manner, and to fail the data flow when an error is encountered in the sink. Additionally, all sinks are defaulted to the same group unless you go into the data flow properties and set different priorities for the sinks.

Data flows allow you to group sinks together into groups from the data flow properties tab in the UI designer. You can both set the order of execution of your sinks as well as to group sinks together using the same group number. To help manage groups, you can ask the service to run sinks in the same group, to run in parallel.

On the pipeline execute data flow activity under the "Sink Properties" section is an option to turn on parallel sink loading. When you enable "run in parallel", you are instructing data flows write to connected sinks at the same time rather than in a sequential manner. In order to utilize the parallel option, the sinks must be group together and connected to the same stream via a New Branch or Conditional Split.

Next steps

- Data flow performance overview
- Optimizing sources
- Optimizing sinks
- Optimizing transformations

See other Data Flow articles related to performance:

- Data Flow activity
- Monitor Data Flow performance
- Integration Runtime performance

Memory optimized compute type for Data Flows in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flow activities in Azure Data Factory and Azure Synapse support the Compute type setting to help optimize the cluster configuration for cost and performance of the workload. The default selection for the setting is **General** and will be sufficient for most data flow workloads. General purpose clusters typically provide the best balance of performance and cost. However, the **Memory optimized** setting can significantly improve performance in some scenarios by maximizing the memory available per core for the cluster.

When to use the memory optimized compute type

If your data flow has many joins and lookups, you may want to use a memory optimized cluster. These more memory intensive operations will benefit particularly by additional memory, and any out-of-memory errors encountered with the default compute type will be minimized. **Memory optimized** clusters do incur the highest cost per core, but may avoid pipeline failures for memory intensive operations. If you experience any out of memory errors when executing data flows, switch to a memory optimized Azure IR configuration.

Next steps

Data Flow type properties

Optimizing performance of the Azure Integration Runtime

8/19/2022 • 3 minutes to read • Edit Online

Data flows run on Spark clusters that are spun up at run-time. The configuration for the cluster used is defined in the integration runtime (IR) of the activity. There are three performance considerations to make when defining your integration runtime: cluster type, cluster size, and time to live.

For more information how to create an Integration Runtime, see Integration Runtime.

The easiest way to get started with data flow integration runtimes is to choose small, medium, or large from the compute size picker. See the mappings to cluster configurations for those sizes below.

Cluster type

There are two available options for the type of Spark cluster to utilize: general purpose & memory optimized.

General purpose clusters are the default selection and will be ideal for most data flow workloads. These tend to be the best balance of performance and cost.

If your data flow has many joins and lookups, you may want to use a **memory optimized** cluster. Memory optimized clusters can store more data in memory and will minimize any out-of-memory errors you may get. Memory optimized have the highest price-point per core, but also tend to result in more successful pipelines. If you experience any out of memory errors when executing data flows, switch to a memory optimized Azure IR configuration.

Cluster size

Data flows distribute the data processing over different nodes in a Spark cluster to perform operations in parallel. A Spark cluster with more cores increases the number of nodes in the compute environment. More nodes increase the processing power of the data flow. Increasing the size of the cluster is often an easy way to reduce the processing time.

The default cluster size is four driver nodes and four worker nodes (small). As you process more data, larger clusters are recommended. Below are the possible sizing options:

WORKER CORES	DRIVER CORES	TOTAL CORES	NOTES
4	4	8	Small
8	8	16	Medium
16	16	32	
32	16	48	
64	16	80	Large
128	16	144	

WORKER CORES	DRIVER CORES	TOTAL CORES	NOTES
256	16	272	

Data flows are priced at vcore-hrs meaning that both cluster size and execution-time factor into this. As you scale up, your cluster cost per minute will increase, but your overall time will decrease.

> **TIP**
> There is a ceiling on how much the size of a cluster affects the performance of a data flow. Depending on the size of your data, there is a point where increasing the size of a cluster will stop improving performance. For example, If you have more nodes than partitions of data, adding additional nodes won't help. A best practice is to start small and scale up to meet your performance needs.

Time to live

By default, every data flow activity spins up a new Spark cluster based upon the Azure IR configuration. Cold cluster start-up time takes a few minutes and data processing can't start until it is complete. If your pipelines contain multiple **sequential** data flows, you can enable a time to live (TTL) value. Specifying a time to live value keeps a cluster alive for a certain period of time after its execution completes. If a new job starts using the IR during the TTL time, it will reuse the existing cluster and start up time will greatly reduced. After the second job completes, the cluster will again stay alive for the TTL time.

However, if most of your data flows execute in parallel, it is not recommended that you enable TTL for the IR that you use for those activities. Only one job can run on a single cluster at a time. If there is an available cluster, but two data flows start, only one will use the live cluster. The second job will spin up its own isolated cluster.

> **NOTE**
> Time to live is not available when using the auto-resolve integration runtime (default).

Next steps

See other Data Flow articles related to performance:

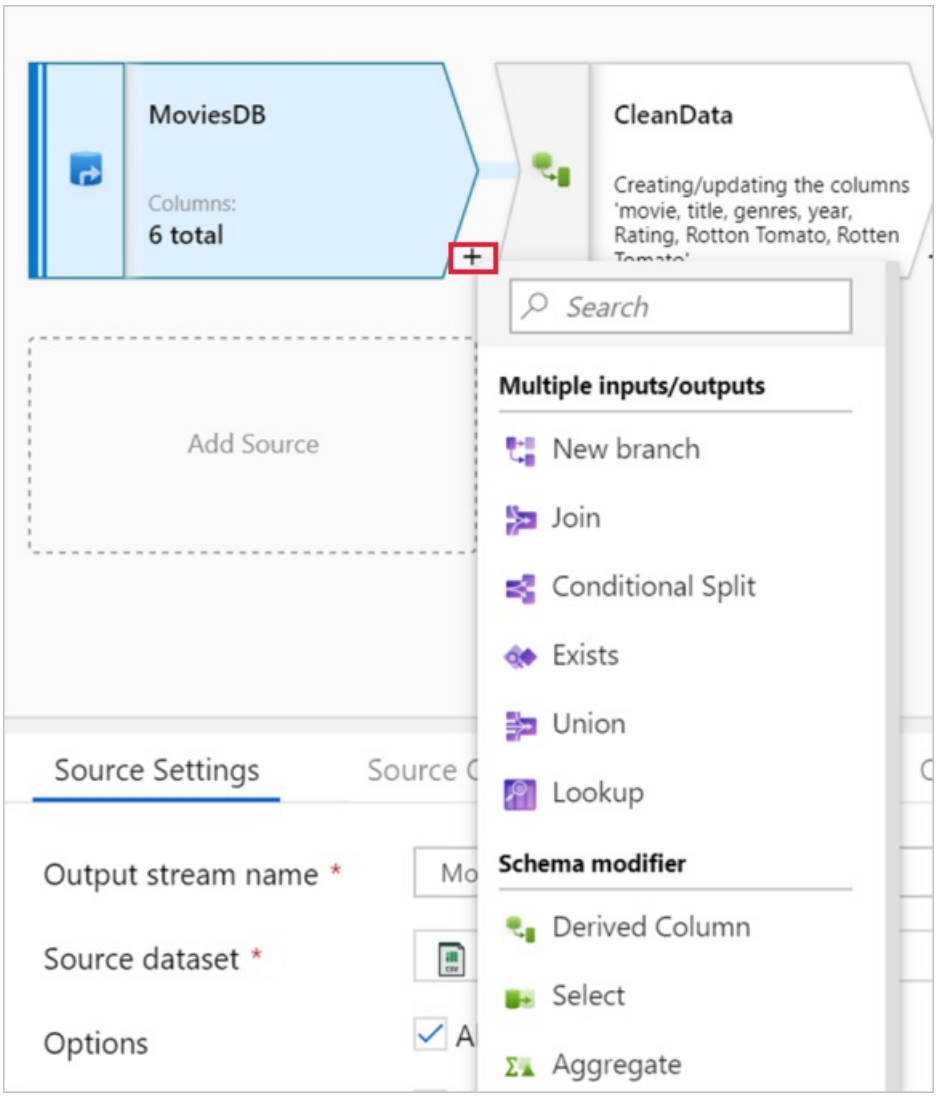
- Data Flow performance
- Data Flow activity
- Monitor Data Flow performance

Managing the mapping data flow graph

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

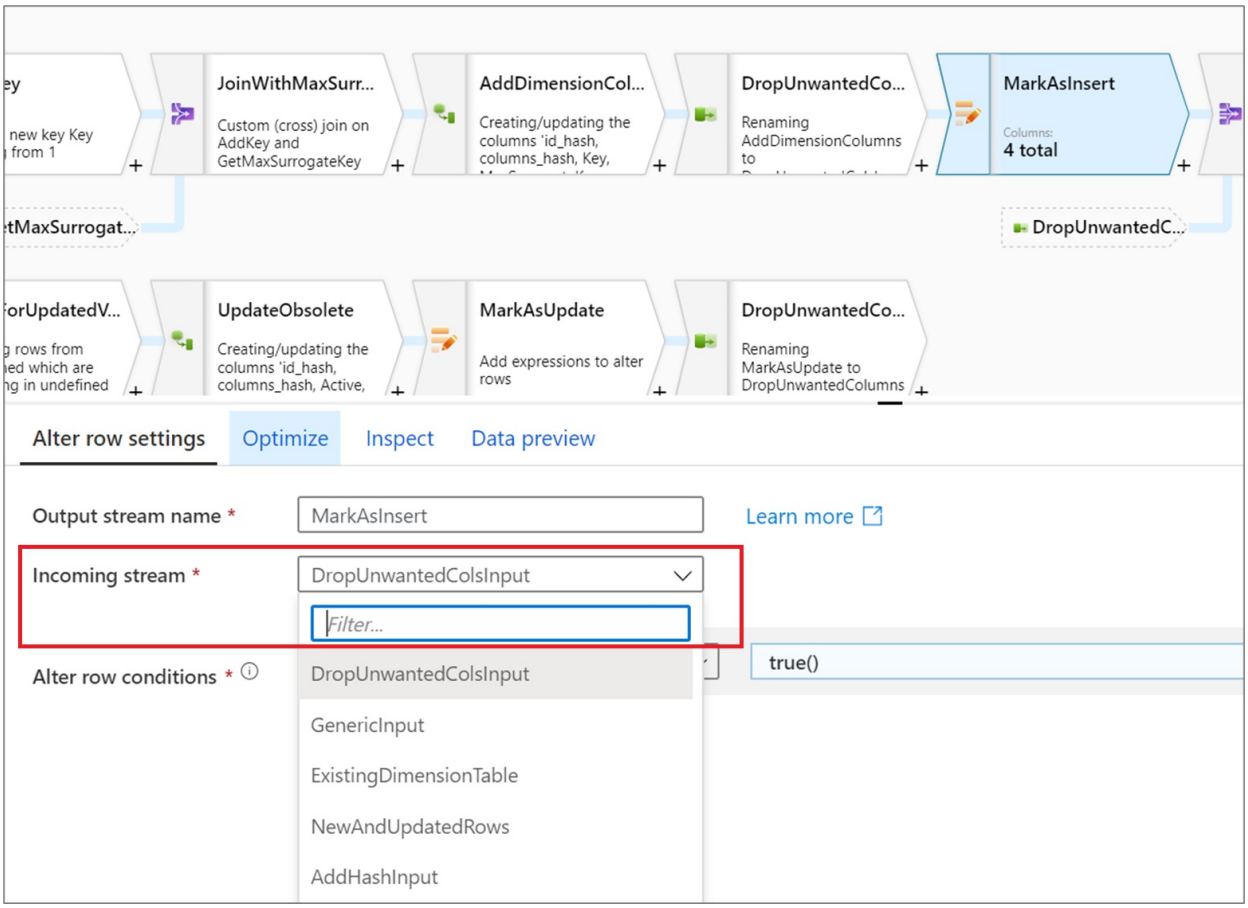
Mapping data flows are authored using a design surface know as the data flow graph. In the graph, transformation logic is built left-to-right and additional data streams are added top-down. To add a new transformation, select the plus sign on the lower right of an existing transformation.



As your data flows get more complex, use the following mechanisms to effectively navigate and manage the data flow graph.

Moving transformations

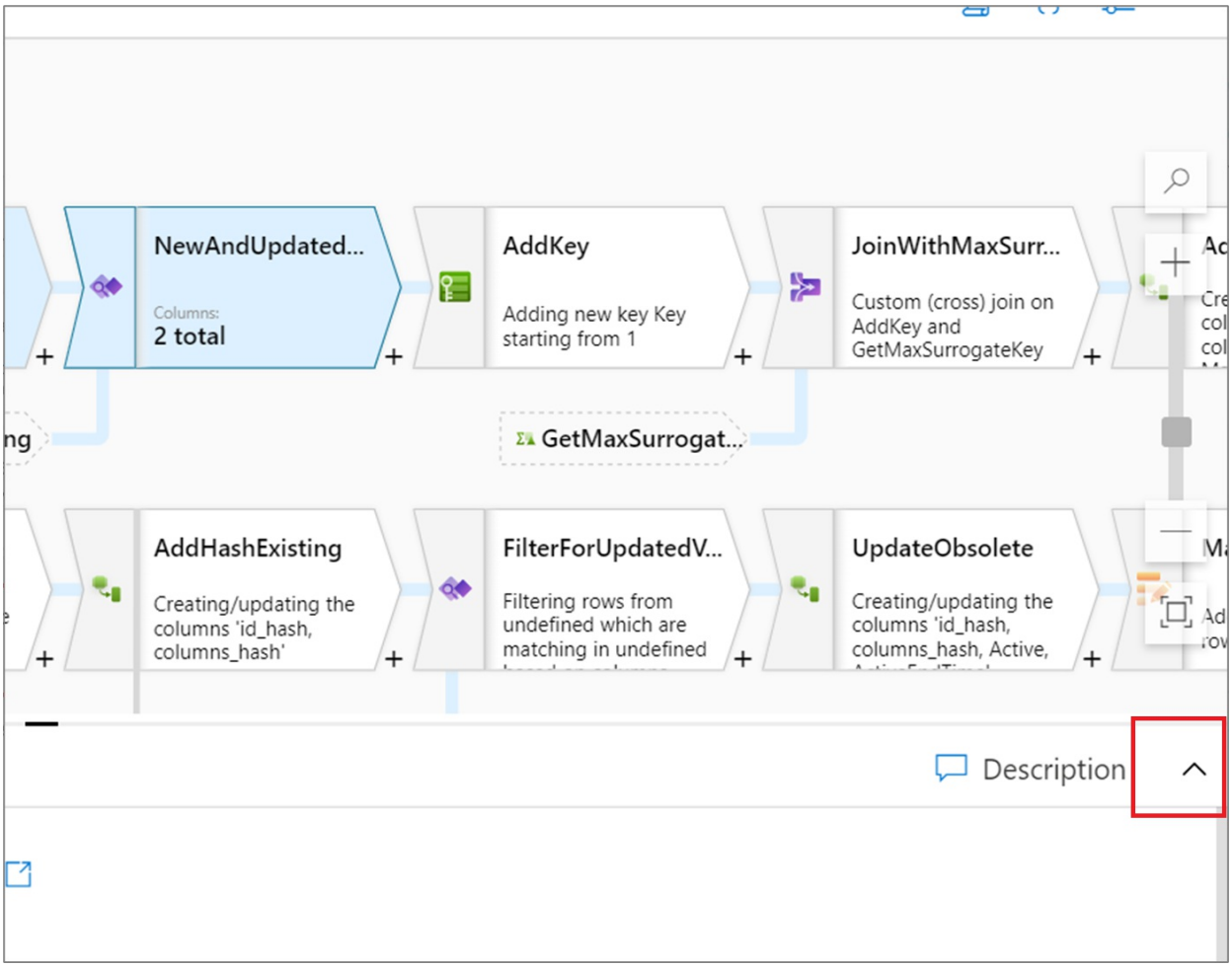
In mapping data flows, a set of connected transformation logic is known as a **stream**. The **Incoming stream** field dictates which data stream is feeding the current transformation. Each transformation has one or two incoming streams depending on its function and represents an output stream. The output schema of the incoming streams determines which column metadata can be referenced by the current transformation.



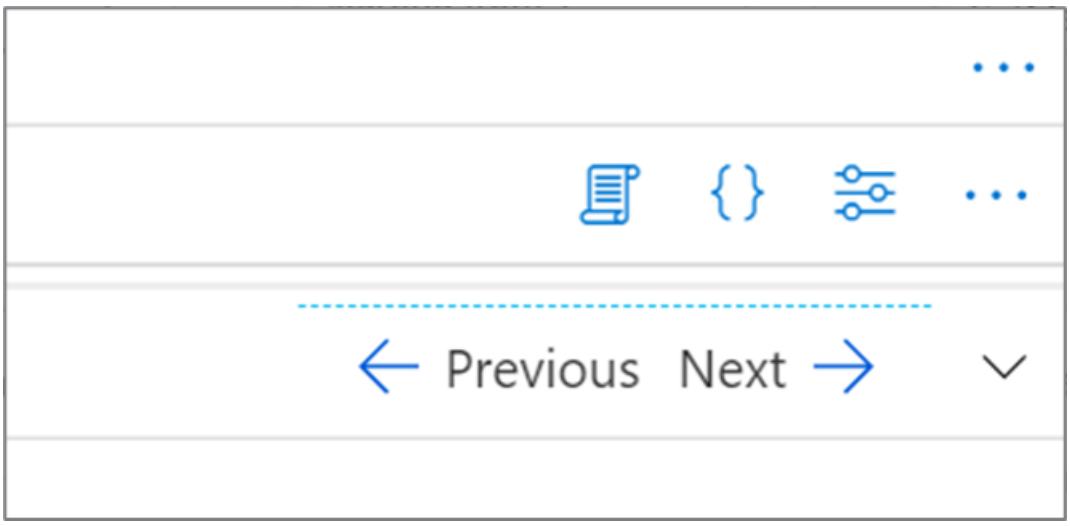
Unlike the pipeline canvas, data flow transformations aren't edited using a drag and drop model. To change the incoming stream of or "move" a transformation, choose a different value from the **Incoming stream** dropdown. When you do this, all downstream transformations will move alongside the edited transformation. The graph will automatically update to show the new logical flow. If you change the incoming stream to a transformation that already has downstream transformation, a new branch or parallel data stream will be created. Learn more about new branches in mapping data flow.

Hide graph and show graph

When editing your transformation, you can expand the configuration panel to take up the entire canvas, hiding the graph. Click on the upward-facing chevron located on the right side of the canvas.

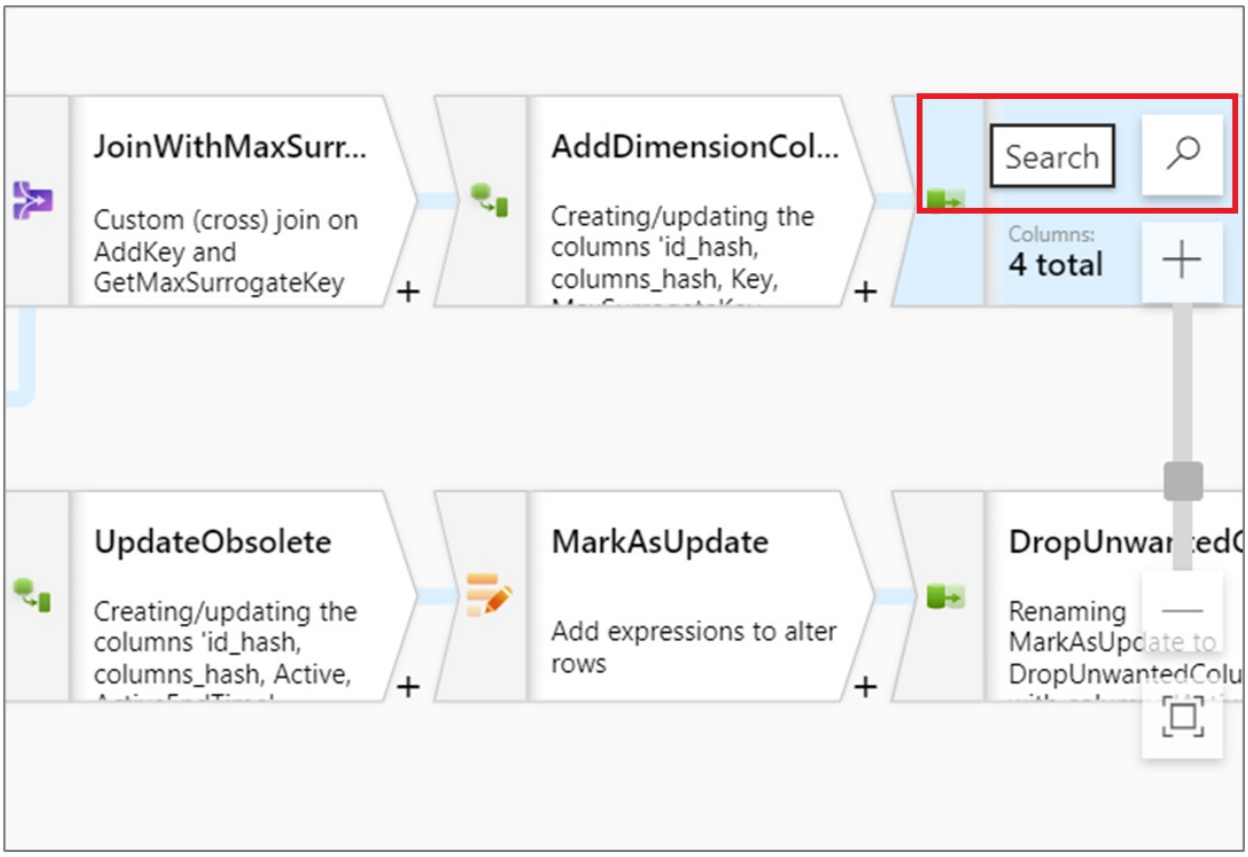


When the graph is hidden, you can move between transformations within a stream by clicking **Next** or **Previous**. Click the downward-facing chevron to show the graph.

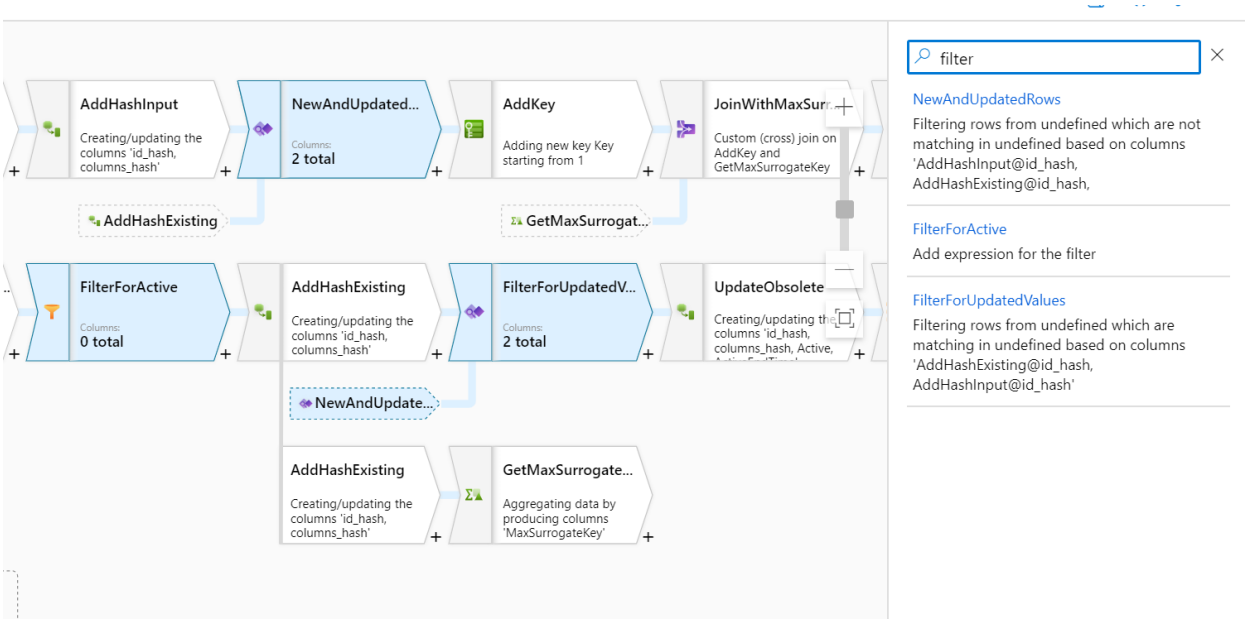


Searching for transformations

To quickly find a transformation in your graph, click on the **Search** icon above the zoom setting.

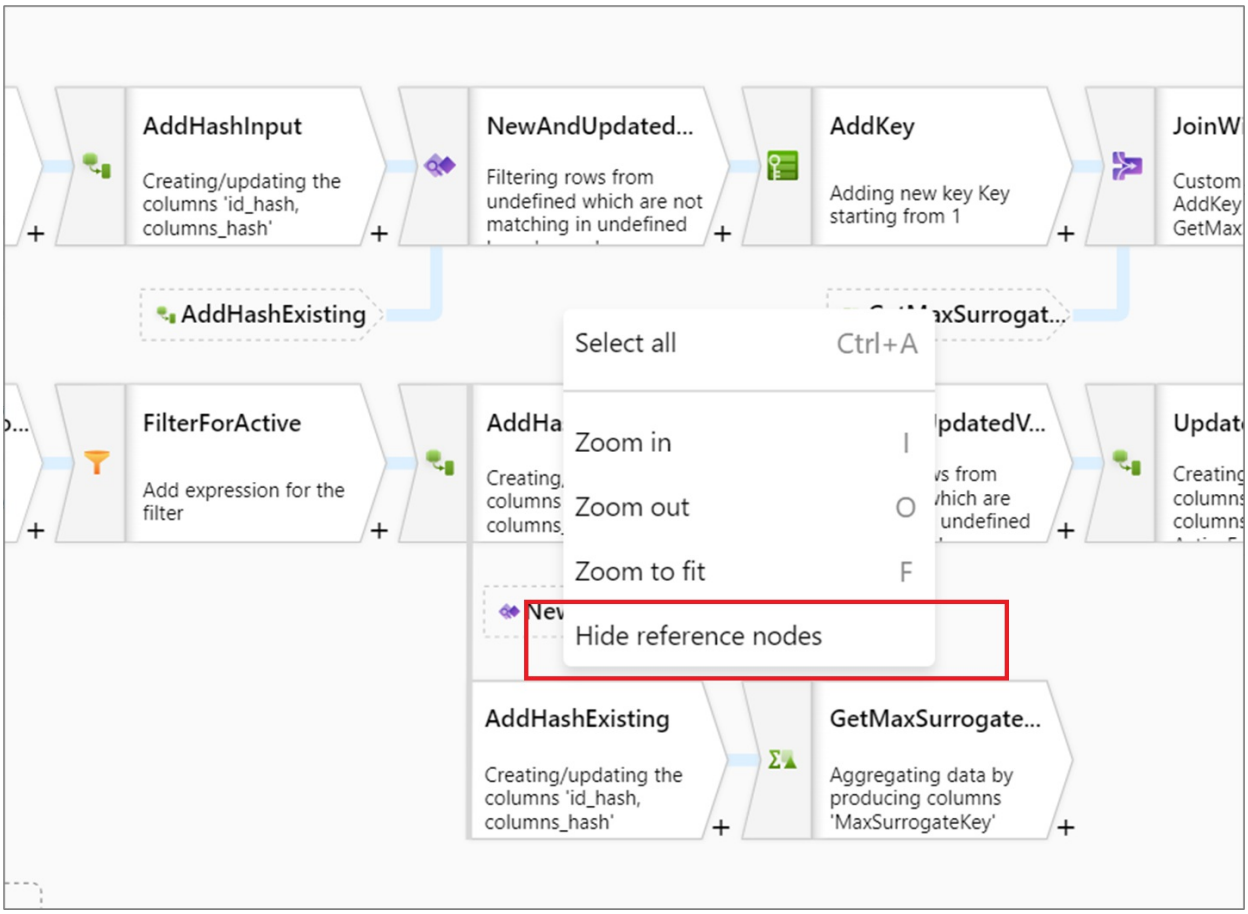


You can search by transformation name or description to locate a transformation.



Hide reference nodes

If your data flow has any join, lookup, exists, or union transformations, data flow shows reference nodes to all incoming streams. If you wish to minimize the amount of vertical space taken, you can minimize your reference nodes. To do so, right click on the canvas and select **Hide reference nodes**.



Next steps

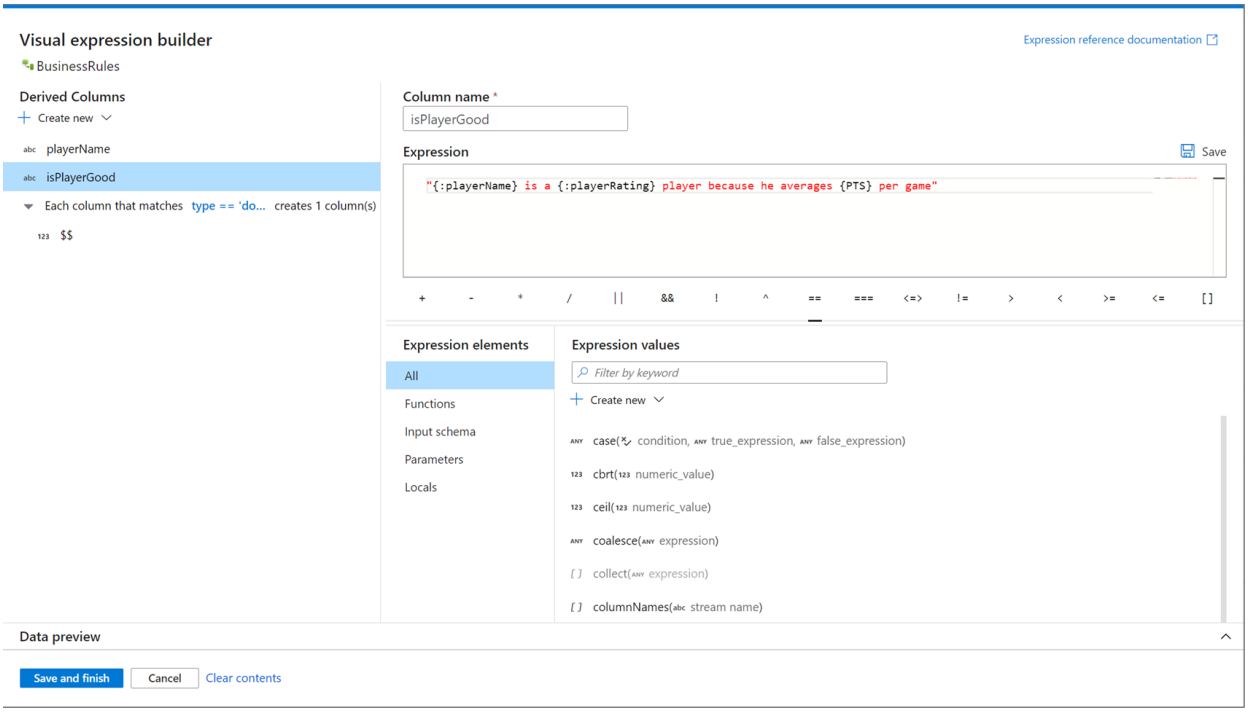
After completing your data flow logic, turn on debug mode and test it out in a data preview.

Build expressions in mapping data flow

8/19/2022 • 7 minutes to read • Edit Online

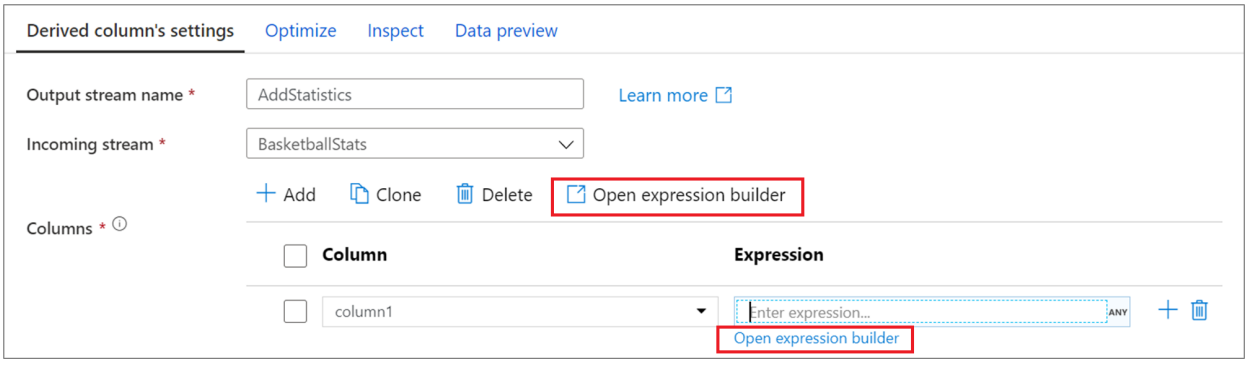
APPLIES TO: Azure Data Factory Azure Synapse Analytics

In mapping data flow, many transformation properties are entered as expressions. These expressions are composed of column values, parameters, functions, operators, and literals that evaluate to a Spark data type at run time. Mapping data flows has a dedicated experience aimed to aid you in building these expressions called the **Expression Builder**. Utilizing IntelliSense code completion for highlighting, syntax checking, and autocompleting, the expression builder is designed to make building data flows easy. This article explains how to use the expression builder to effectively build your business logic.

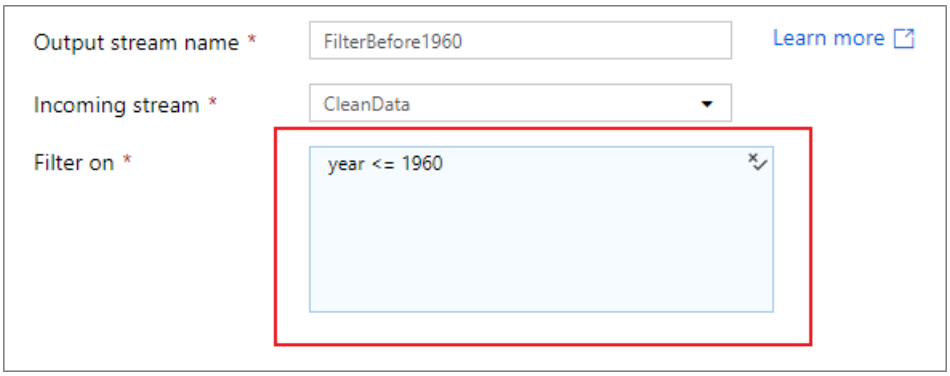


Open Expression Builder

There are multiple entry points to opening the expression builder. These are all dependent on the specific context of the data flow transformation. The most common use case is in transformations like derived column and aggregate where users create or update columns using the data flow expression language. The expression builder can be opened by selecting **Open expression builder** above the list of columns. You can also click on a column context and open the expression builder directly to that expression.

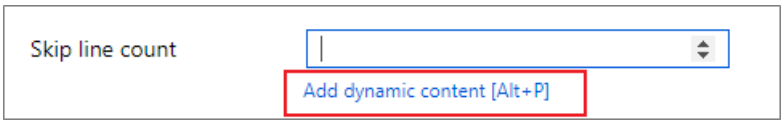


In some transformations like filter, clicking on a blue expression text box will open the expression builder.



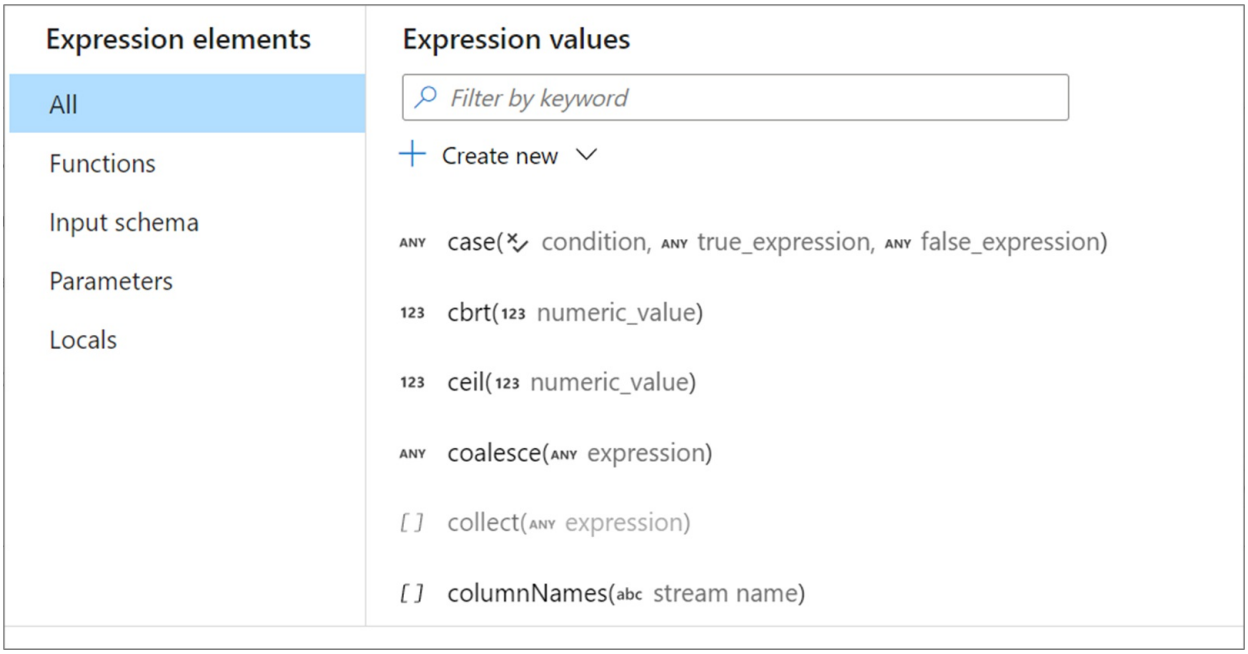
When you reference columns in a matching or group-by condition, an expression can extract values from columns. To create an expression, select **Computed column**.

In cases where an expression or a literal value are valid inputs, select **Add dynamic content** to build an expression that evaluates to a literal value.



Expression elements

In mapping data flows, expressions can be composed of column values, parameters, functions, local variables, operators, and literals. These expressions must evaluate to a Spark data type such as string, boolean, or integer.



Functions

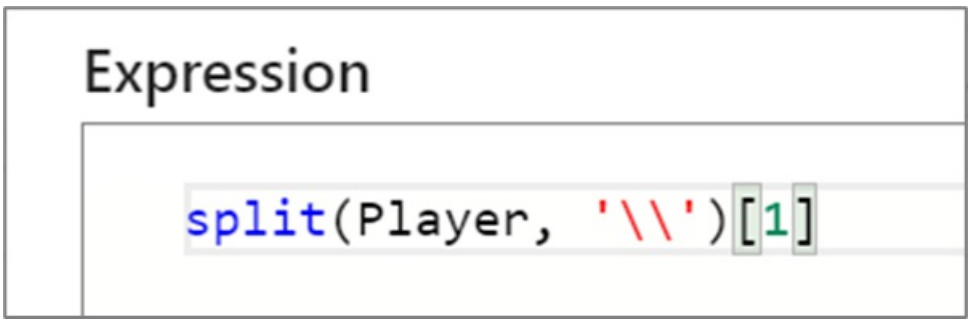
Mapping data flows has built-in functions and operators that can be used in expressions. For a list of available functions, see the mapping data flow language reference.

User Defined Functions (Preview)

Mapping data flows supports the creation and use of user defined functions. To see how to create and use user defined functions see user defined functions.

Address array indexes

When dealing with columns or functions that return array types, use brackets ([]) to access a specific element. If the index doesn't exist, the expression evaluates into NULL.



> **IMPORTANT**
> In mapping data flows, arrays are one-based meaning the first element is referenced by index one. For example, myArray[1] will access the first element of an array called 'myArray'.

Input schema

If your data flow uses a defined schema in any of its sources, you can reference a column by name in many expressions. If you are utilizing schema drift, you can reference columns explicitly using the `byName()` or `byNames()` functions or match using column patterns.

Column names with special characters

When you have column names that include special characters or spaces, surround the name with curly braces to reference them in an expression.

```
{[dbo].this_is my complex name$$$}
```

Parameters

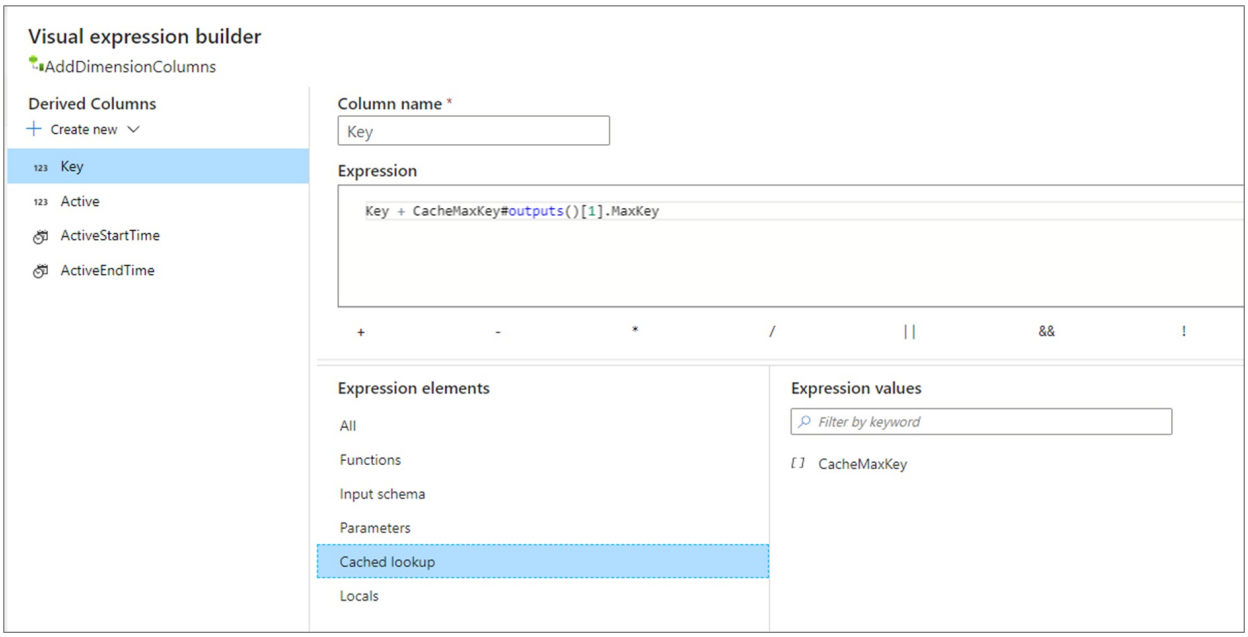
Parameters are values that are passed into a data flow at run time from a pipeline. To reference a parameter, either click on the parameter from the **Expression elements** view or reference it with a dollar sign in front of its name. For example, a parameter called parameter1 would be referenced by `$parameter1`. To learn more, see parameterizing mapping data flows.

Cached lookup

A cached lookup allows you to do an inline lookup of the output of a cached sink. There are two functions available to use on each sink, `lookup()` and `outputs()`. The syntax to reference these functions is `cacheSinkName#functionName()`. For more information, see cache sinks.

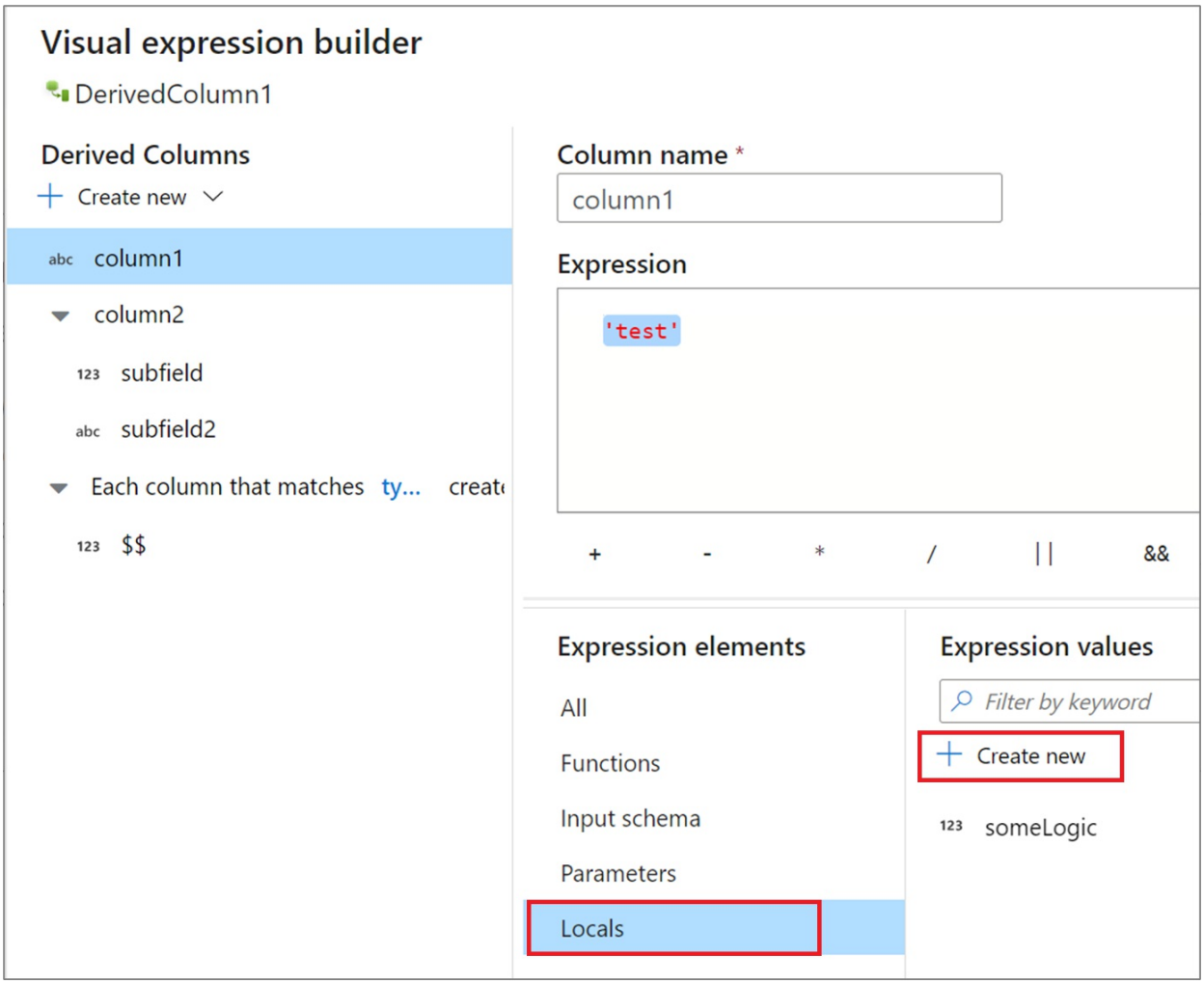
`lookup()` takes in the matching columns in the current transformation as parameters and returns a complex column equal to the row matching the key columns in the cache sink. The complex column returned contains a subcolumn for each column mapped in the cache sink. For example, if you had an error code cache sink `errorCodeCache` that had a key column matching on the code and a column called `Message`. Calling `errorCodeCache#lookup(errorCode).Message` would return the message corresponding with the code passed in.

`outputs()` takes no parameters and returns the entire cache sink as an array of complex columns. This can't be called if key columns are specified in the sink and should only be used if there is a small number of rows in the cache sink. A common use case is appending the max value of an incrementing key. If a cached single aggregated row `CacheMaxKey` contains a column `MaxKey`, you can reference the first value by calling `CacheMaxKey#outputs()[1].MaxKey`.

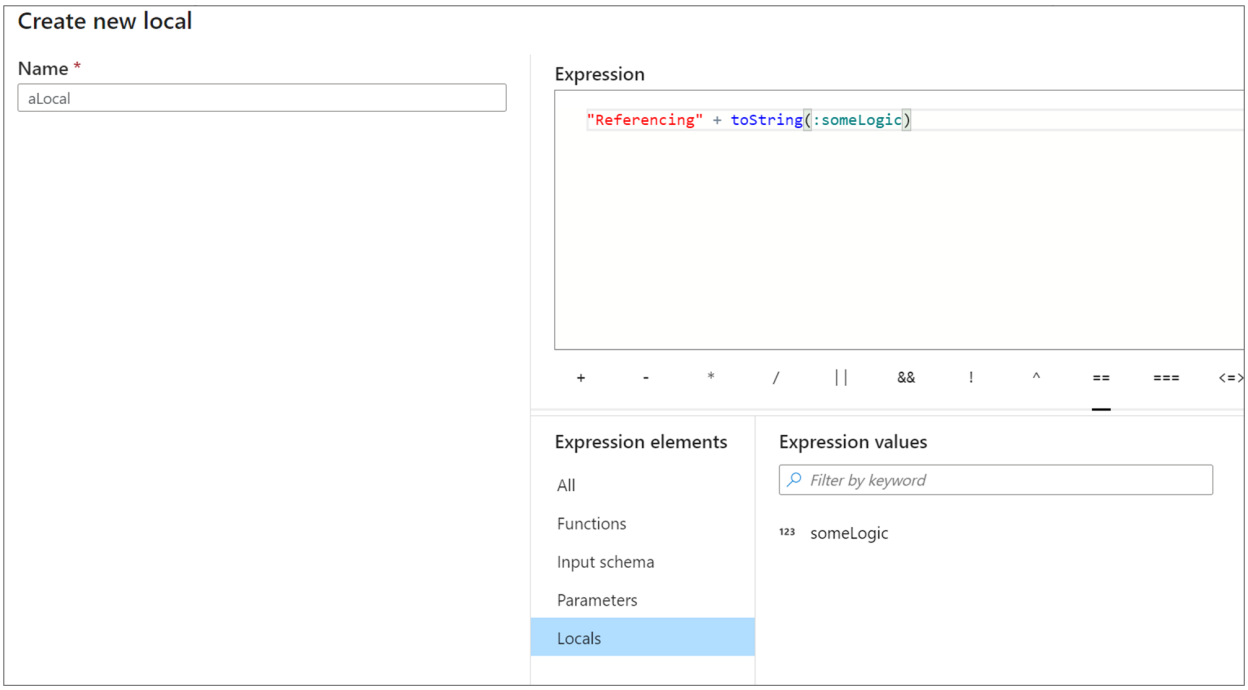


Locals

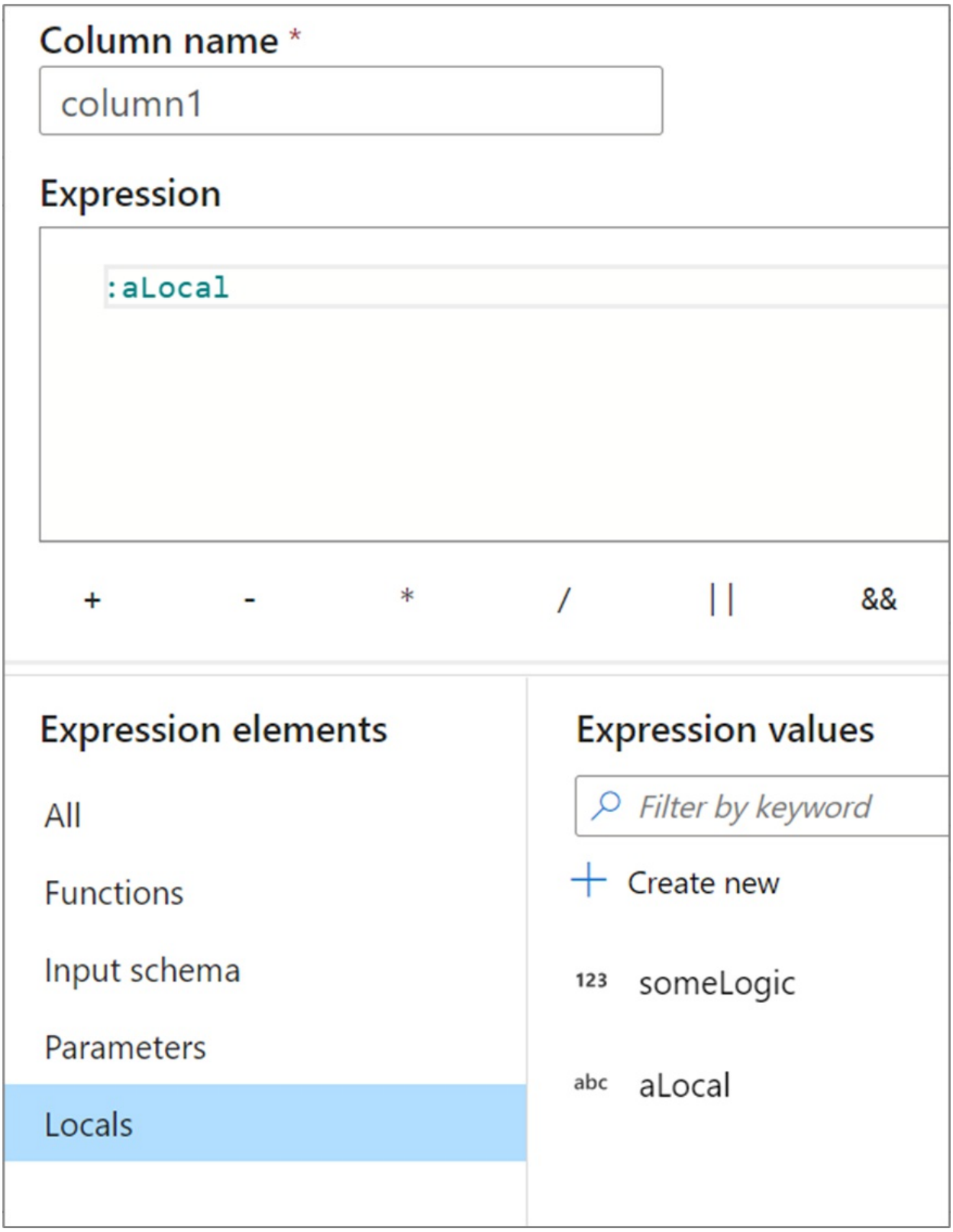
If you are sharing logic across multiple columns or want to compartmentalize your logic, you can create a local variable. A local is a set of logic that doesn't get propagated downstream to the following transformation. Locals can be created within the expression builder by going to **Expression elements** and selecting **Locals**. Create a new one by selecting **Create new**.



Locals can reference any expression element including functions, input schema, parameters, and other locals. When referencing other locals, order does matter as the referenced local needs to be "above" the current one.

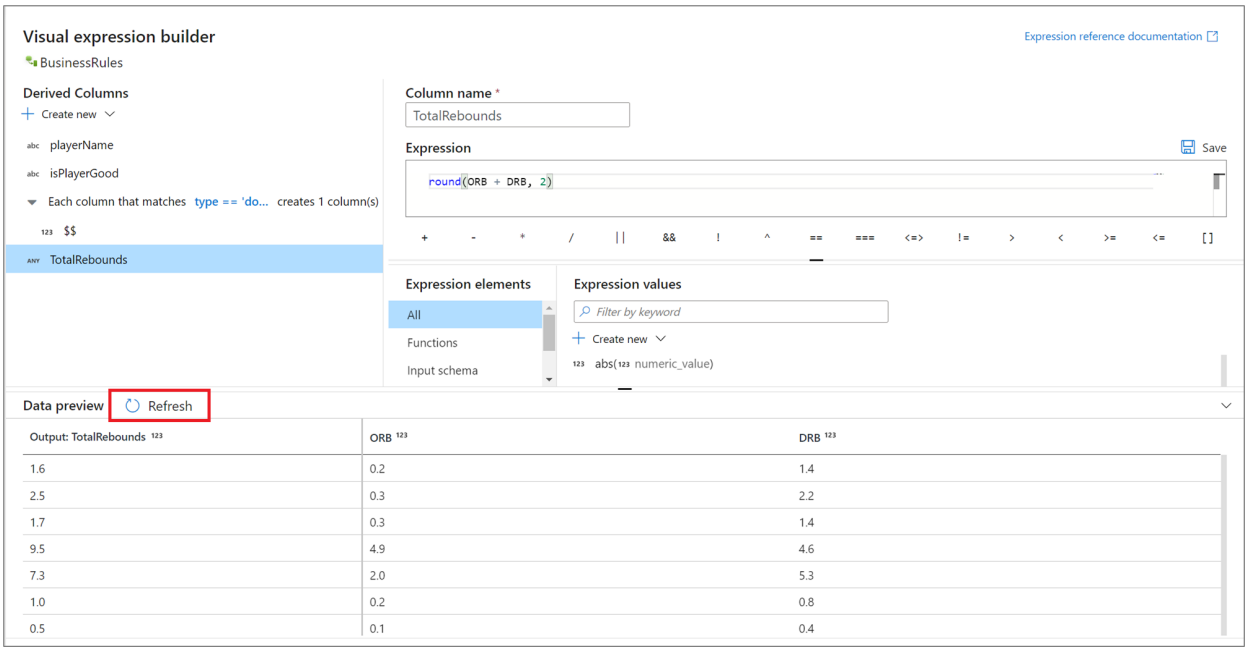


To reference a local in a transformation, either click on the local from the **Expression elements** view or reference it with a colon in front of its name. For example, a local called local1 would be referenced by `:local1`. To edit a local definition, hover over it in the expression elements view and click on the pencil icon.



Preview expression results

If debug mode is switched on, you can interactively use the debug cluster to preview what your expression evaluates to. Select **Refresh** next to data preview to update the results of the data preview. You can see the output of each row given the input columns.



String interpolation

When creating long strings that use expression elements, use string interpolation to easily build up complex string logic. String interpolation avoids extensive use of string concatenation when parameters are included in query strings. Use double quotation marks to enclose literal string text together with expressions. You can include expression functions, columns, and parameters. To use expression syntax, enclose it in curly braces,

Some examples of string interpolation:

- `"My favorite movie is {iif(instr(title,', The')>0,"The {split(title,', The')[1]}",title)}"`
- `"select * from {$tablename} where orderyear > {$year}"`
- `"Total cost with sales tax is {round(totalcost * 1.08,2)}"`
- `"{:playerName} is a {:playerRating} player"`

> **NOTE**
> When using string interpolation syntax in SQL source queries, the query string must be on one single line, without '/n'.

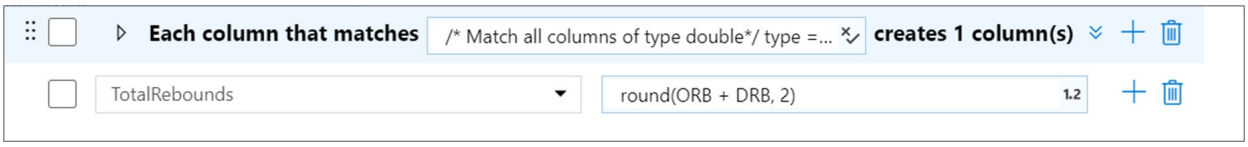
Commenting expressions

Add comments to your expressions by using single-line and multiline comment syntax.

The following examples are valid comments:

- `/* This is my comment */`
- `/* This is a`
- `multi-line comment */`

If you put a comment at the top of your expression, it appears in the transformation text box to document your transformation expressions.



Regular expressions

Many expression language functions use regular expression syntax. When you use regular expression functions, Expression Builder tries to interpret a backslash (\) as an escape character sequence. When you use backslashes in your regular expression, either enclose the entire regex in backticks (`) or use a double backslash.

An example that uses backticks:

```
regex_replace('100 and 200', `(\d+)`, 'digits')
```

An example that uses double slashes:

```
regex_replace('100 and 200', '(\\d+)', 'digits')
```

Keyboard shortcuts

Below are a list of shortcuts available in the expression builder. Most intellisense shortcuts are available when creating expressions.

- Ctrl+K Ctrl+C: Comment entire line.
- Ctrl+K Ctrl+U: Uncomment.
- F1: Provide editor help commands.
- Alt+Down arrow key: Move down current line.
- Alt+Up arrow key: Move up current line.
- Ctrl+Spacebar: Show context help.

Commonly used expressions

Convert to dates or timestamps

To include string literals in your timestamp output, wrap your conversion in `toString()`.

```
toString(toTimestamp('12/31/2016T00:12:00', 'MM/dd/yyyy\'T\'HH:mm:ss'), 'MM/dd /yyyy\'T\'HH:mm:ss')
```

To convert milliseconds from epoch to a date or timestamp, use `toTimestamp(<number of milliseconds>)`. If time is coming in seconds, multiply by 1,000.

```
toTimestamp(1574127407*1000l)
```

The trailing "l" at the end of the previous expression signifies conversion to a long type as inline syntax.

Find time from epoch or Unix Time

toLong(currentTimestamp() - toTimestamp('1970-01-01 00:00:00.000', 'yyyy-MM-dd HH:mm:ss.SSS')) * 1000l

Data flow time evaluation

Dataflow processes till milliseconds. For *2018-07-31T20:00:00.2170000*, you will see *2018-07-31T20:00:00.217* in output. In the portal for the service, timestamp is being shown in the **current browser setting**, which can eliminate 217, but when you will run the data flow end to end, 217 (milliseconds part will be processed as well). You can use toString(myDateTimeColumn) as expression and see full precision data in preview. Process datetime as datetime rather than string for all practical purposes.

Next steps

Begin building data transformation expressions

Expression functions in mapping data flow

8/19/2022 • 11 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

The following articles provide details about expression functions supported by Azure Data Factory and Azure Synapse Analytics in mapping data flows.

Expression functions list

In Data Factory and Synapse pipelines, use the expression language of the mapping data flow feature to configure data transformations.

EXPRESSION FUNCTION	TASK
abs	Absolute value of a number.
acos	Calculates a cosine inverse value.
add	Adds a pair of strings or numbers. Adds a date to a number of days. Adds a duration to a timestamp. Appends one array of similar type to another. Same as the + operator.
and	Logical AND operator. Same as &&.
asin	Calculates an inverse sine value.
assertErrorMessages	Returns map of all assert messages.
atan	Calculates an inverse tangent value.
atan2	Returns the angle in radians between the positive x-axis of a plane and the point given by the coordinates.
between	Checks if the first value is in between two other values inclusively. Numeric, string and datetime values can be compared
bitwiseAnd	Bitwise And operator across integral types. Same as & operator.
bitwiseOr	Bitwise Or operator across integral types. Same as | operator.
bitwiseXor	Bitwise Or operator across integral types. Same as | operator.

EXPRESSION FUNCTION	TASK
blake2b	Calculates the Blake2 digest of set of columns of varying primitive datatypes given a bit length. The bit length can only be multiples of 8 between 8 and 512. It can be used to calculate a fingerprint for a row.
blake2bBinary	Calculates the Blake2 digest of set of column of varying primitive datatypes given a bit length, which can only be multiples of 8 between 8 & 512. It can be used to calculate a fingerprint for a row
case	Based on alternating conditions applies one value or the other. If the number of inputs are even, the other is defaulted to NULL for last condition.
cbrt	Calculates the cube root of a number.
ceil	Returns the smallest integer not smaller than the number.
coalesce	Returns the first not null value from a set of inputs. All inputs should be of the same type.
columnNames	Gets the names of all output columns for a stream. You can pass an optional stream name as the first argument and optional second argument to only return schema drift columns.
columns	Gets the values of all output columns for a stream. You can pass an optional stream name as the second argument.
compare	Compares two values of the same type. Returns a negative integer if value1 < value2, 0 if value1 == value2, positive value if value1 > value2.
concat	Concatenates a variable number of strings together. Same as the + operator with strings.
concatWS	Concatenates a variable number of strings together with a separator. The first parameter is the separator.
cos	Calculates a cosine value.
cosh	Calculates a hyperbolic cosine of a value.
crc32	Calculates the CRC32 hash of set of column of varying primitive datatypes given a bit length. The bit length must be of values 0 (256), 224, 256, 384, or 512. It can be used to calculate a fingerprint for a row.
degrees	Converts radians to degrees.
divide	Divides pair of numbers. Same as the `/` operator.

EXPRESSION FUNCTION	TASK
dropLeft	Removes as many characters from the left of the string. If the drop requested exceeds the length of the string, an empty string is returned.
dropRight	Removes as many characters from the right of the string. If the drop requested exceeds the length of the string, an empty string is returned.
endsWith	Checks if the string ends with the supplied string.
equals	Comparison equals operator. Same as == operator.
equalsIgnoreCase	Comparison equals operator, ignoring case. Same as <=> operator.
escape	Escapes a string according to a format. Literal values for acceptable format are 'json', 'xml', 'ecmascript', 'html', 'java'.
expr	Results in an expression from a string. It is equivalent to writing the expression in a non-literal form and can be used to pass parameters as string representations.
factorial	Calculates the factorial of a number.
false	Always returns a false value. Use the function `syntax(false())` if there's a column named 'false'.
floor	Returns the largest integer not greater than the number.
fromBase64	Decodes the given base64-encoded string.
greater	Comparison greater operator. Same as > operator.
greaterOrEqual	Comparison greater than or equal operator. Same as >= operator.
greatest	Returns the greatest value among the list of values as input skipping null values. Returns null if all inputs are null.
hasColumn	Checks for a column value by name in the stream. You can pass an optional stream name as the second argument. Column names known at design time should be addressed just by their name. Computed inputs aren't supported but you can use parameter substitutions.
hasError	Checks if the assert with provided ID is marked as error.
iif	Based on a condition applies one value or the other. If other is unspecified, it's considered NULL. Both the values must be compatible(numeric, string...).
iifNull	Given two or more inputs, returns the first not null item. This function is equivalent to coalesce.

EXPRESSION FUNCTION	TASK
initCap	Converts the first letter of every word to uppercase. Words are identified as separated by whitespace.
instr	Finds the position(1 based) of the substring within a string. 0 is returned if not found.
isDelete	Checks if the row is marked for delete. For transformations taking more than one input stream you can pass the (1-based) index of the stream. The stream index should be either 1 or 2 and the default value is 1.
isError	Checks if the row is marked as error. For transformations taking more than one input stream you can pass the (1-based) index of the stream. The stream index should be either 1 or 2 and the default value is 1.
isIgnore	Checks if the row is marked to be ignored. For transformations taking more than one input stream you can pass the (1-based) index of the stream. The stream index should be either 1 or 2 and the default value is 1.
isInsert	Checks if the row is marked for insert. For transformations taking more than one input stream you can pass the (1-based) index of the stream. The stream index should be either 1 or 2 and the default value is 1.
isMatch	Checks if the row is matched at lookup. For transformations taking more than one input stream you can pass the (1-based) index of the stream. The stream index should be either 1 or 2 and the default value is 1.
isNull	Checks if the value is NULL.
isUpdate	Checks if the row is marked for update. For transformations taking more than one input stream you can pass the (1-based) index of the stream. The stream index should be either 1 or 2 and the default value is 1.
isUpsert	Checks if the row is marked for insert. For transformations taking more than one input stream you can pass the (1-based) index of the stream. The stream index should be either 1 or 2 and the default value is 1.
jaroWinkler	Gets the JaroWinkler distance between two strings.
least	Comparison lesser than or equal operator. Same as <= operator.
left	Extracts a substring start at index 1 with number of characters. Same as SUBSTRING(str, 1, n).
length	Returns the length of the string.
lesser	Comparison less operator. Same as < operator.

EXPRESSION FUNCTION	TASK
lesserOrEqual	Comparison lesser than or equal operator. Same as <= operator.
levenshtein	Gets the levenshtein distance between two strings.
like	The pattern is a string that is matched literally. The exceptions are the following special symbols: _ matches any one character in the input (similar to. in `posix` regular expressions)
locate	Finds the position(1 based) of the substring within a string starting a certain position. If the position is omitted, it's considered from the beginning of the string. 0 is returned if not found.
log	Calculates log value. An optional base can be supplied else a Euler number if used.
log10	Calculates log value based on 10 base.
lower	Lowercases a string.
lpad	Left pads the string by the supplied padding until it is of a certain length. If the string is equal to or greater than the length, then it's trimmed to the length.
ltrim	Left trims a string of leading characters. If second parameter is unspecified, it trims whitespace. Else it trims any character specified in the second parameter.
md5	Calculates the MD5 digest of set of column of varying primitive datatypes and returns a 32-character hex string. It can be used to calculate a fingerprint for a row.
minus	Subtracts numbers. Subtract number of days from a date. Subtract duration from a timestamp. Subtract two timestamps to get difference in milliseconds. Same as the - operator.
mod	Modulus of pair of numbers. Same as the % operator.
multiply	Multiplies pair of numbers. Same as the * operator.
negate	Negates a number. Turns positive numbers to negative and vice versa.
nextSequence	Returns the next unique sequence. The number is consecutive only within a partition and is prefixed by the partitionId.
normalize	Normalizes the string value to separate accented unicode characters.
not	Logical negation operator.

EXPRESSION FUNCTION	TASK
notEquals	Comparison not equals operator. Same as != operator.
notNull	Checks if the value isn't NULL.
null	Returns a NULL value. Use the function `syntax(null())` if there's a column named 'null'. Any operation that uses will result in a NULL.
or	Logical OR operator. Same as ||.
pMod	Positive Modulus of pair of numbers.
partitionId	Returns the current partition ID the input row is in.
power	Raises one number to the power of another.
radians	Converts degrees to radians
random	Returns a random number given an optional seed within a partition. The seed should be a fixed value and is used with the partitionId to produce random values
regexExtract	Extract a matching substring for a given regex pattern. The last parameter identifies the match group and is defaulted to 1 if omitted. Use `` `<regex>` `` (back quote) to match a string without escaping.
regexMatch	Checks if the string matches the given regex pattern. Use `` `<regex>` `` (back quote) to match a string without escaping.
regexReplace	Replace all occurrences of a regex pattern with another substring in the given string Use `` `<regex>` `` (back quote) to match a string without escaping.
regexSplit	Splits a string based on a delimiter based on regex and returns an array of strings.
replace	Replace all occurrences of a substring with another substring in the given string. If the last parameter is omitted, it's default to empty string.
reverse	Reverses a string.
right	Extracts a substring with number of characters from the right. Same as SUBSTRING(str, LENGTH(str) - n, n).
rlike	Checks if the string matches the given regex pattern.
round	Rounds a number given an optional scale and an optional rounding mode. If the scale is omitted, it's defaulted to 0. If the mode is omitted, it's defaulted to ROUND_HALF_UP(5). The values for rounding include

EXPRESSION FUNCTION	TASK
rpad	Right pads the string by the supplied padding until it is of a certain length. If the string is equal to or greater than the length, then it's trimmed to the length.
rtrim	Right trims a string of trailing characters. If second parameter is unspecified, it trims whitespace. Else it trims any character specified in the second parameter.
sha1	Calculates the SHA-1 digest of set of column of varying primitive datatypes and returns a 40 character hex string. It can be used to calculate a fingerprint for a row.
sha2	Calculates the SHA-2 digest of set of column of varying primitive datatypes given a bit length, which can only be of values 0(256), 224, 256, 384, 512. It can be used to calculate a fingerprint for a row.
sin	Calculates a sine value.
sinh	Calculates a hyperbolic sine value.
soundex	Gets the `soundex` code for the string.
split	Splits a string based on a delimiter and returns an array of strings.
sqrt	Calculates the square root of a number.
startsWith	Checks if the string starts with the supplied string.
substring	Extracts a substring of a certain length from a position. Position is 1 based. If the length is omitted, it's defaulted to end of the string.
substringIndex	Extracts the substring before `count` occurrences of the delimiter. If `count` is positive, everything to the left of the final delimiter (counting from the left) is returned. If `count` is negative, everything to the right of the final delimiter (counting from the right) is returned.
tan	Calculates a tangent value.
tanh	Calculates a hyperbolic tangent value.
translate	Replace one set of characters by another set of characters in the string. Characters have 1 to 1 replacement.
trim	Trims a string of leading and trailing characters. If second parameter is unspecified, it trims whitespace. Else it trims any character specified in the second parameter.
true	Always returns a true value. Use the function `syntax(true())` if there's a column named 'true'.

EXPRESSION FUNCTION	TASK
typeMatch	Matches the type of the column. Can only be used in pattern expressions.number matches short, integer, long, double, float or decimal, integral matches short, integer, long, fractional matches double, float, decimal and datetime matches date or timestamp type.
unescape	Unescapes a string according to a format. Literal values for acceptable format are 'json', 'xml', 'ecmascript', 'html', 'java'.
upper	Uppercases a string.
uuid	Returns the generated UUID.
xor	Logical XOR operator. Same as ^ operator.

Next steps

- List of all aggregate functions.
- List of all array functions.
- List of all cached lookup functions.
- List of all conversion functions.
- List of all date and time functions.
- List of all map functions.
- List of all metafunctions.
- List of all window functions.
- Usage details of all data transformation expressions.
- Learn how to use Expression Builder.

Visual authoring in Azure Data Factory

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

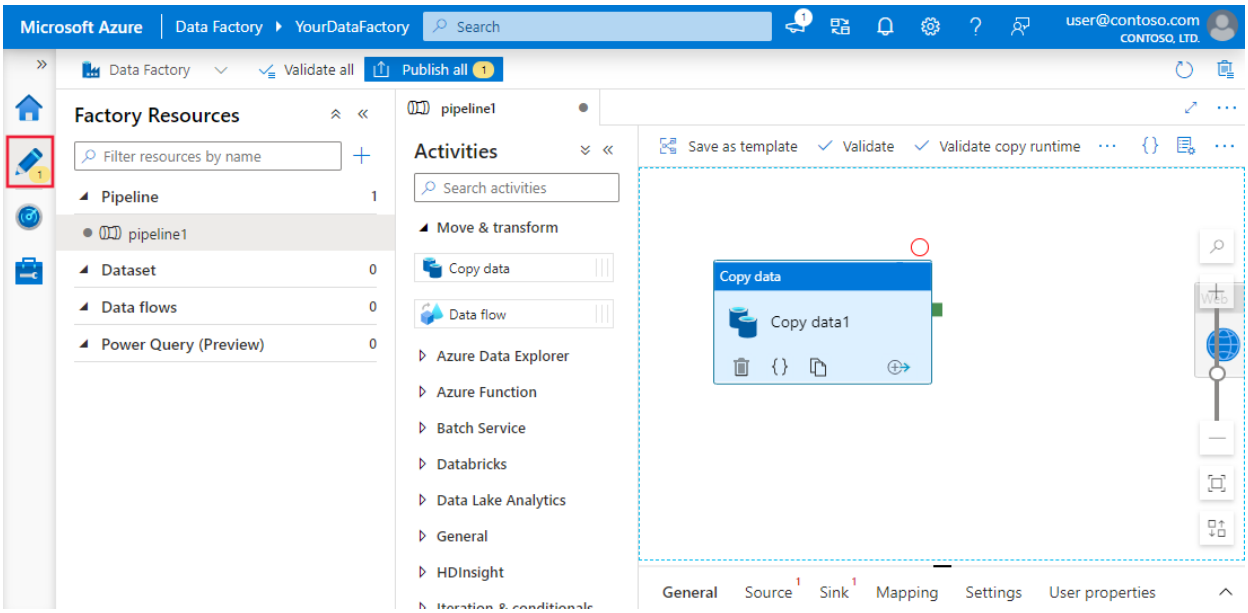
The Azure Data Factory and Synapse Analytics user interface (UI) experience lets you visually author and deploy resources for your data factory or Synapse pipelines without having to write any code. You can drag activities to a pipeline canvas, perform test runs, debug iteratively, and deploy and monitor your pipeline runs.

Currently, the UI is only supported in Microsoft Edge and Google Chrome.

Authoring canvas

- Azure Data Factory
- Synapse Analytics

To open the **authoring canvas**, click on the pencil icon.

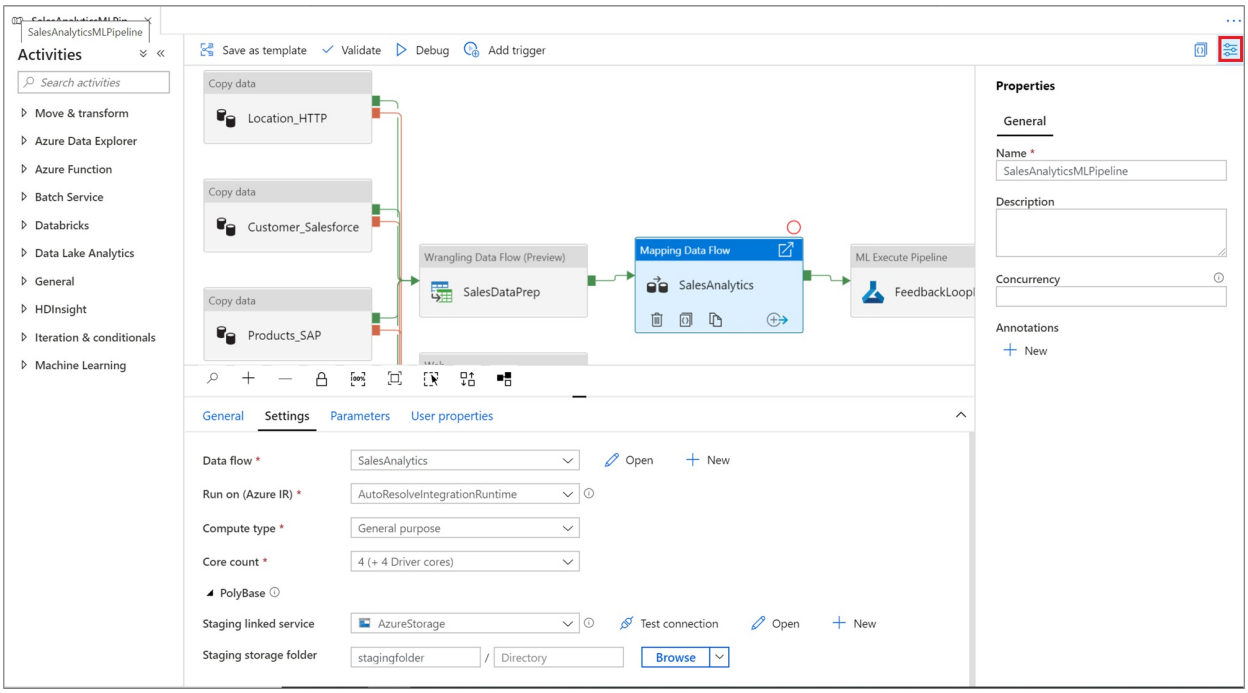


Here, you author the pipelines, activities, datasets, and data flows that comprise your factory. Similarly, linked services, triggers and integration runtimes can be edited on the Manage tab. To get started building a pipeline using the authoring canvas, see Copy data using the copy Activity.

The default visual authoring experience is directly working with the service. Azure Repos Git or GitHub integration is also supported to allow source control and collaboration for work on your pipelines. To learn more about the differences between these authoring experiences, see Source control.

Properties pane

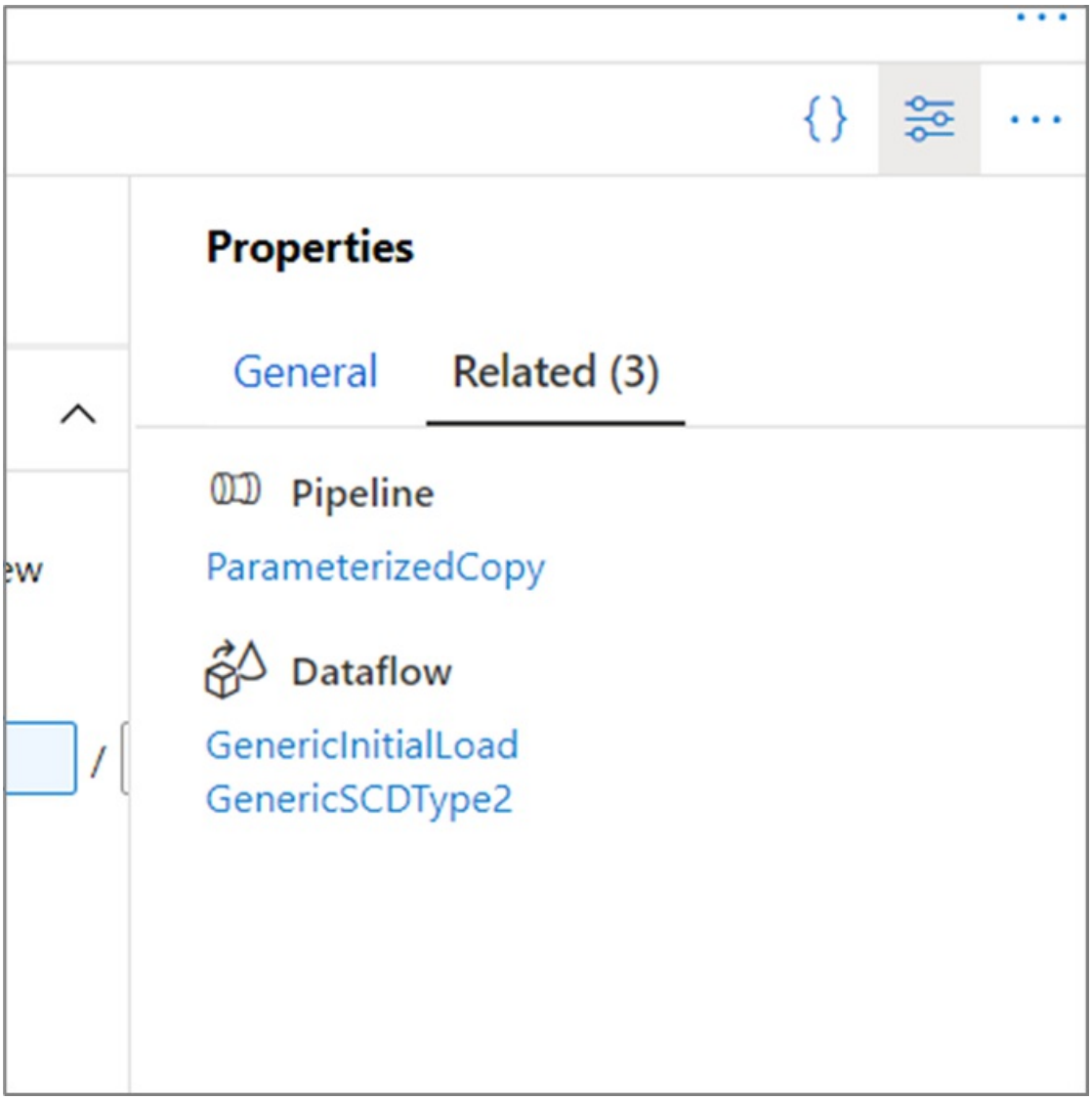
For top-level resources such as pipelines, datasets, and data flows, high-level properties are editable in the properties pane on the right-hand side of the canvas. The properties pane contains properties such as name, description, annotations, and other high-level properties. Subresources such as pipeline activities and data flow transformations are edited using the panel at the bottom of the canvas.



The properties pane only opens by default on resource creation. To edit it, click on the properties pane icon located in the top-right corner of the canvas.

Related resources

In the properties pane, you can see what resources are dependent on the selected resource by selecting the **Related** tab. Any resource that references the current resource will be listed here.

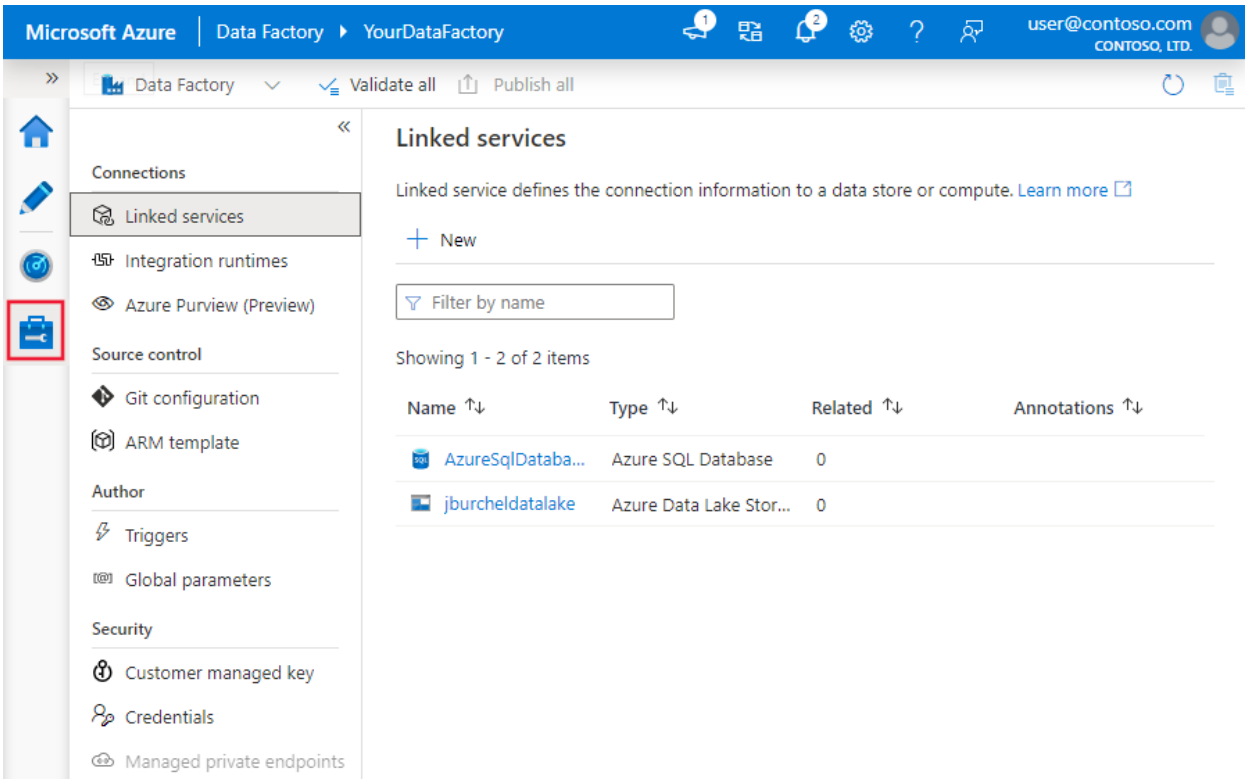


For example, in the above image, one pipeline and two data flows use the dataset currently selected.

Management hub

The management hub, accessed by the *Manage* tab in the UI, is a portal that hosts global management actions for the service. Here, you can manage your connections to data stores and external computes, source control configuration, and trigger settings. For more information, learn about the capabilities of the management hub.

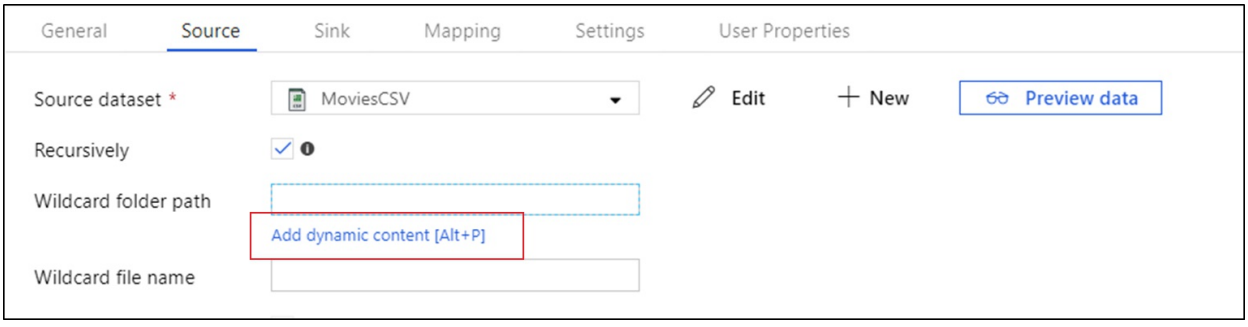
- Azure Data Factory
- Synapse Analytics



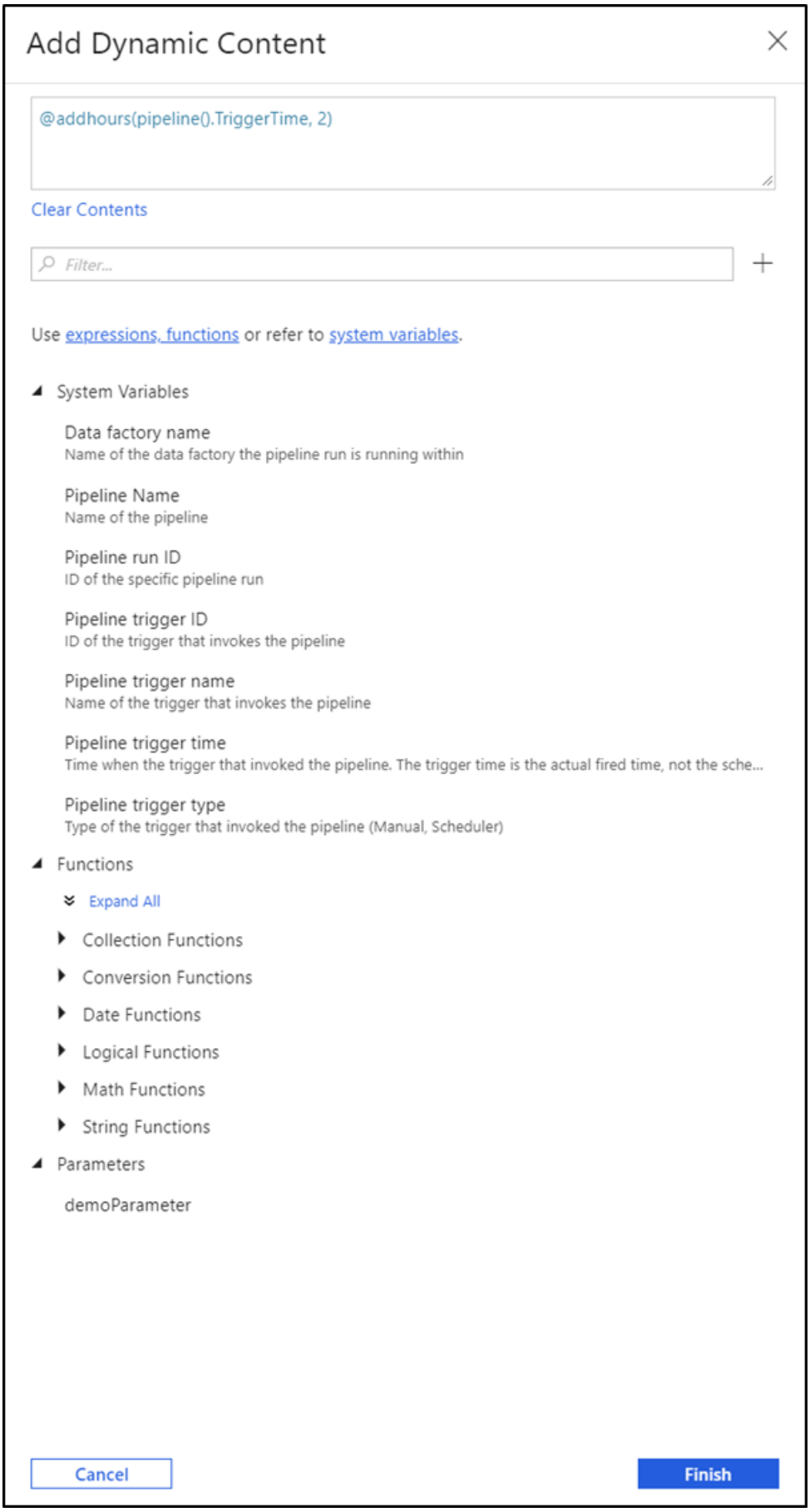
Expressions and functions

Expressions and functions can be used instead of static values to specify many properties within the service.

To specify an expression for a property value, select **Add Dynamic Content** or click **Alt + P** while focusing on the field.



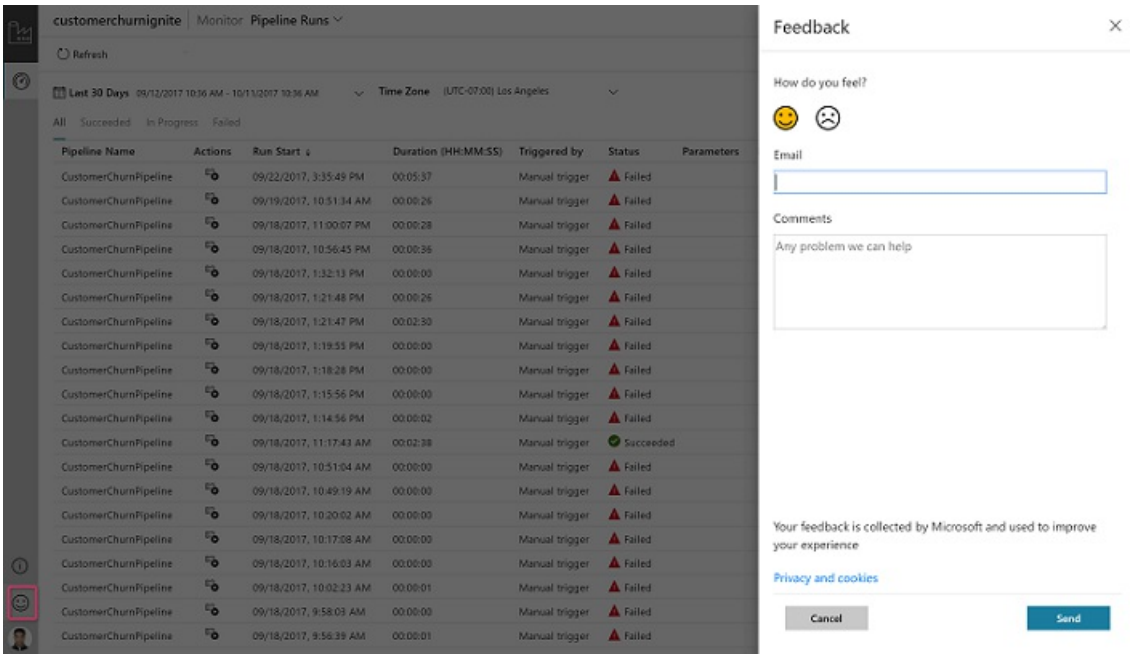
This opens the **Expression Builder** where you can build expressions from supported system variables, activity output, functions, and user-specified variables or parameters.



For information about the expression language, see Expressions and functions.

Provide feedback

Select **Feedback** to comment about features or to notify Microsoft about issues with the tool:



Next steps

To learn more about monitoring and managing pipelines, see Monitor and manage pipelines programmatically.

Iterative development and debugging with Azure Data Factory and Synapse Analytics pipelines

8/19/2022 • 4 minutes to read • Edit Online

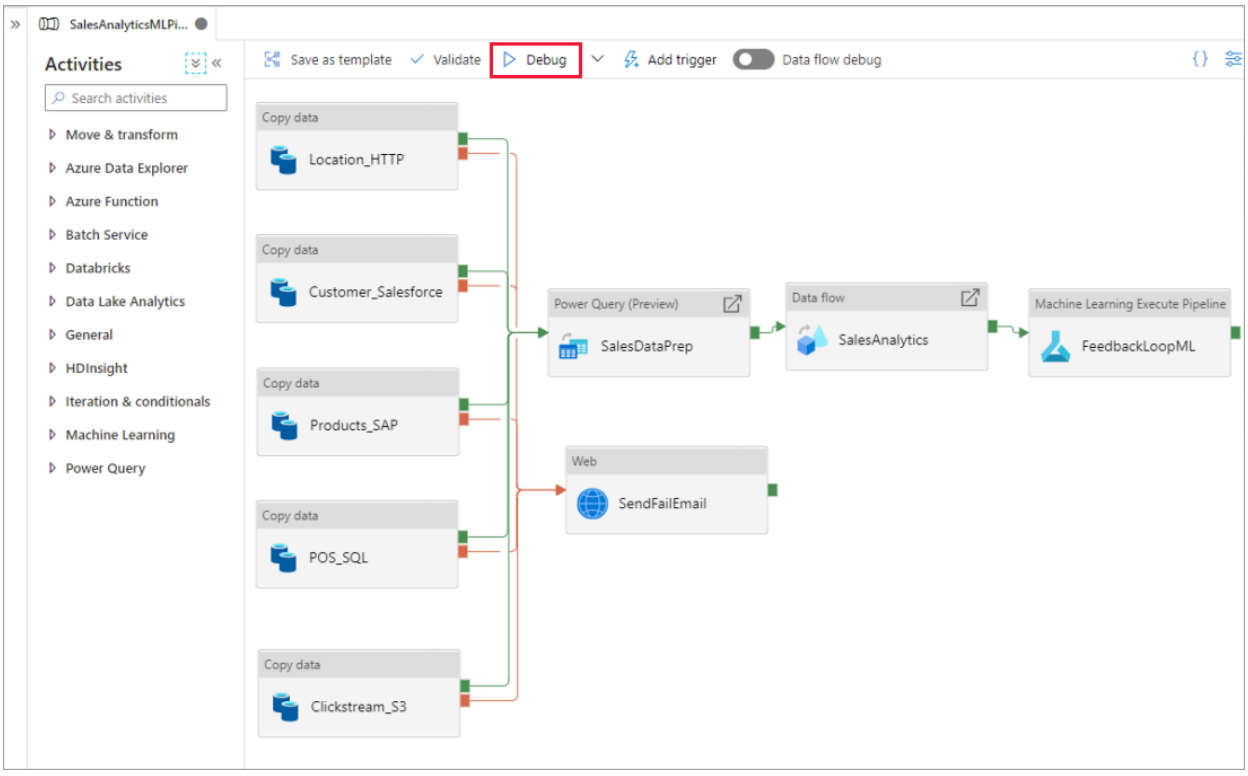
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Azure Data Factory and Synapse Analytics supports iterative development and debugging of pipelines. These features allow you to test your changes before creating a pull request or publishing them to the service.

For an eight-minute introduction and demonstration of this feature, watch the following video:

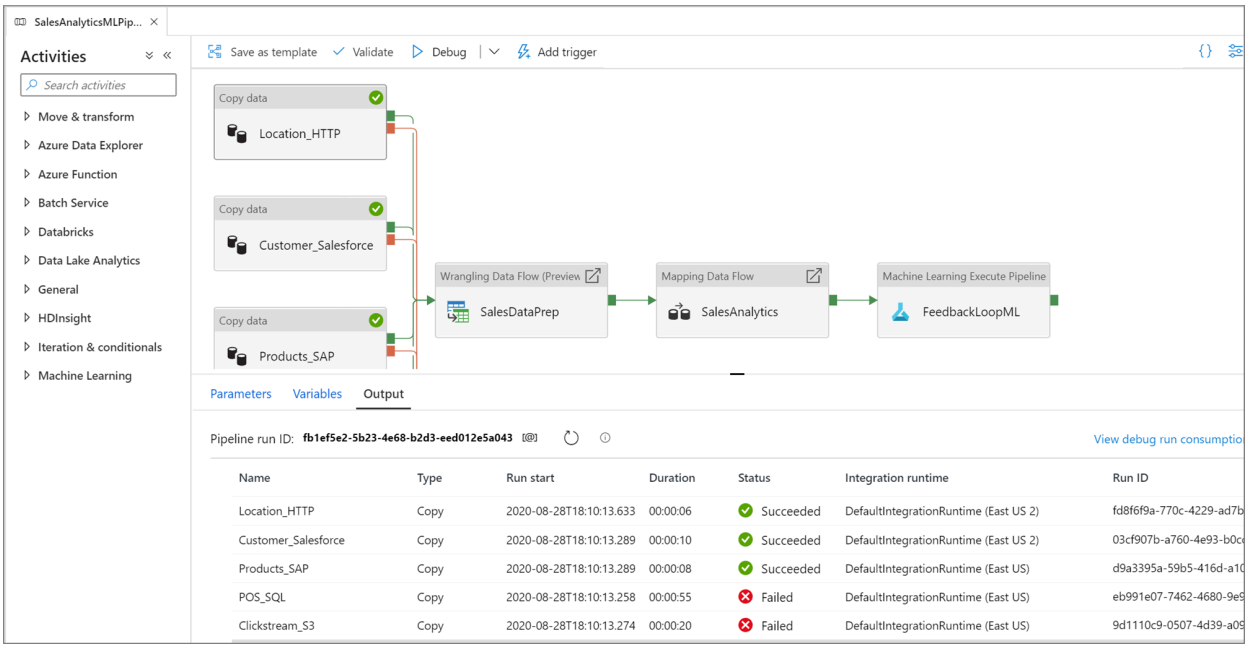
Debugging a pipeline

As you author using the pipeline canvas, you can test your activities using the **Debug** capability. When you do test runs, you don't have to publish your changes to the service before you select **Debug**. This feature is helpful in scenarios where you want to make sure that the changes work as expected before you update the workflow.



As the pipeline is running, you can see the results of each activity in the **Output** tab of the pipeline canvas.

View the results of your test runs in the **Output** window of the pipeline canvas.

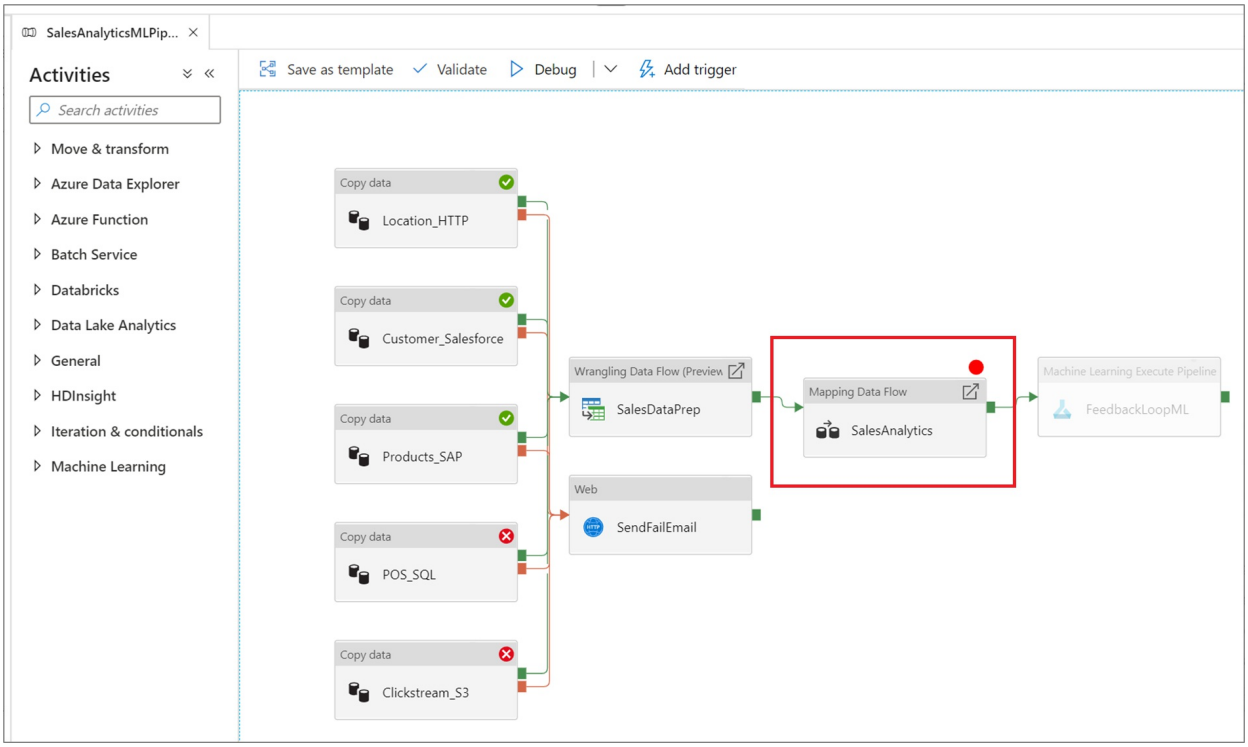


After a test run succeeds, add more activities to your pipeline and continue debugging in an iterative manner. You can also **Cancel** a test run while it is in progress.

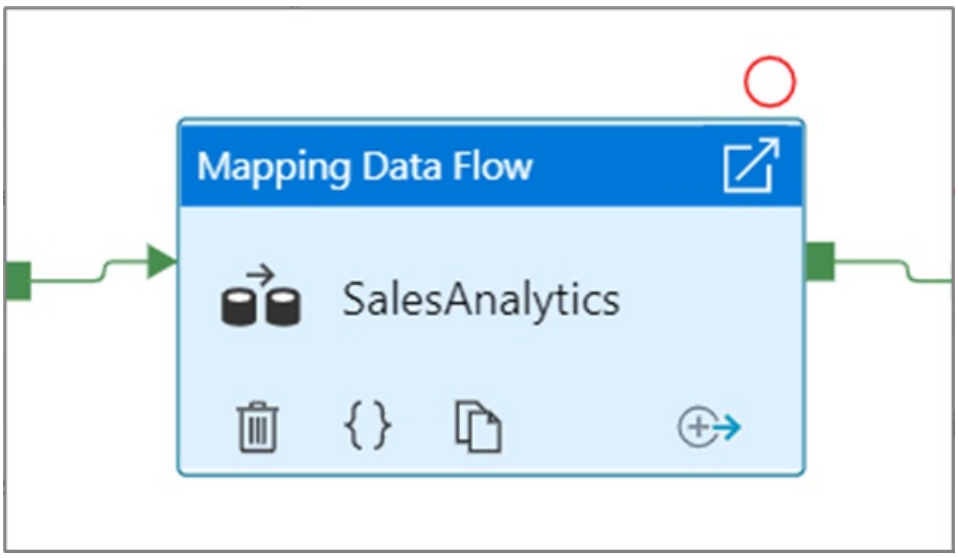
> **IMPORTANT**
>
> Selecting **Debug** actually runs the pipeline. For example, if the pipeline contains copy activity, the test run copies data from source to destination. As a result, we recommend that you use test folders in your copy activities and other activities when debugging. After you've debugged the pipeline, switch to the actual folders that you want to use in normal operations.

Setting breakpoints

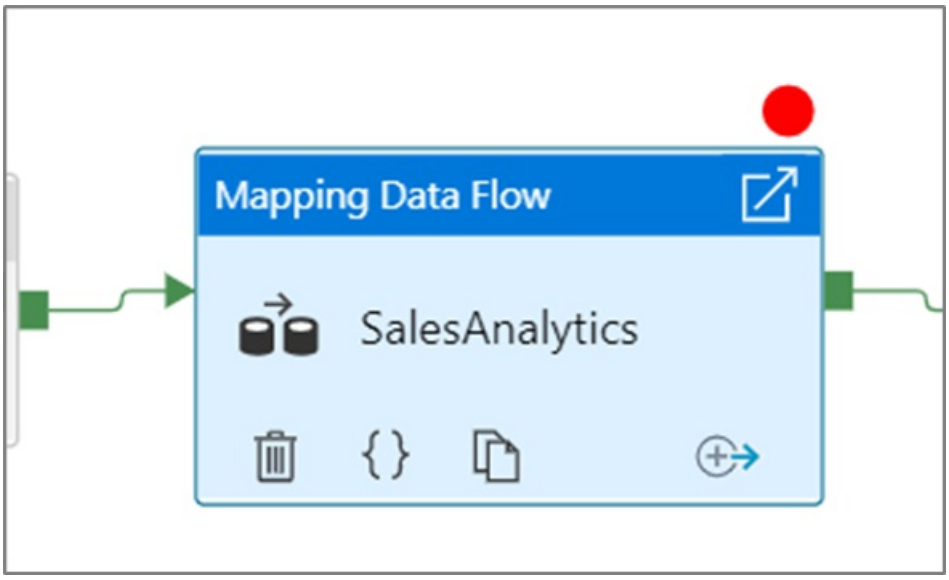
The service allows for you to debug a pipeline until you reach a particular activity on the pipeline canvas. Put a breakpoint on the activity until which you want to test, and select **Debug**. The service ensures that the test runs only until the breakpoint activity on the pipeline canvas. This *Debug Until* feature is useful when you don't want to test the entire pipeline, but only a subset of activities inside the pipeline.



To set a breakpoint, select an element on the pipeline canvas. A *Debug Until* option appears as an empty red circle at the upper right corner of the element.

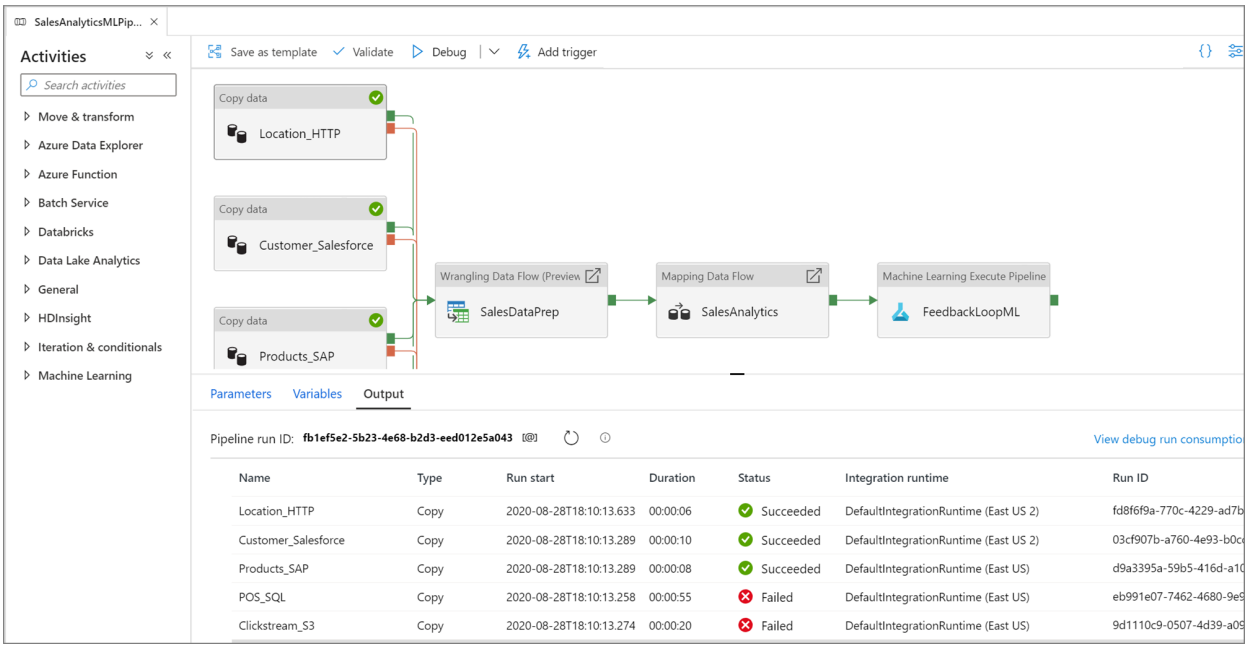


After you select the *Debug Until* option, it changes to a filled red circle to indicate the breakpoint is enabled.



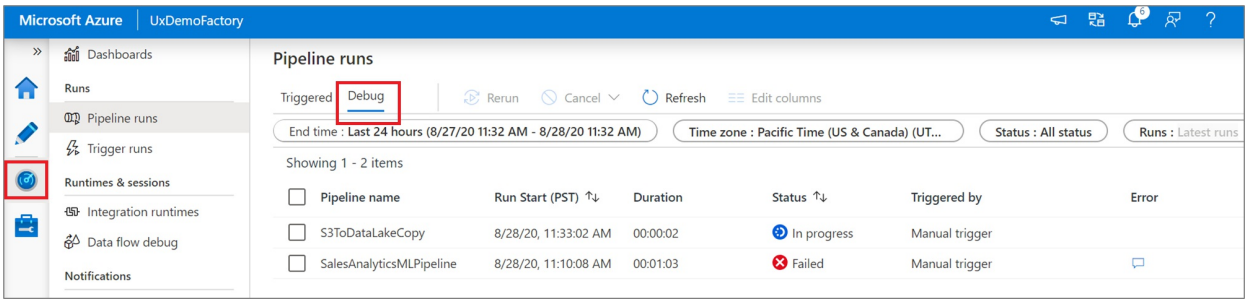
Monitoring debug runs

When you run a pipeline debug run, the results will appear in the **Output** window of the pipeline canvas. The output tab will only contain the most recent run that occurred during the current browser session.



To view a historical view of debug runs or see a list of all active debug runs, you can go into the **Monitor** experience.

- Azure Data Factory
- Synapse Analytics

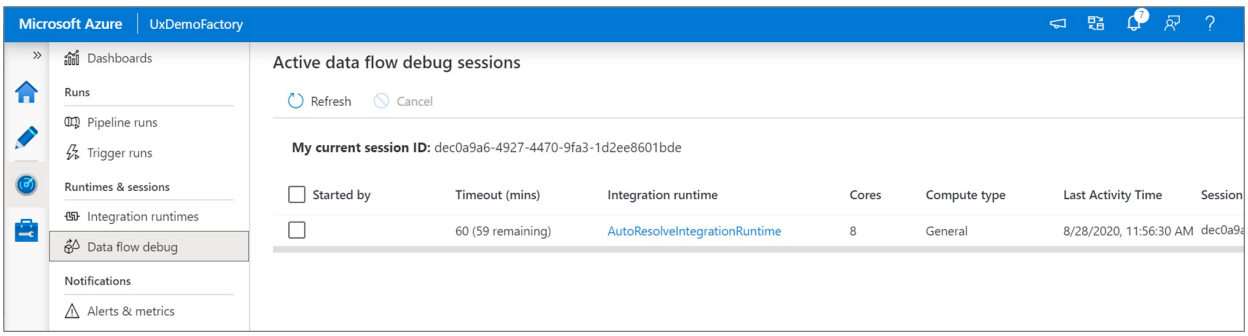


> **NOTE**
> The service only persists debug run history for 15 days.

Debugging mapping data flows

Mapping data flows allow you to build code-free data transformation logic that runs at scale. When building your logic, you can turn on a debug session to interactively work with your data using a live Spark cluster. To learn more, read about mapping data flow debug mode.

You can monitor active data flow debug sessions in the **Monitor** experience.



Data preview in the data flow designer and pipeline debugging of data flows are intended to work best with

small samples of data. However, if you need to test your logic in a pipeline or data flow against large amounts of data, increase the size of the Azure Integration Runtime being used in the debug session with more cores and a minimum of general purpose compute.

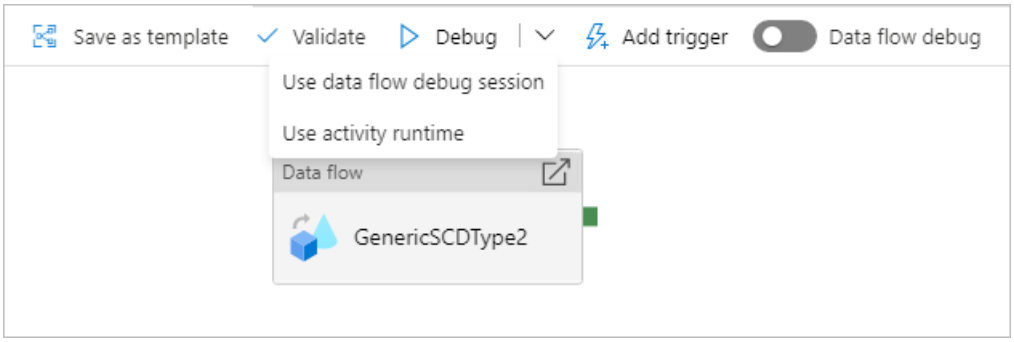
Debugging a pipeline with a data flow activity

When executing a debug pipeline run with a data flow, you have two options on which compute to use. You can either use an existing debug cluster or spin up a new just-in-time cluster for your data flows.

Using an existing debug session will greatly reduce the data flow start up time as the cluster is already running, but is not recommended for complex or parallel workloads as it may fail when multiple jobs are run at once.

Using the activity runtime will create a new cluster using the settings specified in each data flow activity's integration runtime. This allows each job to be isolated and should be used for complex workloads or performance testing. You can also control the TTL in the Azure IR so that the cluster resources used for debugging will still be available for that time period to serve additional job requests.

> **NOTE**
> If you have a pipeline with data flows executing in parallel or data flows that need to be tested with large datasets, choose "Use Activity Runtime" so that the service can use the Integration Runtime that you've selected in your data flow activity. This will allow the data flows to execute on multiple clusters and can accommodate your parallel data flow executions.



Next steps

After testing your changes, promote them to higher environments using continuous integration and deployment.

Azure Data Factory and Azure Synapse Analytics connector overview

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Azure Data Factory and Azure Synapse Analytics pipelines support the following data stores and formats via Copy, Data Flow, Look up, Get Metadata, and Delete activities. Click each data store to learn the supported capabilities and the corresponding configurations in details.

Supported data stores

CATEGORY	DATA STORE	COPY ACTIVITY (SOURCE/SINK)	MAPPING DATA FLOW (SOURCE/SINK)	LOOKUP ACTIVITY	GET METADATA ACTIVITY/VALIDATION ACTIVITY	DELETE ACTIVITY	MANAGED PRIVATE ENDPOINT
Azure	Azure Blob Storage	✓/✓	✓/✓	✓	✓	✓	✓ Exclude storage account V1
	Azure Cognitive Search Index	–/✓					✓
	Azure Cosmos DB (SQL API)	✓/✓	✓/✓	✓			✓
	Azure Cosmos DB's API for MongoDB	✓/✓					✓
	Azure Data Explorer	✓/✓	✓/✓	✓			
	Azure Data Lake Storage Gen1	✓/✓	✓/✓	✓	✓	✓	
	Azure Data Lake Storage Gen2	✓/✓	✓/✓	✓	✓	✓	✓
	Azure Database for MariaDB	✓/–		✓			✓

CATEGORY	DATA STORE	COPY ACTIVITY (SOURCE/SINK)	MAPPING DATA FLOW (SOURCE/SINK)	LOOKUP ACTIVITY	GET METADATA ACTIVITY/VALIDATION ACTIVITY	DELETE ACTIVITY	MANAGED PRIVATE ENDPOINT
	Azure Database for MySQL	✓/✓	✓/✓	✓			✓
	Azure Database for PostgreSQL	✓/✓	✓/✓	✓			✓
	Azure Databricks Delta Lake	✓/✓	✓/✓ Use delta format	✓			
	Azure Files	✓/✓		✓	✓	✓	✓ Exclude storage account V1
	Azure SQL Database	✓/✓	✓/✓	✓	✓		✓
	Azure SQL Managed Instance	✓/✓	✓/✓	✓	✓		✓ Public preview
	Azure Synapse Analytics	✓/✓	✓/✓	✓	✓		✓
	Azure Table Storage	✓/✓		✓			✓ Exclude storage account V1
Database	Amazon RDS for Oracle	✓/–		✓			
	Amazon RDS for SQL Server	✓/–		✓	✓		
	Amazon Redshift	✓/–		✓			
	DB2	✓/–		✓			
	Drill	✓/–		✓			
	Google BigQuery	✓/–		✓			
	Greenplum	✓/–		✓			

CATEGORY	DATA STORE	COPY ACTIVITY (SOURCE/SINK)	MAPPING DATA FLOW (SOURCE/SINK)	LOOKUP ACTIVITY	GET METADATA ACTIVITY/VALIDATION ACTIVITY	DELETE ACTIVITY	MANAGED PRIVATE ENDPOINT
	HBase	✓/–		✓			
	Hive	✓/–	✓/–	✓			
	Apache Impala	✓/–		✓			
	Informix	✓/✓		✓			
	MariaDB	✓/–		✓			
	Microsoft Access	✓/✓		✓			
	MySQL	✓/–		✓			
	Netezza	✓/–		✓			
	Oracle	✓/✓		✓			
	Phoenix	✓/–		✓			
	PostgreSQL	✓/–		✓			
	Presto (Preview)	✓/–		✓			
	SAP Business Warehouse Open Hub	✓/–		✓			
	SAP Business Warehouse via MDX	✓/–		✓			
	SAP HANA	✓/✓		✓			
	SAP Table	✓/–		✓			
	Snowflake	✓/✓	✓/✓	✓			
	Spark	✓/–		✓			
	SQL Server	✓/✓	✓/✓ Use Managed VNET	✓	✓		

CATEGORY	DATA STORE	COPY ACTIVITY (SOURCE/SINK)	MAPPING DATA FLOW (SOURCE/SINK)	LOOKUP ACTIVITY	GET METADATA ACTIVITY/VALIDATION ACTIVITY	DELETE ACTIVITY	MANAGED PRIVATE ENDPOINT
	Sybase	✓/–		✓			
	Teradata	✓/–		✓			
	Vertica	✓/–		✓			
NoSQL	Cassandra	✓/–		✓			
	Couchbase (Preview)	✓/–		✓			
	MongoDB	✓/✓					
	MongoDB Atlas	✓/✓					
File	Amazon S3	✓/–	✓/–	✓	✓	✓	
	Amazon S3 Compatible Storage	✓/–		✓	✓	✓	
	File System	✓/✓		✓	✓	✓	
	FTP	✓/–		✓	✓	✓	
	Google Cloud Storage	✓/–		✓	✓	✓	
	HDFS	✓/–		✓	✓	✓	
	Oracle Cloud Storage	✓/–		✓	✓	✓	
	SFTP	✓/✓	✓/✓	✓	✓	✓	
Generic protocol	Generic HTTP	✓/–		✓			
	Generic OData	✓/–		✓			
	Generic ODBC	✓/✓		✓			
	Generic REST	✓/✓	✓/✓				

CATEGORY	DATA STORE	COPY ACTIVITY (SOURCE/SINK)	MAPPING DATA FLOW (SOURCE/SINK)	LOOKUP ACTIVITY	GET METADATA ACTIVITY/VALIDATION ACTIVITY	DELETE ACTIVITY	MANAGED PRIVATE ENDPOINT
Services and apps	Amazon Marketplace Web Service	✓/–		✓			
	Asana (Preview)		✓/–				
	Concur (Preview)	✓/–		✓			
	data.world (Preview)		✓/–				
	Dataverse	✓/✓	✓/✓	✓			
	Dynamics 365	✓/✓	✓/✓	✓			
	Dynamics AX	✓/–		✓			
	Dynamics CRM	✓/✓	✓/✓	✓			
	GitHub		For Common Data Model entity reference				
	Google AdWords	✓/–		✓			
	HubSpot (Preview)	✓/–		✓			
	Jira	✓/–		✓			
	Magento (Preview)	✓/–		✓			
	Marketo (Preview)	✓/–		✓			
	Microsoft 365	✓/–		✓			
	Oracle Eloqua (Preview)	✓/–		✓			

CATEGORY	DATA STORE	COPY ACTIVITY (SOURCE/SINK)	MAPPING DATA FLOW (SOURCE/SINK)	LOOKUP ACTIVITY	GET METADATA ACTIVITY/VALIDATION ACTIVITY	DELETE ACTIVITY	MANAGED PRIVATE ENDPOINT
	Oracle Responsys (Preview)	✓/–		✓			
	Oracle Service Cloud (Preview)	✓/–		✓			
	PayPal (Preview)	✓/–		✓			
	Quickbase (Preview)		✓/–				
	QuickBooks (Preview)	✓/–		✓			
	Salesforce	✓/✓		✓			
	Salesforce Service Cloud	✓/✓		✓			
	Salesforce Marketing Cloud	✓/–		✓			
	SAP Cloud for Customer (C4C)	✓/✓		✓			
	SAP ECC	✓/–		✓			
	ServiceNow	✓/–		✓			
	SharePoint Online List	✓/–		✓			
	Shopify (Preview)	✓/–		✓			
	Smartsheet (Preview)		✓/–				
	Square (Preview)	✓/–		✓			
	TeamDesk (Preview)		✓/–				

CATEGORY	DATA STORE	COPY ACTIVITY (SOURCE/SINK)	MAPPING DATA FLOW (SOURCE/SINK)	LOOKUP ACTIVITY	GET METADATA ACTIVITY/VALIDATION ACTIVITY	DELETE ACTIVITY	MANAGED PRIVATE ENDPOINT
	Twilio (Preview)		✓/–				
	Web Table (HTML table)	✓/–		✓			
	Xero	✓/–		✓			
	Zendesk (Preview)		✓/–				
	Zoho (Preview)	✓/–		✓			

NOTE

Any connector marked as *Preview* means that you can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Integrate with more data stores

Azure Data Factory and Synapse pipelines can reach broader set of data stores than the list mentioned above. If you need to move data to/from a data store that is not in the service built-in connector list, here are some extensible options:

- For database and data warehouse, usually you can find a corresponding ODBC driver, with which you can use generic ODBC connector.
- For SaaS applications:
  - If it provides RESTful APIs, you can use generic REST connector.
  - If it has OData feed, you can use generic OData connector.
  - If it provides SOAP APIs, you can use generic HTTP connector.
  - If it has ODBC driver, you can use generic ODBC connector.
- For others, check if you can load data to or expose data as any supported data stores, e.g. Azure Blob/File/FTP/SFTP/etc, then let the service pick up from there. You can invoke custom data loading mechanism via Azure Function, Custom activity, Databricks/HDInsight, Web activity, etc.

Supported file formats

The following file formats are supported. Refer to each article for format-based settings.

- Avro format
- Binary format
- Common Data Model format
- Delimited text format
- Delta format
- Excel format

- JSON format
- ORC format
- Parquet format
- XML format

Next steps

- Copy activity
- Mapping Data Flow
- Lookup Activity
- Get Metadata Activity
- Delete Activity

Copy data from Amazon Marketplace Web Service using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Amazon Marketplace Web Service. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Amazon Marketplace Web Service connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

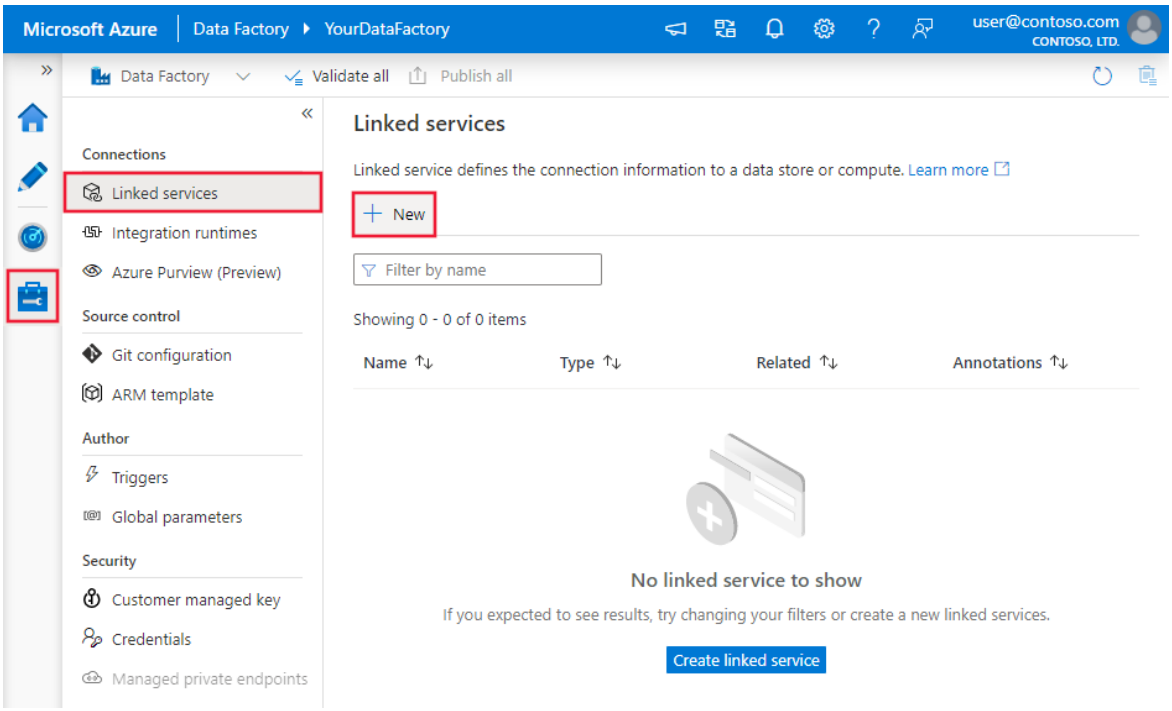
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Amazon Marketplace Web Service using UI

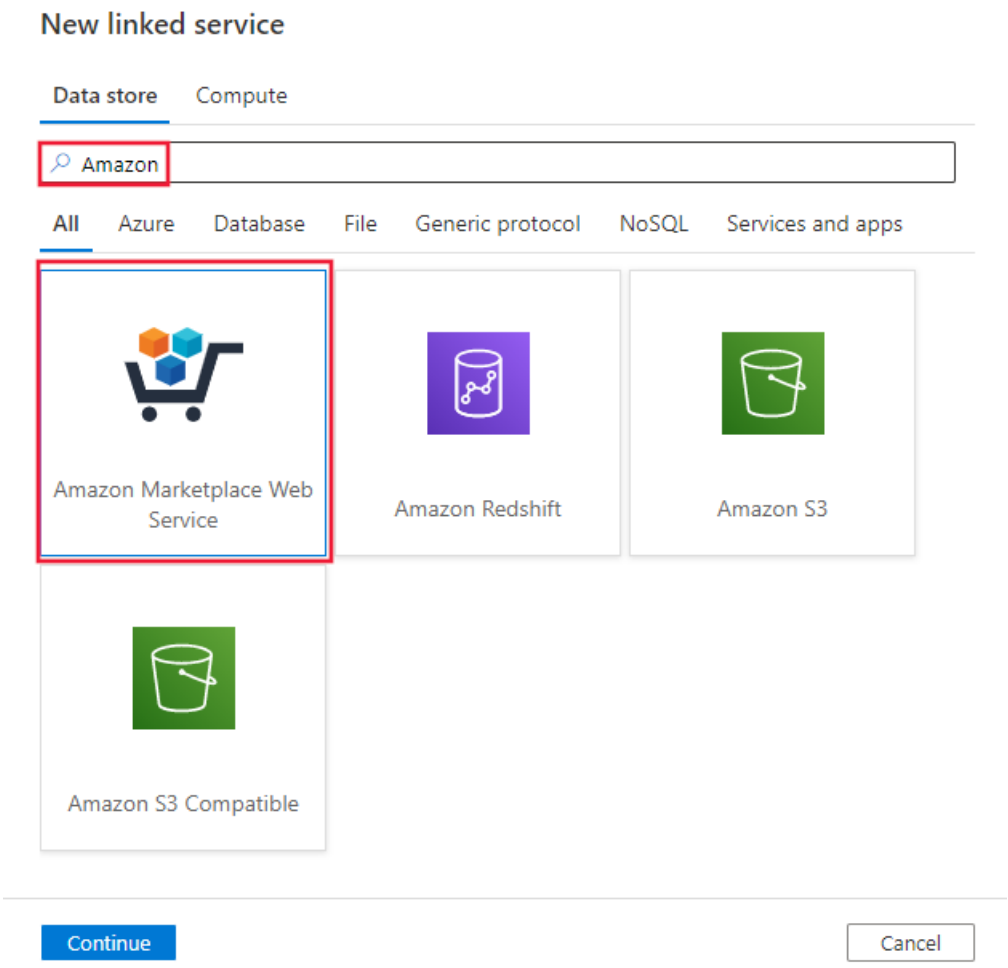
Use the following steps to create a linked service to Amazon Marketplace Web Service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

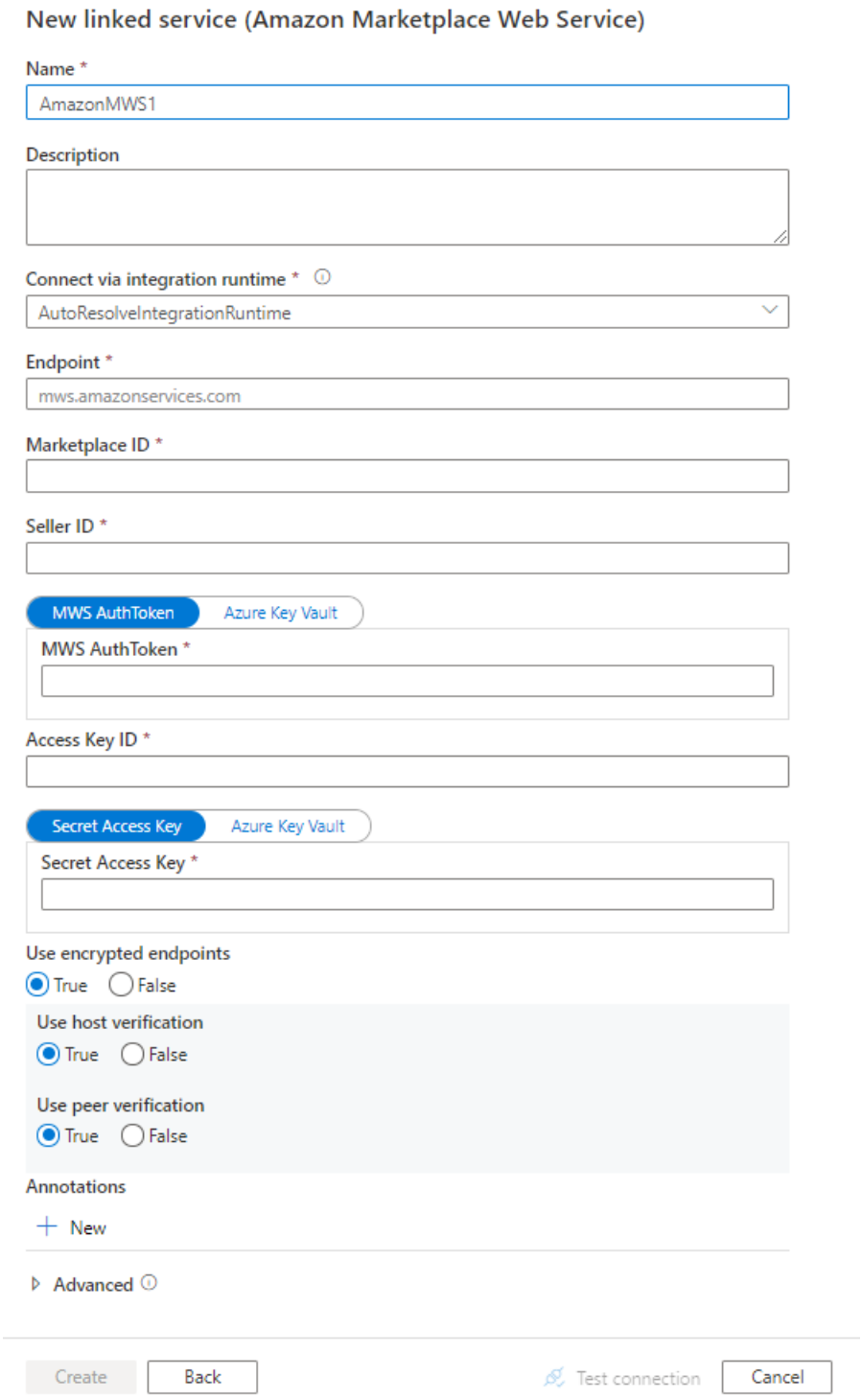
   - Azure Data Factory
   - Azure Synapse



2. Search for Amazon and select the Amazon Marketplace Web Service connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Amazon Marketplace Web Service connector.

Linked service properties

The following properties are supported for Amazon Marketplace Web Service linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **AmazonMWS**	Yes

PROPERTY	DESCRIPTION	REQUIRED
endpoint	The endpoint of the Amazon MWS server, (that is, mws.amazonservices.com)	Yes
marketplaceID	The Amazon Marketplace ID you want to retrieve data from. To retrieve data from multiple Marketplace IDs, separate them with a comma (`,`). (that is, A2EUQ1WTGCTBG2)	Yes
sellerID	The Amazon seller ID.	Yes
mwsAuthToken	The Amazon MWS authentication token. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
accessKeyId	The access key ID used to access data.	Yes
secretKey	The secret key used to access data. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
    "name": "AmazonMWSLinkedService",
    "properties": {
        "type": "AmazonMWS",
        "typeProperties": {
            "endpoint" : "mws.amazonservices.com",
            "marketplaceID" : "A2EUQ1WTGCTBG2",
            "sellerID" : "<sellerID>",
            "mwsAuthToken": {
                 "type": "SecureString",
                 "value": "<mwsAuthToken>"
            },
            "accessKeyId" : "<accessKeyId>",
            "secretKey": {
                 "type": "SecureString",
                 "value": "<secretKey>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Amazon Marketplace Web Service dataset.

To copy data from Amazon Marketplace Web Service, set the type property of the dataset to **AmazonMWSObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **AmazonMWSObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "AmazonMWSDataset",
    "properties": {
        "type": "AmazonMWSObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<AmazonMWS linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Amazon Marketplace Web Service source.

Amazon MWS as source

To copy data from Amazon Marketplace Web Service, set the source type in the copy activity to **AmazonMWSSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **AmazonMWSSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM Orders where Amazon_Order_Id = 'xx'"` .	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromAmazonMWS",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<AmazonMWS input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AmazonMWSSource",
                "query": "SELECT * FROM Orders where Amazon_Order_Id = 'xx'"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Amazon RDS for Oracle by using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 12 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the copy activity in Azure Data Factory to copy data from an Amazon RDS for Oracle database. It builds on the copy activity overview.

Supported capabilities

This Amazon RDS for Oracle connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks by the copy activity, see the Supported data stores table.

Specifically, this Amazon RDS for Oracle connector supports:

- The following versions of an Amazon RDS for Oracle database:
  - Amazon RDS for Oracle 19c R1 (19.1) and higher
  - Amazon RDS for Oracle 18c R1 (18.1) and higher
  - Amazon RDS for Oracle 12c R1 (12.1) and higher
  - Amazon RDS for Oracle 11g R1 (11.1) and higher
- Parallel copying from an Amazon RDS for Oracle source. See the Parallel copy from Amazon RDS for Oracle section for details.

> **NOTE**
> Amazon RDS for Oracle proxy server isn't supported.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

The integration runtime provides a built-in Amazon RDS for Oracle driver. Therefore, you don't need to manually install a driver when you copy data from Amazon RDS for Oracle.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK

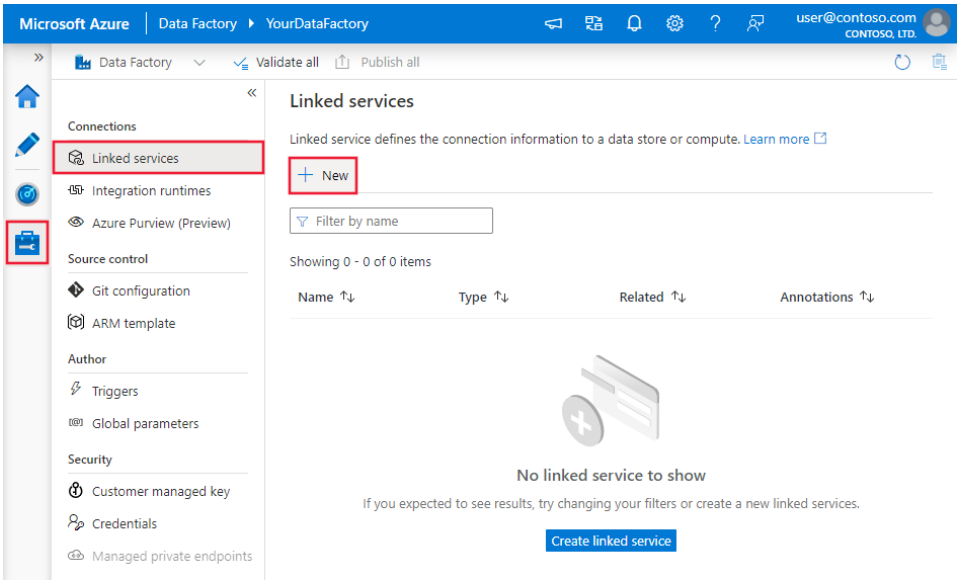
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Amazon RDS for Oracle using UI

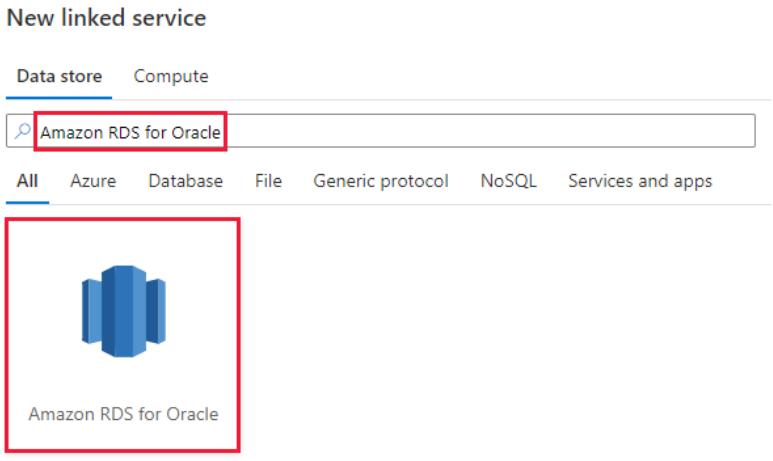
Use the following steps to create a linked service to Amazon RDS for Oracle in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

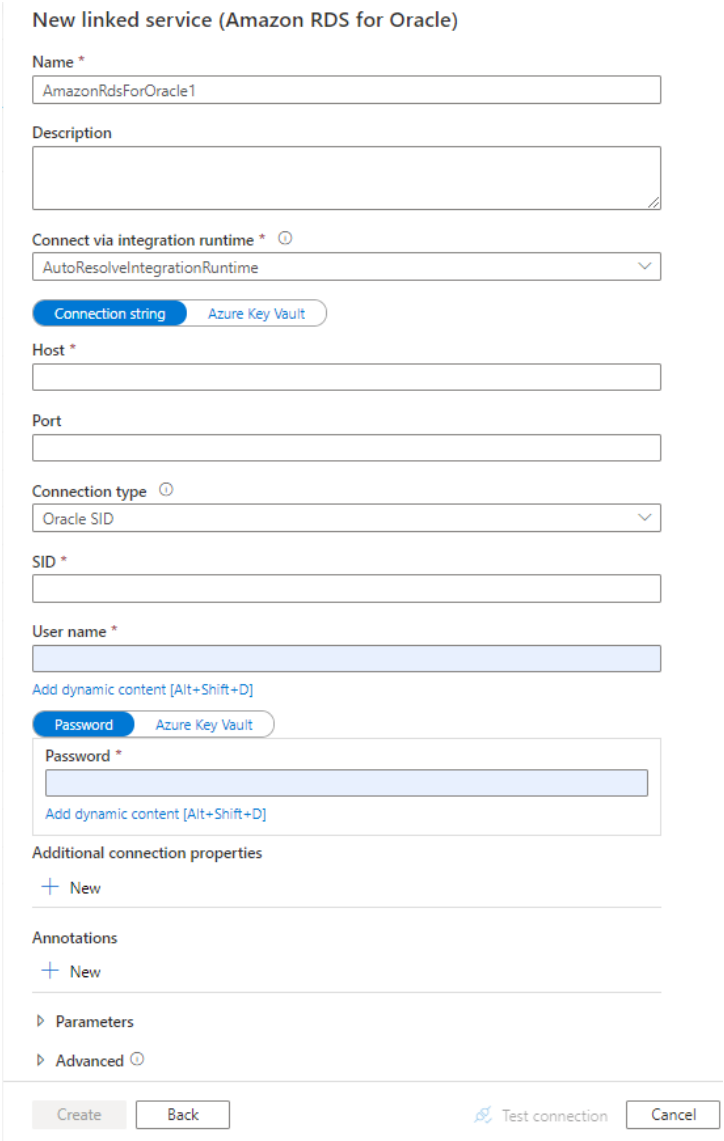
   - Azure Data Factory
   - Azure Synapse



2. Search for Amazon RDS for Oracle and select the Amazon RDS for Oracle connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to the Amazon RDS for Oracle connector.

Linked service properties

The Amazon RDS for Oracle linked service supports the following properties:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AmazonRdsForOracle**.	Yes

PROPERTY	DESCRIPTION	REQUIRED

connectionString	Specifies the information needed to connect to the Amazon RDS for Oracle Database instance.
	You can also put a password in Azure Key Vault, and pull the `password` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault with more details.
	
	Supported connection type: You can use **Amazon RDS for Oracle SID** or **Amazon RDS for Oracle Service Name** to identify your database:
	- If you use SID:
	`Host=<host>;Port=<port>;Sid=<sid>;User Id=<username>;Password=<password>;`
	- If you use Service Name:
	`Host=<host>;Port=<port>;ServiceName=<servicename>;User Id=<username>;Password=<password>;`
	For advanced Amazon RDS for Oracle native connection options, you can choose to add an entry in TNSNAMES.ORA file on the Amazon RDS for Oracle server, and in Amazon RDS for Oracle linked service, choose to use Amazon RDS for Oracle Service Name connection type and configure the corresponding service name.
		Yes

connectVia	The integration runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, the default Azure Integration Runtime is used.	No

If you have multiple Amazon RDS for Oracle instances for failover scenario, you can create Amazon RDS for Oracle linked service and fill in the primary host, port, user name, password, etc., and add a new "**Additional connection properties**" with property name as `AlternateServers` and value as `(HostName=<secondary host>:PortNumber=<secondary port>:ServiceName=<secondary service name>)` - do not miss the brackets and pay attention to the colons (`:`) as separator. As an example, the following value of alternate servers defines two alternate database servers for connection failover:

```
(HostName=AccountingAmazonRdsForOracleServer:PortNumber=1521:SID=Accounting,HostName=255.201.11.24:PortNumber=1522:ServiceName=ABackup.
```

.

More connection properties you can set in connection string per your case:

PROPERTY	DESCRIPTION	ALLOWED VALUES
ArraySize	The number of bytes the connector can fetch in a single network round trip. For example, `ArraySize=10485760`. Larger values increase throughput by reducing the number of times to fetch data across the network. Smaller values increase response time, as there is less of a delay waiting for the server to transmit data.	An integer from 1 to 4294967296 (4 GB). Default value is `60000`. The value 1 does not define the number of bytes, but indicates allocating space for exactly one row of data.

To enable encryption on Amazon RDS for Oracle connection, you have two options:

- To use **Triple-DES Encryption (3DES) and Advanced Encryption Standard (AES)**, on the Amazon RDS for Oracle server side, go to Oracle Advanced Security (OAS) and configure the encryption settings. For details, see this Oracle documentation. The Amazon RDS for Oracle Application Development Framework (ADF) connector automatically negotiates the encryption method to use the one you configure in OAS when establishing a connection to Amazon RDS for Oracle.

- To use **TLS**:

  1. Get the TLS/SSL certificate info. Get the Distinguished Encoding Rules (DER)-encoded certificate information of your TLS/SSL cert, and save the output (----- Begin Certificate ... End Certificate -----) as a text file.

     ```
     openssl x509 -inform DER -in [Full Path to the DER Certificate including the name of the DER Certificate] -text
     ```

     Example: Extract cert info from DERcert.cer, and then save the output to cert.txt.

     ```
     openssl x509 -inform DER -in DERcert.cer -text
     Output:
     -----BEGIN CERTIFICATE-----
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     XXXXXXXXX
     -----END CERTIFICATE-----
     ```

  2. Build the `keystore` or `truststore`. The following command creates the `truststore` file, with or without a password, in PKCS-12 format.

     ```
     openssl pkcs12 -in [Path to the file created in the previous step] -out [Path and name of TrustStore] -passout pass:[Keystore PWD] -nokeys -export
     ```

     Example: Create a PKCS12 `truststore` file, named MyTrustStoreFile, with a password.

     ```
     openssl pkcs12 -in cert.txt -out MyTrustStoreFile -passout pass:ThePWD -nokeys -export
     ```

  3. Place the `truststore` file on the self-hosted IR machine. For example, place the file at C:\MyTrustStoreFile.

  4. In the service, configure the Amazon RDS for Oracle connection string with `EncryptionMethod=1` and the corresponding `TrustStore` / `TrustStorePassword` value. For example, `Host=<host>;Port=<port>;Sid=<sid>;User Id=<username>;Password=<password>;EncryptionMethod=1;TrustStore=C:\\MyTrustStoreFile;TrustStorePassword=<trust_store_password>`.

Example:

```
{
    "name": "AmazonRdsForOracleLinkedService",
    "properties": {
        "type": "AmazonRdsForOracle",
        "typeProperties": {
            "connectionString": "Host=<host>;Port=<port>;Sid=<sid>;User Id=<username>;Password=<password>;"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "AmazonRdsForOracleLinkedService",
    "properties": {
        "type": "AmazonRdsForOracle",
        "typeProperties": {
            "connectionString": "Host=<host>;Port=<port>;Sid=<sid>;User Id=<username>;",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

This section provides a list of properties supported by the Amazon RDS for Oracle dataset. For a full list of sections and properties available for defining datasets, see Datasets.

To copy data from Amazon RDS for Oracle, set the type property of the dataset to `AmazonRdsForOracleTable`. The following properties are supported.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to `AmazonRdsForOracleTable`.	Yes
schema	Name of the schema.	No
table	Name of the table/view.	No
tableName	Name of the table/view with schema. This property is supported for backward compatibility. For new workload, use `schema` and `table`.	No

Example:

```
{
    "name": "AmazonRdsForOracleDataset",
    "properties":
    {
        "type": "AmazonRdsForOracleTable",
        "schema": [],
        "typeProperties": {
            "schema": "<schema_name>",
            "table": "<table_name>"
        },
        "linkedServiceName": {
            "referenceName": "<Amazon RDS for Oracle linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

This section provides a list of properties supported by the Amazon RDS for Oracle source. For a full list of sections and properties available for defining activities, see Pipelines.

Amazon RDS for Oracle as source

TIP

To load data from Amazon RDS for Oracle efficiently by using data partitioning, learn more from Parallel copy from Amazon RDS for Oracle.

To copy data from Amazon RDS for Oracle, set the source type in the copy activity to `AmazonRdsForOracleSource`. The following properties are supported in the copy activity **source** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to `AmazonRdsForOracleSource`.	Yes
oracleReaderQuery	Use the custom SQL query to read data. An example is `"SELECT * FROM MyTable"`. When you enable partitioned load, you need to hook any corresponding built-in partition parameters in your query. For examples, see the Parallel copy from Amazon RDS for Oracle section.	No
partitionOptions	Specifies the data partitioning options used to load data from Amazon RDS for Oracle. Allowed values are: **None** (default), **PhysicalPartitionsOfTable**, and **DynamicRange**. When a partition option is enabled (that is, not `None`), the degree of parallelism to concurrently load data from an Amazon RDS for Oracle database is controlled by the `parallelCopies` setting on the copy activity.	No
partitionSettings	Specify the group of the settings for data partitioning. Apply when the partition option isn't `None`.	No
partitionNames	The list of physical partitions that needs to be copied. Apply when the partition option is `PhysicalPartitionsOfTable`. If you use a query to retrieve the source data, hook `?AdfTabularPartitionName` in the WHERE clause. For an example, see the Parallel copy from Amazon RDS for Oracle section.	No
partitionColumnName	Specify the name of the source column **in integer type** that will be used by range partitioning for parallel copy. If not specified, the primary key of the table is auto-detected and used as the partition column. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfRangePartitionColumnName` in the WHERE clause. For an example, see the Parallel copy from Amazon RDS for Oracle section.	No

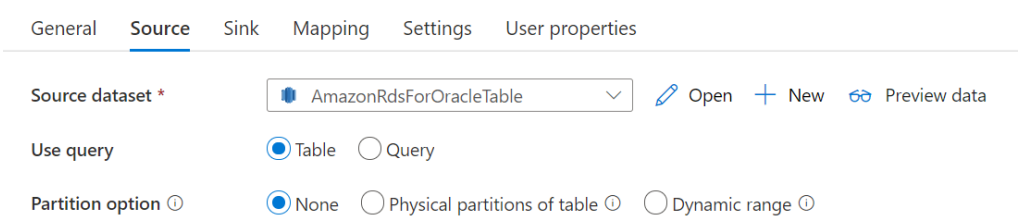
PROPERTY	DESCRIPTION	REQUIRED
partitionUpperBound	The maximum value of the partition column to copy data out. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfRangePartitionUpbound` in the WHERE clause. For an example, see the Parallel copy from Amazon RDS for Oracle section.	No
partitionLowerBound	The minimum value of the partition column to copy data out. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfRangePartitionLowbound` in the WHERE clause. For an example, see the Parallel copy from Amazon RDS for Oracle section.	No

Example: copy data by using a basic query without partition

```
"activities":[
    {
        "name": "CopyFromAmazonRdsForOracle",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Amazon RDS for Oracle input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AmazonRdsForOracleSource",
                "oracleReaderQuery": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Parallel copy from Amazon RDS for Oracle

The Amazon RDS for Oracle connector provides built-in data partitioning to copy data from Amazon RDS for Oracle in parallel. You can find data partitioning options on the **Source** tab of the copy activity.



When you enable partitioned copy, the service runs parallel queries against your Amazon RDS for Oracle source to load data by partitions. The parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your Amazon RDS for Oracle database.

You are suggested to enable parallel copy with data partitioning especially when you load large amount of data from your Amazon RDS for Oracle database. The following are suggested configurations for different scenarios.

When copying data into file-based data store, it's recommanded to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

SCENARIO	SUGGESTED SETTINGS
Full load from large table, with physical partitions.	**Partition option**: Physical partitions of table. During execution, the service automatically detects the physical partitions, and copies data by partitions.
Full load from large table, without physical partitions, while with an integer column for data partitioning.	**Partition options**: Dynamic range partition. **Partition column**: Specify the column used to partition data. If not specified, the primary key column is used.
Load a large amount of data by using a custom query, with physical partitions.	**Partition option**: Physical partitions of table. **Query**: `SELECT * FROM <TABLENAME> PARTITION("?AdfTabularPartitionName") WHERE <your_additional_where_clause>` . **Partition name**: Specify the partition name(s) to copy data from. If not specified, the service automatically detects the physical partitions on the table you specified in the Amazon RDS for Oracle dataset. During execution, the service replaces `?AdfTabularPartitionName` with the actual partition name, and sends to Amazon RDS for Oracle.
Load a large amount of data by using a custom query, without physical partitions, while with an integer column for data partitioning.	**Partition options**: Dynamic range partition. **Query**: `SELECT * FROM <TABLENAME> WHERE ?AdfRangePartitionColumnName <= ? AdfRangePartitionUpbound AND ?AdfRangePartitionColumnName >= ? AdfRangePartitionLowbound AND <your_additional_where_clause>` . **Partition column**: Specify the column used to partition data. You can partition against the column with integer data type. **Partition upper bound** and **partition lower bound**: Specify if you want to filter against partition column to retrieve data only between the lower and upper range. During execution, the service replaces `?AdfRangePartitionColumnName`, `?AdfRangePartitionUpbound`, and `?AdfRangePartitionLowbound` with the actual column name and value ranges for each partition, and sends to Amazon RDS for Oracle. For example, if your partition column "ID" is set with the lower bound as 1 and the upper bound as 80, with parallel copy set as 4, the service retrieves data by 4 partitions. Their IDs are between [1, 20], [21, 40], [41, 60], and [61, 80], respectively.

TIP

When copying data from a non-partitioned table, you can use "Dynamic range" partition option to partition against an integer column. If your source data doesn't have such type of column, you can leverage ORA_HASH function in source query to generate a column and use it as partition column.

Example: query with physical partition

```
"source": {
    "type": "AmazonRdsForOracleSource",
    "query":"SELECT * FROM <TABLENAME> PARTITION(\"?AdfTabularPartitionName\") WHERE
<your_additional_where_clause>",
    "partitionOption": "PhysicalPartitionsOfTable",
    "partitionSettings": {
        "partitionNames": [
            "<partitionA_name>",
            "<partitionB_name>"
        ]
    }
}
```

Example: query with dynamic range partition

```
"source": {
    "type": "AmazonRdsForOracleSource",
    "query":"SELECT * FROM <TABLENAME> WHERE ?AdfRangePartitionColumnName <= ?AdfRangePartitionUpbound AND ?
AdfRangePartitionColumnName >= ?AdfRangePartitionLowbound AND <your_additional_where_clause>",
    "partitionOption": "DynamicRange",
    "partitionSettings": {
        "partitionColumnName": "<partition_column_name>",
        "partitionUpperBound": "<upper_value_of_partition_column>",
        "partitionLowerBound": "<lower_value_of_partition_column>"
    }
}
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from Amazon RDS for SQL Server by using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 15 minutes to read • Edit Online

This article outlines how to use the copy activity in Azure Data Factory and Azure Synapse pipelines to copy data from Amazon RDS for SQL Server database. To learn more, read the introductory article for Azure Data Factory or Azure Synapse Analytics.

Supported capabilities

This Amazon RDS for SQL Server connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②
GetMetadata activity	① ②
Stored procedure activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks by the copy activity, see the Supported data stores table.

Specifically, this Amazon RDS for SQL Server connector supports:

- SQL Server version 2005 and above.
- Copying data by using SQL or Windows authentication.
- As a source, retrieving data by using a SQL query or a stored procedure. You can also choose to parallel copy from Amazon RDS for SQL Server source, see the Parallel copy from SQL database section for details.

SQL Server Express LocalDB is not supported.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data

access strategies.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

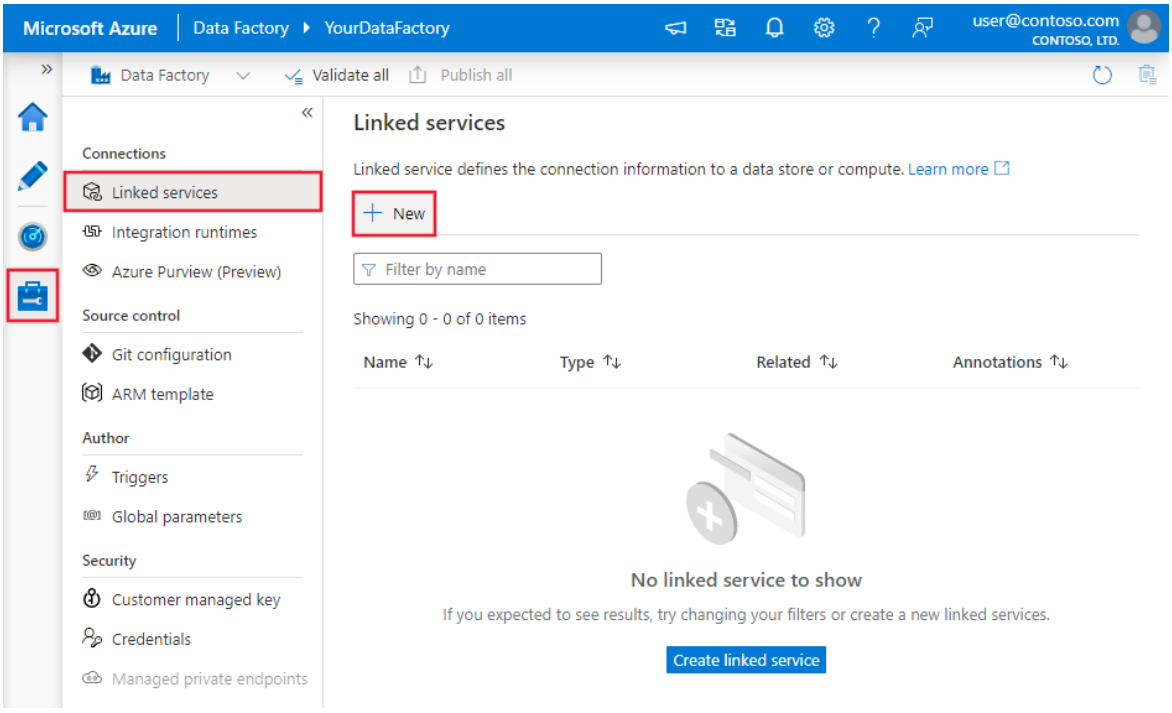
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create an Amazon RDS for SQL Server linked service using UI

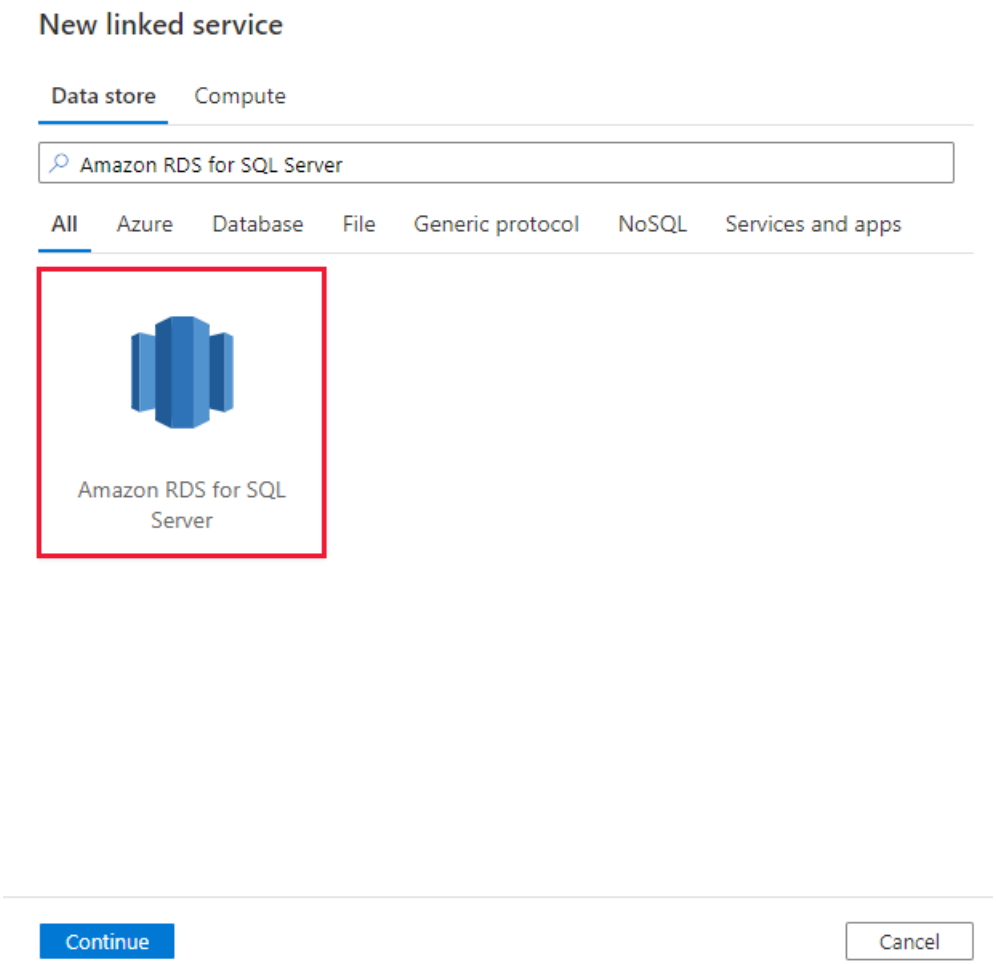
Use the following steps to create an Amazon RDS for SQL Server linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

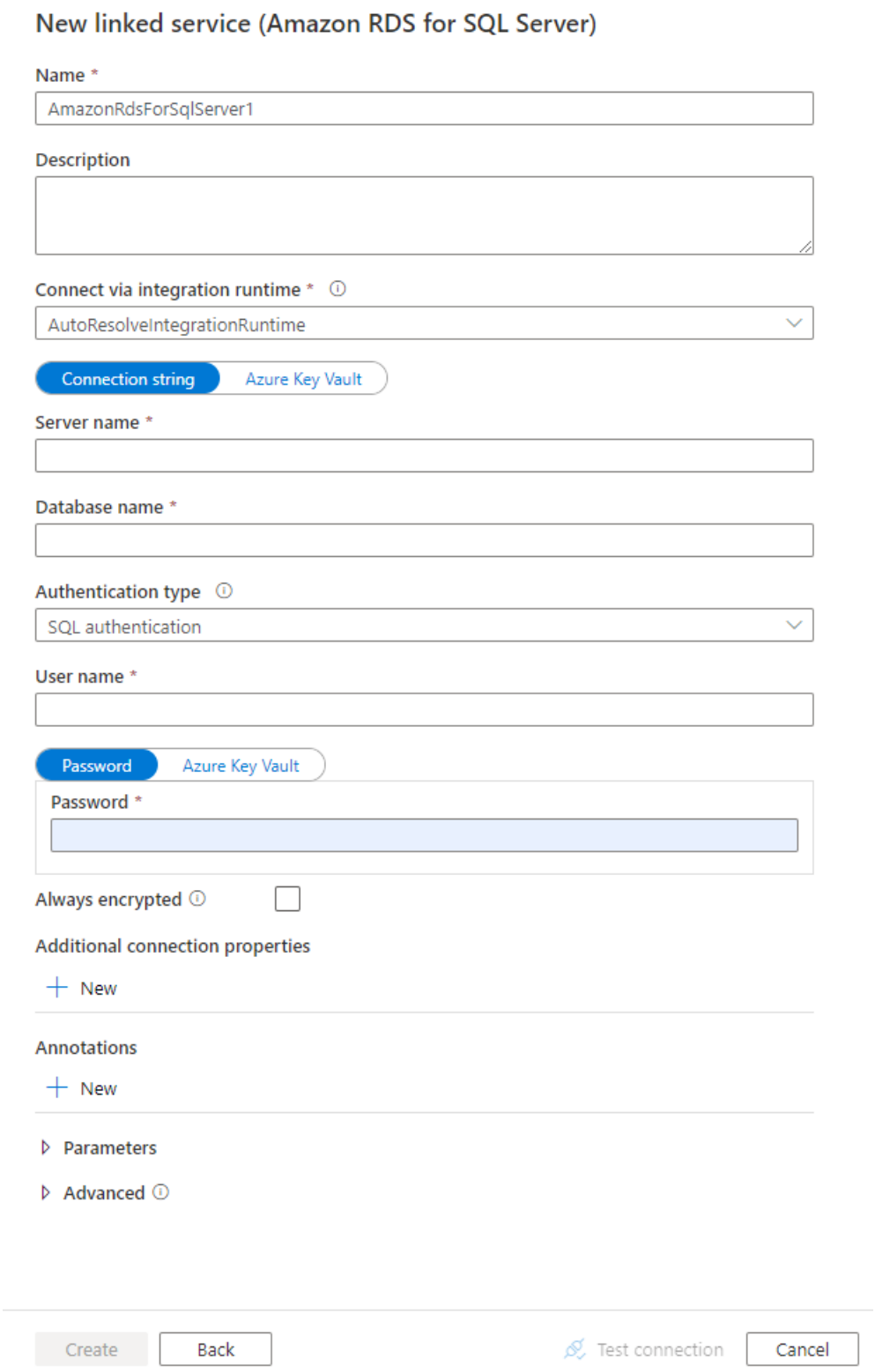
   - Azure Data Factory
   - Azure Synapse



2. Search for Amazon RDS for SQL server and select the Amazon RDS for SQL Server connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory and Synapse pipeline entities specific to the Amazon RDS for SQL Server database connector.

Linked service properties

The following properties are supported for the Amazon RDS for SQL Server linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AmazonRdsForSqlServer**.	Yes
connectionString	Specify **connectionString** information that's needed to connect to the Amazon RDS for SQL Server database by using either SQL authentication or Windows authentication. Refer to the following samples. You also can put a password in Azure Key Vault. If it's SQL authentication, pull the `password` configuration out of the connection string. For more information, see the JSON example following the table and Store credentials in Azure Key Vault.	Yes
userName	Specify a user name if you use Windows authentication. An example is **domainname\username**.	No
password	Specify a password for the user account you specified for the user name. Mark this field as **SecureString** to store it securely. Or, you can reference a secret stored in Azure Key Vault.	No
alwaysEncryptedSettings	Specify **alwaysencryptedsettings** information that's needed to enable Always Encrypted to protect sensitive data stored in Amazon RDS for SQL Server by using either managed identity or service principal. For more information, see the JSON example following the table and Using Always Encrypted section. If not specified, the default always encrypted setting is disabled.	No
connectVia	This integration runtime is used to connect to the data store. Learn more from Prerequisites section. If not specified, the default Azure integration runtime is used.	No

NOTE

Amazon RDS for SQL Server **Always Encrypted** is not supported in data flow.

TIP

If you hit an error with the error code "UserErrorFailedToConnectToSqlServer" and a message like "The session limit for the database is XXX and has been reached," add `Pooling=false` to your connection string and try again.

Example 1: Use SQL authentication

```
{
    "name": "AmazonSqlLinkedService",
    "properties": {
        "type": "AmazonRdsForSqlServer",
        "typeProperties": {
            "connectionString": "Data Source=<servername>\\<instance name if using named instance>;Initial
Catalog=<databasename>;Integrated Security=False;User ID=<username>;Password=<password>;"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 2: Use SQL authentication with a password in Azure Key Vault

```
{
    "name": "AmazonSqlLinkedService",
    "properties": {
        "type": "AmazonRdsForSqlServer",
        "typeProperties": {
            "connectionString": "Data Source=<servername>\\<instance name if using named instance>;Initial
Catalog=<databasename>;Integrated Security=False;User ID=<username>;",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 3: Use Windows authentication

```
{
    "name": "AmazonSqlLinkedService",
    "properties": {
        "type": "AmazonRdsForSqlServer",
        "typeProperties": {
            "connectionString": "Data Source=<servername>\\<instance name if using named instance>;Initial
Catalog=<databasename>;Integrated Security=True;",
            "userName": "<domain\\username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 4: Use Always Encrypted

```
{
    "name": "AmazonSqlLinkedService",
    "properties": {
        "type": "AmazonRdsForSqlServer",
        "typeProperties": {
            "connectionString": "Data Source=<servername>\\<instance name if using named instance>;Initial
Catalog=<databasename>;Integrated Security=False;User ID=<username>;Password=<password>;"
        },
        "alwaysEncryptedSettings": {
            "alwaysEncryptedAkvAuthType": "ServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by the Amazon RDS for SQL Server dataset.

To copy data from an Amazon RDS for SQL Server database, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **AmazonRdsForSqlServerTable**.	Yes
schema	Name of the schema.	No
table	Name of the table/view.	No
tableName	Name of the table/view with schema. This property is supported for backward compatibility. For new workload, use `schema` and `table`.	No

Example

```
{
    "name": "AmazonRdsForSQLServerDataset",
    "properties":
    {
        "type": "AmazonRdsForSqlServerTable",
        "linkedServiceName": {
            "referenceName": "<Amazon RDS for SQL Server linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "typeProperties": {
            "schema": "<schema_name>",
            "table": "<table_name>"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for use to define activities, see the Pipelines article. This section provides a list of properties supported by the Amazon RDS for SQL Server source.

Amazon RDS for SQL Server as a source

> **TIP**
> To load data from Amazon RDS for SQL Server efficiently by using data partitioning, learn more from Parallel copy from SQL database.

To copy data from Amazon RDS for SQL Server, set the source type in the copy activity to **AmazonRdsForSqlServerSource**. The following properties are supported in the copy activity source section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **AmazonRdsForSqlServerSource**.	Yes
sqlReaderQuery	Use the custom SQL query to read data. An example is `select * from MyTable`.	No
sqlReaderStoredProcedureName	This property is the name of the stored procedure that reads data from the source table. The last SQL statement must be a SELECT statement in the stored procedure.	No
storedProcedureParameters	These parameters are for the stored procedure. Allowed values are name or value pairs. The names and casing of parameters must match the names and casing of the stored procedure parameters.	No

PROPERTY	DESCRIPTION	REQUIRED
isolationLevel	Specifies the transaction locking behavior for the SQL source. The allowed values are: **ReadCommitted**, **ReadUncommitted**, **RepeatableRead**, **Serializable**, **Snapshot**. If not specified, the database's default isolation level is used. Refer to this doc for more details.	No
partitionOptions	Specifies the data partitioning options used to load data from Amazon RDS for SQL Server. Allowed values are: **None** (default), **PhysicalPartitionsOfTable**, and **DynamicRange**. When a partition option is enabled (that is, not `None`), the degree of parallelism to concurrently load data from Amazon RDS for SQL Server is controlled by the `parallelCopies` setting on the copy activity.	No
partitionSettings	Specify the group of the settings for data partitioning. Apply when the partition option isn't `None`.	No
Under `partitionSettings`***:***		
partitionColumnName	Specify the name of the source column **in integer or date/datetime type** (`int`, `smallint`, `bigint`, `date`, `smalldatetime`, `datetime`, `datetime2`, or `datetimeoffset`) that will be used by range partitioning for parallel copy. If not specified, the index or the primary key of the table is auto-detected and used as the partition column. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfDynamicRangePartitionCondition` in the WHERE clause. For an example, see the Parallel copy from SQL database section.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionUpperBound	The maximum value of the partition column for partition range splitting. This value is used to decide the partition stride, not for filtering the rows in table. All rows in the table or query result will be partitioned and copied. If not specified, copy activity auto detect the value. Apply when the partition option is `DynamicRange`. For an example, see the Parallel copy from SQL database section.	No
partitionLowerBound	The minimum value of the partition column for partition range splitting. This value is used to decide the partition stride, not for filtering the rows in table. All rows in the table or query result will be partitioned and copied. If not specified, copy activity auto detect the value. Apply when the partition option is `DynamicRange`. For an example, see the Parallel copy from SQL database section.	No

Note the following points:

- If **sqlReaderQuery** is specified for **AmazonRdsForSqlServerSource**, the copy activity runs this query against the Amazon RDS for SQL Server source to get the data. You also can specify a stored procedure by specifying **sqlReaderStoredProcedureName** and **storedProcedureParameters** if the stored procedure takes parameters.
- When using stored procedure in source to retrieve data, note if your stored procedure is designed as returning different schema when different parameter value is passed in, you may encounter failure or see unexpected result when importing schema from UI or when copying data to SQL database with auto table creation.

Example: Use SQL query

```
"activities":[
    {
        "name": "CopyFromAmazonRdsForSQLServer",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Amazon RDS for SQL Server input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AmazonRdsForSqlServerSource",
                "sqlReaderQuery": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Example: Use a stored procedure

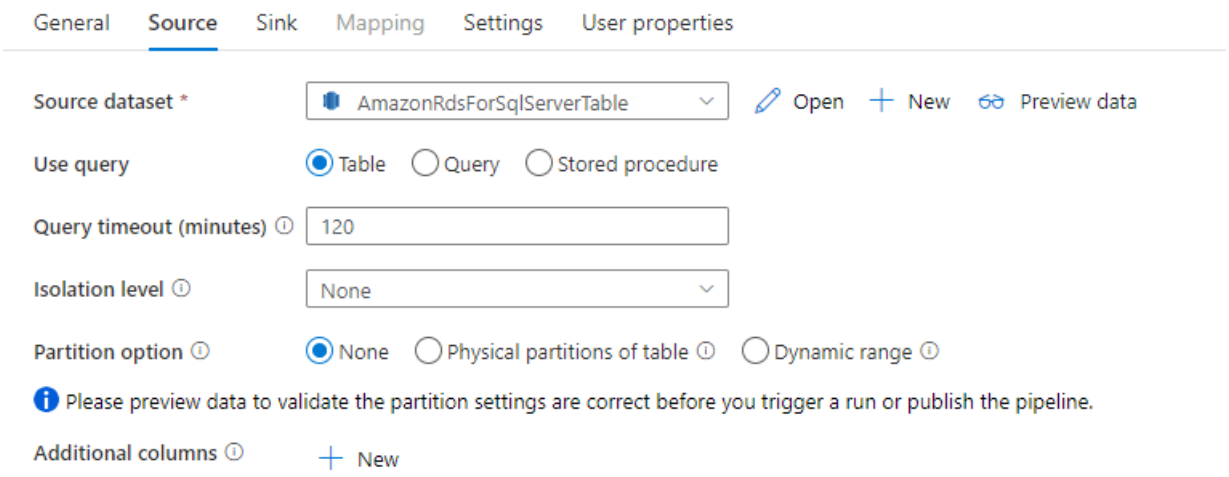
```
"activities":[
    {
        "name": "CopyFromAmazonRdsForSQLServer",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Amazon RDS for SQL Server input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AmazonRdsForSqlServerSource",
                "sqlReaderStoredProcedureName": "CopyTestSrcStoredProcedureWithParameters",
                "storedProcedureParameters": {
                    "stringData": { "value": "str3" },
                    "identifier": { "value": "$$Text.Format('{0:yyyy}', <datetime parameter>)", "type":
"Int"}
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

The stored procedure definition

```
CREATE PROCEDURE CopyTestSrcStoredProcedureWithParameters
(
    @stringData varchar(20),
    @identifier int
)
AS
SET NOCOUNT ON;
BEGIN
    select *
    from dbo.UnitTestSrcTable
    where dbo.UnitTestSrcTable.stringData != stringData
    and dbo.UnitTestSrcTable.identifier != identifier
END
GO
```

Parallel copy from SQL database

The Amazon RDS for SQL Server connector in copy activity provides built-in data partitioning to copy data in parallel. You can find data partitioning options on the **Source** tab of the copy activity.



When you enable partitioned copy, copy activity runs parallel queries against your Amazon RDS for SQL Server source to load data by partitions. The parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your Amazon RDS for SQL Server.

You are suggested to enable parallel copy with data partitioning especially when you load large amount of data from your Amazon RDS for SQL Server. The following are suggested configurations for different scenarios. When copying data into file-based data store, it's recommended to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

SCENARIO	SUGGESTED SETTINGS
Full load from large table, with physical partitions.	**Partition option**: Physical partitions of table. During execution, the service automatically detects the physical partitions, and copies data by partitions. To check if your table has physical partition or not, you can refer to this query.

SCENARIO	SUGGESTED SETTINGS
Full load from large table, without physical partitions, while with an integer or datetime column for data partitioning.	**Partition options**: Dynamic range partition. **Partition column** (optional): Specify the column used to partition data. If not specified, the primary key column is used. **Partition upper bound** and **partition lower bound** (optional): Specify if you want to determine the partition stride. This is not for filtering the rows in table, all rows in the table will be partitioned and copied. If not specified, copy activity auto detects the values and it can take long time depending on MIN and MAX values. It is recommended to provide upper bound and lower bound. For example, if your partition column "ID" has values range from 1 to 100, and you set the lower bound as 20 and the upper bound as 80, with parallel copy as 4, the service retrieves data by 4 partitions - IDs in range <=20, [21, 50], [51, 80], and >=81, respectively.

Load a large amount of data by using a custom query, without physical partitions, while with an integer or date/datetime column for data partitioning.	**Partition options**: Dynamic range partition.
	Query:
	`SELECT * FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	.
	Partition column: Specify the column used to partition data.
	Partition upper bound and **partition lower bound** (optional): Specify if you want to determine the partition stride. This is not for filtering the rows in table, all rows in the query result will be partitioned and copied. If not specified, copy activity auto detect the value.
	
	During execution, the service replaces `?AdfRangePartitionColumnName` with the actual column name and value ranges for each partition, and sends to Amazon RDS for SQL Server.
	For example, if your partition column "ID" has values range from 1 to 100, and you set the lower bound as 20 and the upper bound as 80, with parallel copy as 4, the service retrieves data by 4 partitions- IDs in range <=20, [21, 50], [51, 80], and >=81, respectively.
	
	Here are more sample queries for different scenarios:
	1. Query the whole table:
	`SELECT * FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition`
	2. Query from a table with column selection and additional where-clause filters:
	`SELECT <column_list> FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	3. Query with subqueries:
	`SELECT <column_list> FROM (<your_sub_query>) AS T WHERE ?AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	4. Query with partition in subquery:
	`SELECT <column_list> FROM (SELECT <your_sub_query_column_list> FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition) AS T`

Best practices to load data with partition option:

1. Choose distinctive column as partition column (like primary key or unique key) to avoid data skew.
2. If the table has built-in partition, use partition option "Physical partitions of table" to get better performance.
3. If you use Azure Integration Runtime to copy data, you can set larger "Data Integration Units (DIU)" (>4) to utilize more computing resource. Check the applicable scenarios there.
4. "Degree of copy parallelism" control the partition numbers, setting this number too large sometime hurts the performance, recommend setting this number as (DIU or number of Self-hosted IR nodes) * (2 to 4).

Example: full load from large table with physical partitions

```
"source": {
    "type": "AmazonRdsForSqlServerSource",
    "partitionOption": "PhysicalPartitionsOfTable"
}
```

Example: query with dynamic range partition

```
"source": {
    "type": "AmazonRdsForSqlServerSource",
    "query":"SELECT * FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition AND
<your_additional_where_clause>",
    "partitionOption": "DynamicRange",
    "partitionSettings": {
        "partitionColumnName": "<partition_column_name>",
        "partitionUpperBound": "<upper_value_of_partition_column (optional) to decide the partition stride,
not as data filter>",
        "partitionLowerBound": "<lower_value_of_partition_column (optional) to decide the partition stride,
not as data filter>"
    }
}
```

Sample query to check physical partition

```
SELECT DISTINCT s.name AS SchemaName, t.name AS TableName, pf.name AS PartitionFunctionName, c.name AS
ColumnName, iif(pf.name is null, 'no', 'yes') AS HasPartition
FROM sys.tables AS t
LEFT JOIN sys.objects AS o ON t.object_id = o.object_id
LEFT JOIN sys.schemas AS s ON o.schema_id = s.schema_id
LEFT JOIN sys.indexes AS i ON t.object_id = i.object_id
LEFT JOIN sys.index_columns AS ic ON ic.partition_ordinal > 0 AND ic.index_id = i.index_id AND ic.object_id
= t.object_id
LEFT JOIN sys.columns AS c ON c.object_id = ic.object_id AND c.column_id = ic.column_id
LEFT JOIN sys.partition_schemes ps ON i.data_space_id = ps.data_space_id
LEFT JOIN sys.partition_functions pf ON pf.function_id = ps.function_id
WHERE s.name='[your schema]' AND t.name = '[your table name]'
```

If the table has physical partition, you would see "HasPartition" as "yes" like the following.

SchemaName	TableName	PartitionFunctionName	ColumnName	HasPartition
dbo	PerfWithPartition	RangePF1	Id	yes

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity

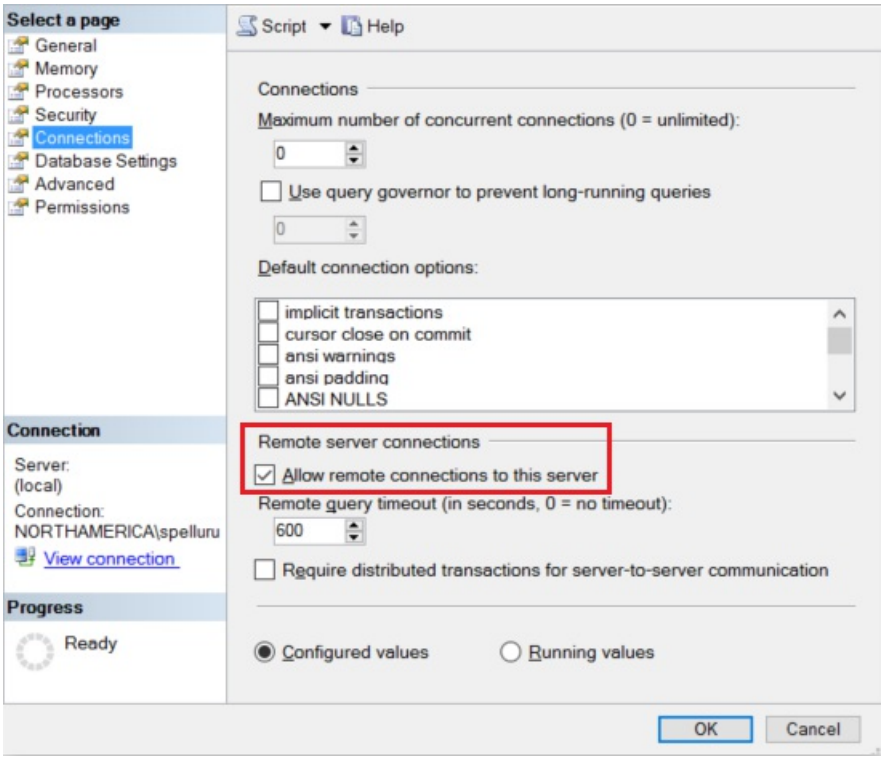
Using Always Encrypted

When you copy data from/to Amazon RDS for SQL Server with Always Encrypted, follow below steps:

1. Store the Column Master Key (CMK) in an Azure Key Vault. Learn more on how to configure Always Encrypted by using Azure Key Vault
2. Make sure to grant access to the key vault where the Column Master Key (CMK) is stored. Refer to this article for required permissions.
3. Create linked service to connect to your SQL database and enable 'Always Encrypted' function by using either managed identity or service principal.

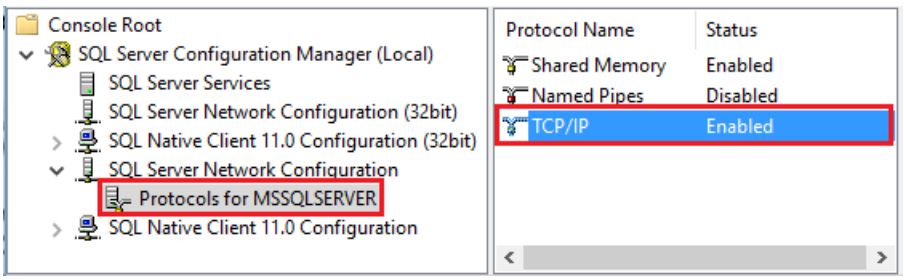
Troubleshoot connection issues

1. Configure your Amazon RDS for SQL Server instance to accept remote connections. Start **Amazon RDS for SQL Server Management Studio**, right-click **server**, and select **Properties**. Select **Connections** from the list, and select the **Allow remote connections to this server** check box.



For detailed steps, see Configure the remote access server configuration option.

2. Start **Amazon RDS for SQL Server Configuration Manager**. Expand **Amazon RDS for SQL Server Network Configuration** for the instance you want, and select **Protocols for MSSQLSERVER**. Protocols appear in the right pane. Enable TCP/IP by right-clicking **TCP/IP** and selecting **Enable**.



For more information and alternate ways of enabling TCP/IP protocol, see Enable or disable a server network protocol.

3. In the same window, double-click **TCP/IP** to launch the **TCP/IP Properties** window.
4. Switch to the **IP Addresses** tab. Scroll down to see the **IPAll** section. Write down the **TCP Port**. The default is **1433**.
5. Create a **rule for the Windows Firewall** on the machine to allow incoming traffic through this port.
6. **Verify connection**: To connect to Amazon RDS for SQL Server by using a fully qualified name, use Amazon RDS for SQL Server Management Studio from a different machine. An example is `"<machine>.<domain>.corp.<company>.com,1433"`.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from Amazon Redshift using Azure Data Factory or Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from an Amazon Redshift. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Amazon Redshift connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks by the copy activity, see the Supported data stores table.

Specifically, this Amazon Redshift connector supports retrieving data from Redshift using query or built-in Redshift UNLOAD support.

> **TIP**
> To achieve the best performance when copying large amounts of data from Redshift, consider using the built-in Redshift UNLOAD through Amazon S3. See Use UNLOAD to copy data from Amazon Redshift section for details.

Prerequisites

- If you are copying data to an on-premises data store using Self-hosted Integration Runtime, grant Integration Runtime (use IP address of the machine) the access to Amazon Redshift cluster. See Authorize access to the cluster for instructions.
- If you are copying data to an Azure data store, see Azure Data Center IP Ranges for the Compute IP address and SQL ranges used by the Azure data centers.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK

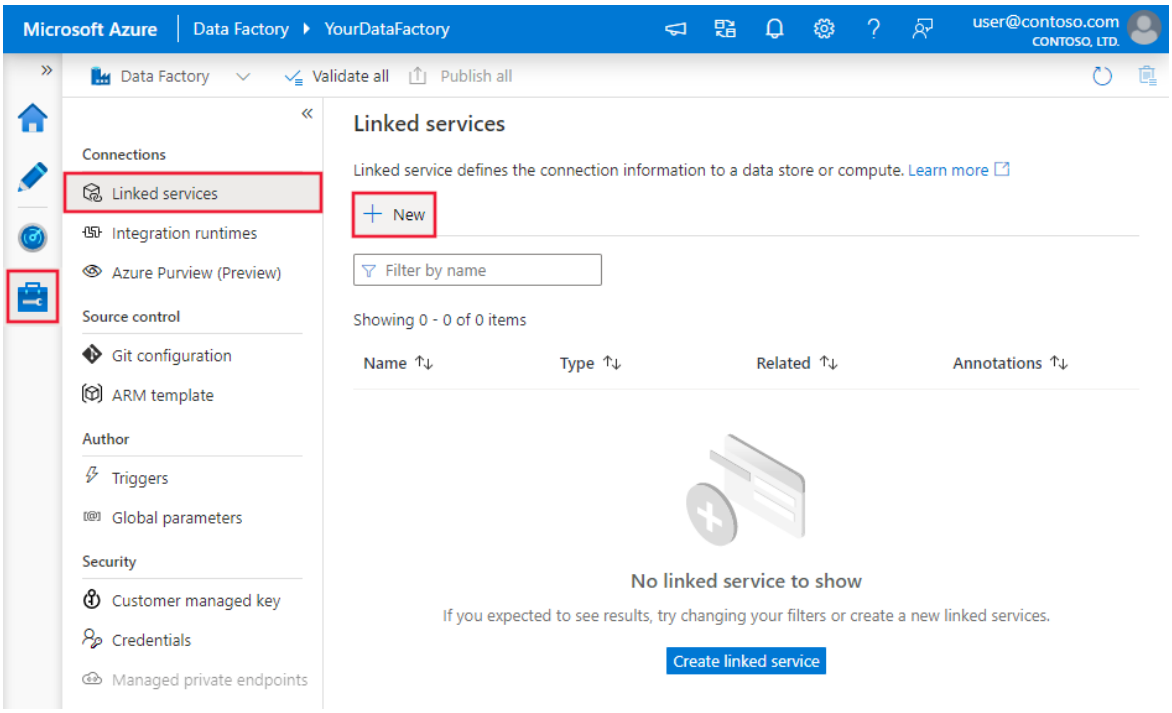
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Amazon Redshift using UI

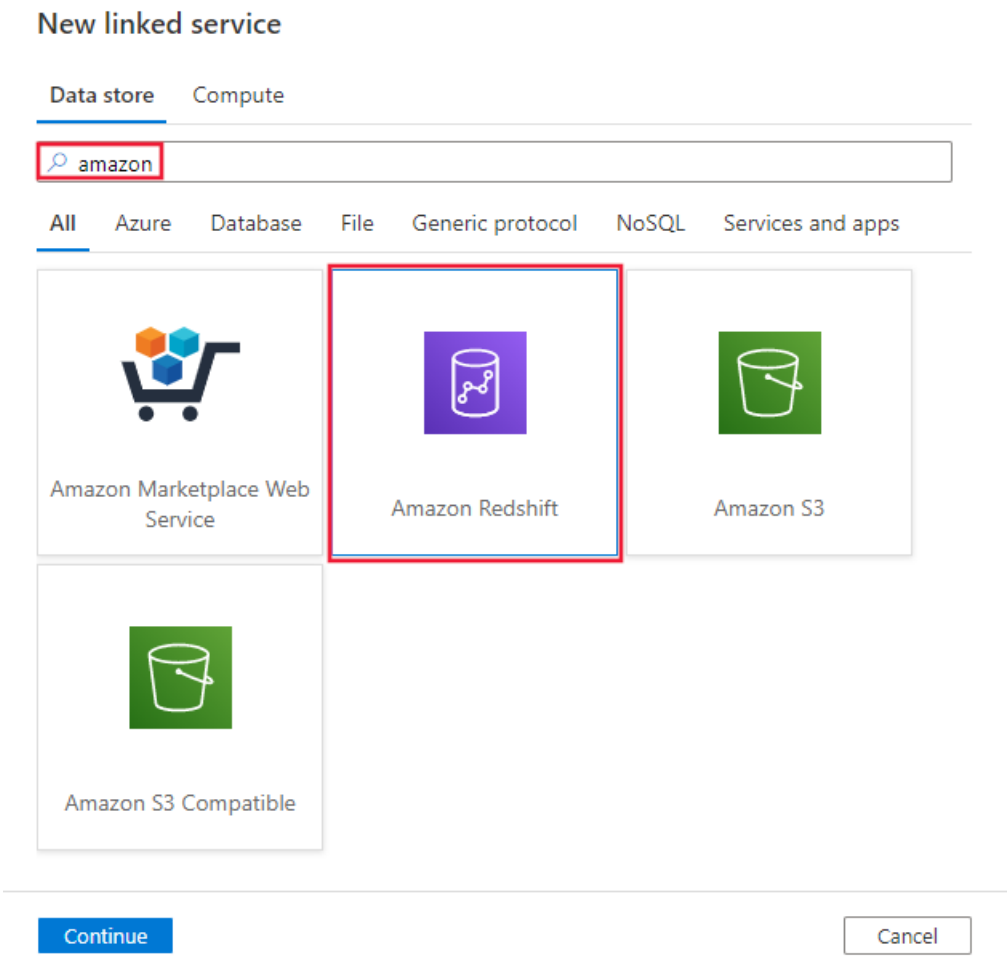
Use the following steps to create a linked service to Amazon Redshift in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

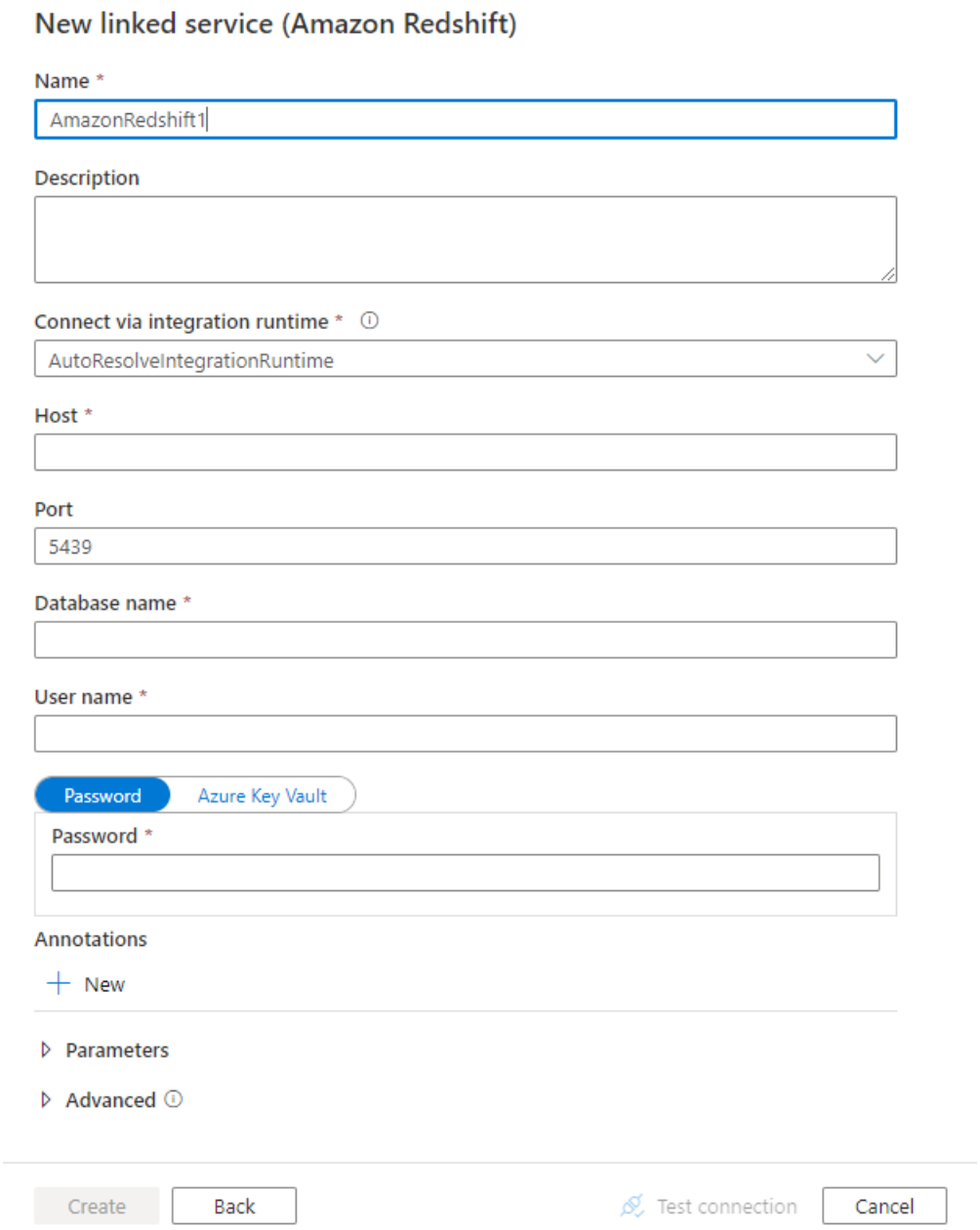
   - Azure Data Factory
   - Azure Synapse



2. Search for Amazon and select the Amazon Redshift connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Amazon Redshift connector.

Linked service properties

The following properties are supported for Amazon Redshift linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **AmazonRedshift**	Yes
server	IP address or host name of the Amazon Redshift server.	Yes
port	The number of the TCP port that the Amazon Redshift server uses to listen for client connections.	No, default is 5439

PROPERTY	DESCRIPTION	REQUIRED
database	Name of the Amazon Redshift database.	Yes
username	Name of user who has access to the database.	Yes
password	Password for the user account. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is located in private network). If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "AmazonRedshiftLinkedService",
    "properties":
    {
        "type": "AmazonRedshift",
        "typeProperties":
        {
            "server": "<server name>",
            "database": "<database name>",
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Amazon Redshift dataset.

To copy data from Amazon Redshift, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **AmazonRedshiftTable**	Yes
schema	Name of the schema.	No (if "query" in activity source is specified)

PROPERTY	DESCRIPTION	REQUIRED
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table with schema. This property is supported for backward compatibility. Use `schema` and `table` for new workload.	No (if "query" in activity source is specified)

Example

```
{
    "name": "AmazonRedshiftDataset",
    "properties":
    {
        "type": "AmazonRedshiftTable",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Amazon Redshift linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

If you were using `RelationalTable` typed dataset, it is still supported as-is, while you are suggested to use the new one going forward.

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Amazon Redshift source.

Amazon Redshift as source

To copy data from Amazon Redshift, set the source type in the copy activity to **AmazonRedshiftSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **AmazonRedshiftSource**	Yes
query	Use the custom query to read data. For example: select * from MyTable.	No (if "tableName" in dataset is specified)
redshiftUnloadSettings	Property group when using Amazon Redshift UNLOAD.	No
s3LinkedServiceName	Refers to an Amazon S3 to-be-used as an interim store by specifying a linked service name of "AmazonS3" type.	Yes if using UNLOAD
bucketName	Indicate the S3 bucket to store the interim data. If not provided, the service generates it automatically.	Yes if using UNLOAD

Example: Amazon Redshift source in copy activity using UNLOAD

```
"source": {
    "type": "AmazonRedshiftSource",
    "query": "<SQL query>",
    "redshiftUnloadSettings": {
        "s3LinkedServiceName": {
            "referenceName": "<Amazon S3 linked service>",
            "type": "LinkedServiceReference"
        },
        "bucketName": "bucketForUnload"
    }
}
```

Learn more on how to use UNLOAD to copy data from Amazon Redshift efficiently from next section.

Use UNLOAD to copy data from Amazon Redshift

UNLOAD is a mechanism provided by Amazon Redshift, which can unload the results of a query to one or more files on Amazon Simple Storage Service (Amazon S3). It is the way recommended by Amazon for copying large data set from Redshift.

Example: copy data from Amazon Redshift to Azure Synapse Analytics using UNLOAD, staged copy and PolyBase

For this sample use case, copy activity unloads data from Amazon Redshift to Amazon S3 as configured in "redshiftUnloadSettings", and then copy data from Amazon S3 to Azure Blob as specified in "stagingSettings", lastly use PolyBase to load data into Azure Synapse Analytics. All the interim format is handled by copy activity properly.

```
"activities":[
    {
        "name": "CopyFromAmazonRedshiftToSQLDW",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "AmazonRedshiftDataset",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "AzureSQLDWDataset",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AmazonRedshiftSource",
                "query": "select * from MyTable",
                "redshiftUnloadSettings": {
                    "s3LinkedServiceName": {
                        "referenceName": "AmazonS3LinkedService",
                        "type": "LinkedServiceReference"
                    },
                    "bucketName": "bucketForUnload"
                }
            },
            "sink": {
                "type": "SqlDWSink",
                "allowPolyBase": true
            },
            "enableStaging": true,
            "stagingSettings": {
                "linkedServiceName": "AzureStorageLinkedService",
                "path": "adfstagingcopydata"
            },
            "dataIntegrationUnits": 32
        }
    }
]
```

Data type mapping for Amazon Redshift

When copying data from Amazon Redshift, the following mappings are used from Amazon Redshift data types to interim data types used internally within the service. See Schema and data type mappings to learn about how copy activity maps the source schema and data type to the sink.

AMAZON REDSHIFT DATA TYPE	INTERIM SERVICE DATA TYPE
BIGINT	Int64
BOOLEAN	String
CHAR	String
DATE	DateTime
DECIMAL	Decimal

AMAZON REDSHIFT DATA TYPE	INTERIM SERVICE DATA TYPE
DOUBLE PRECISION	Double
INTEGER	Int32
REAL	Single
SMALLINT	Int16
TEXT	String
TIMESTAMP	DateTime
VARCHAR	String

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy and transform data in Amazon Simple Storage Service using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 18 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity to copy data from Amazon Simple Storage Service (Amazon S3), and use Data Flow to transform data in Amazon S3. To learn more, read the introductory articles for Azure Data Factory and Synapse Analytics.

> **TIP**
> To learn more about the data migration scenario from Amazon S3 to Azure Storage, see Migrate data from Amazon S3 to Azure Storage.

Supported capabilities

This Amazon S3 connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Mapping data flow (source/-)	①
Lookup activity	① ②
GetMetadata activity	① ②
Delete activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

Specifically, this Amazon S3 connector supports copying files as is or parsing files with the supported file formats and compression codecs. You can also choose to preserve file metadata during copy. The connector uses AWS Signature Version 4 to authenticate requests to S3.

> **TIP**
> If you want to copy data from *any S3-compatible storage provider*, see Amazon S3 Compatible Storage.

Required permissions

To copy data from Amazon S3, make sure you've been granted the following permissions for Amazon S3 object operations: `s3:GetObject` and `s3:GetObjectVersion`.

If you use Data Factory UI to author, additional `s3:ListAllMyBuckets` and `s3:ListBucket` / `s3:GetBucketLocation`

permissions are required for operations like testing connection to linked service and browsing from root. If you don't want to grant these permissions, you can choose "Test connection to file path" or "Browse from specified path" options from the UI.

For the full list of Amazon S3 permissions, see Specifying Permissions in a Policy on the AWS site.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

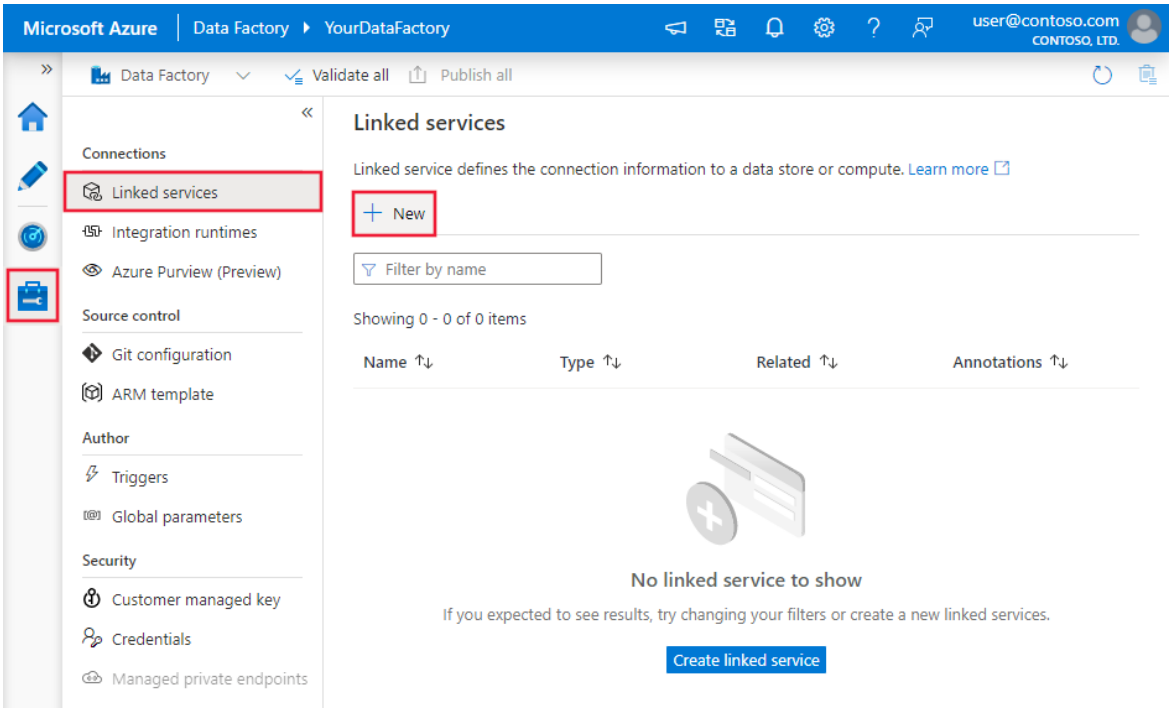
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create an Amazon Simple Storage Service (S3) linked service using UI

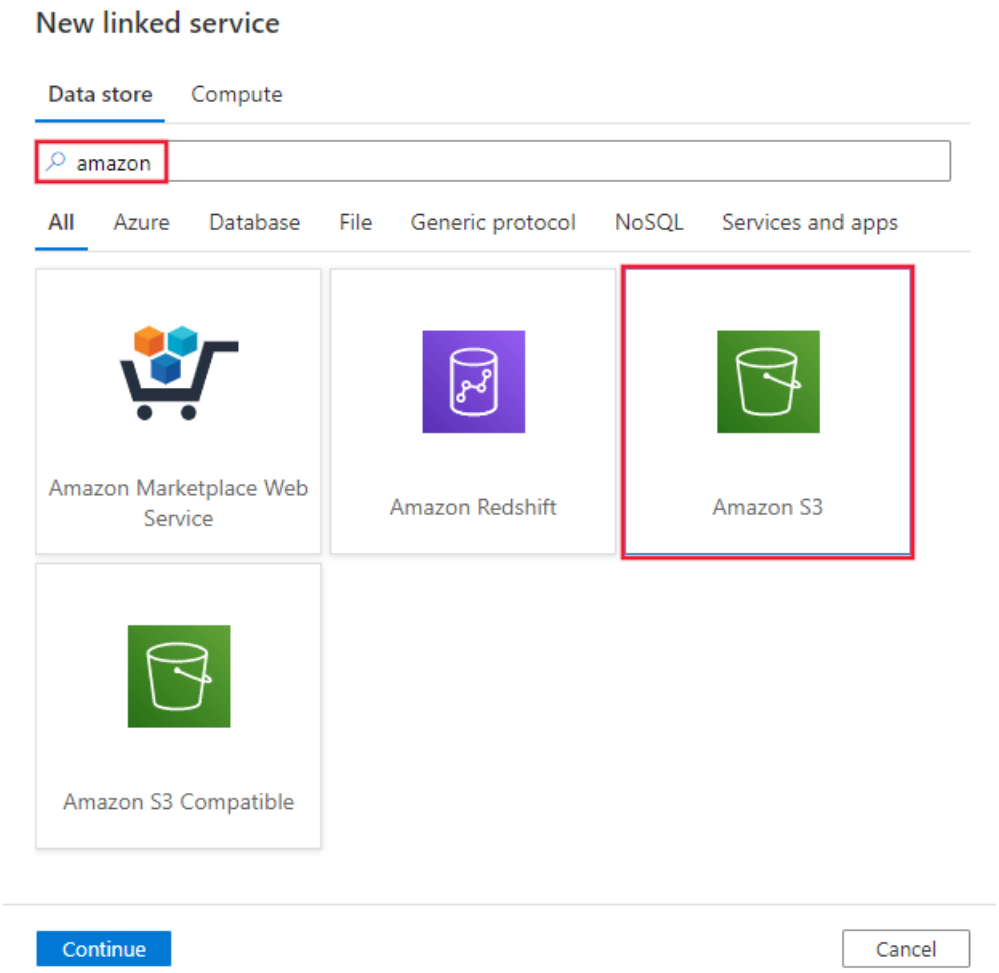
Use the following steps to create an Amazon S3 linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

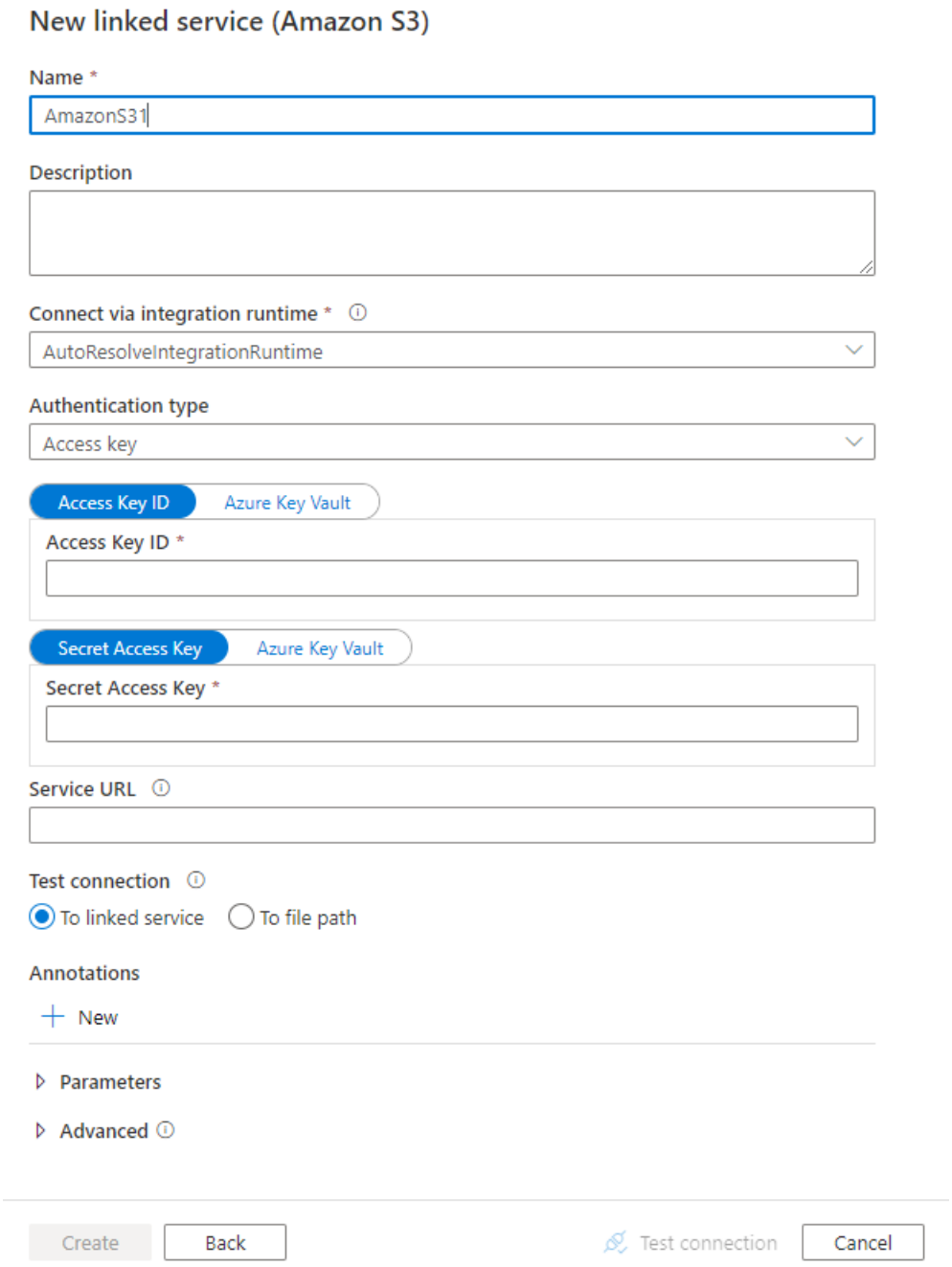
   - Azure Data Factory
   - Azure Synapse



2. Search for Amazon and select the Amazon S3 connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Amazon S3.

Linked service properties

The following properties are supported for an Amazon S3 linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **AmazonS3**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
authenticationType	Specify the authentication type used to connect to Amazon S3. You can choose to use access keys for an AWS Identity and Access Management (IAM) account, or temporary security credentials. Allowed values are: `AccessKey` (default) and `TemporarySecurityCredentials`.	No
accessKeyId	ID of the secret access key.	Yes
secretAccessKey	The secret access key itself. Mark this field as a **SecureString** to store it securely, or reference a secret stored in Azure Key Vault.	Yes
sessionToken	Applicable when using temporary security credentials authentication. Learn how to request temporary security credentials from AWS. Note AWS temporary credential expires between 15 minutes to 36 hours based on settings. Make sure your credential is valid when activity executes, especially for operationalized workload - for example, you can refresh it periodically and store it in Azure Key Vault. Mark this field as a **SecureString** to store it securely, or reference a secret stored in Azure Key Vault.	No
serviceUrl	Specify the custom S3 endpoint `https://<service url>`. Change it only if you want to try a different service endpoint or want to switch between https and http.	No
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

Example: using access key authentication

```
{
    "name": "AmazonS3LinkedService",
    "properties": {
        "type": "AmazonS3",
        "typeProperties": {
            "accessKeyId": "<access key id>",
            "secretAccessKey": {
                "type": "SecureString",
                "value": "<secret access key>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: using temporary security credential authentication

```
{
    "name": "AmazonS3LinkedService",
    "properties": {
        "type": "AmazonS3",
        "typeProperties": {
            "authenticationType": "TemporarySecurityCredentials",
            "accessKeyId": "<access key id>",
            "secretAccessKey": {
                "type": "SecureString",
                "value": "<secret access key>"
            },
            "sessionToken": {
                "type": "SecureString",
                "value": "<session token>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Amazon S3 under `location` settings in a format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property under `location` in a dataset must be set to **AmazonS3Location**.	Yes
bucketName	The S3 bucket name.	Yes
folderPath	The path to the folder under the given bucket. If you want to use a wildcard to filter the folder, skip this setting and specify that in the activity source settings.	No
fileName	The file name under the given bucket and folder path. If you want to use a wildcard to filter files, skip this setting and specify that in the activity source settings.	No
version	The version of the S3 object, if S3 versioning is enabled. If it's not specified, the latest version will be fetched.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<Amazon S3 linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AmazonS3Location",
                "bucketName": "bucketname",
                "folderPath": "folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties that the Amazon S3 source supports.

Amazon S3 as a source type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format

- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Amazon S3 under `storeSettings` settings in a format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property under `storeSettings` must be set to **AmazonS3ReadSettings**.	Yes
Locate the files to copy:		
OPTION 1: static path	Copy from the given bucket or folder/file path specified in the dataset. If you want to copy all files from a bucket or folder, additionally specify `wildcardFileName` as `*`.	
OPTION 2: S3 prefix - prefix	Prefix for the S3 key name under the given bucket configured in a dataset to filter source S3 files. S3 keys whose names start with `bucket_in_dataset/this_prefix` are selected. It utilizes S3's service-side filter, which provides better performance than a wildcard filter. When you use prefix and choose to copy to file-based sink with preserving hierarchy, note the sub-path after the last "/" in prefix will be preserved. For example, you have source `bucket/folder/subfolder/file.txt`, and configure prefix as `folder/sub`, then the preserved file path is `subfolder/file.txt`.	No
OPTION 3: wildcard - wildcardFolderPath	The folder path with wildcard characters under the given bucket configured in a dataset to filter source folders. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your folder name has a wildcard or this escape character inside. See more examples in Folder and file filter examples.	No

PROPERTY	DESCRIPTION	REQUIRED
OPTION 3: wildcard - wildcardFileName	The file name with wildcard characters under the given bucket and folder path (or wildcard folder path) to filter source files. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your file name has a wildcard or this escape character inside. See more examples in Folder and file filter examples.	Yes
OPTION 4: a list of files - fileListPath	Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset. When you're using this option, do not specify a file name in the dataset. See more examples in File list examples.	No
Additional settings:		
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when **recursive** is set to **true** and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**. This property doesn't apply when you configure `fileListPath`.	No
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	Files are filtered based on the attribute: last modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to a UTC time zone in the format of "2018-12-01T05:00:00Z". The properties can be **NULL**, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is **NULL**, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is **NULL**, the files whose last modified attribute is less than the datetime value will be selected. This property doesn't apply when you configure `fileListPath`.	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. - When you use prefix, partition root path is sub-path before the last "/". For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromAmazonS3",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "AmazonS3ReadSettings",
                    "recursive": true,
                    "wildcardFolderPath": "myfolder*A",
                    "wildcardFileName": "*.csv"
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

BUCKET	KEY	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
bucket	`Folder*/*`	false	bucket FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv

BUCKET	KEY	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
bucket	`Folder*/*`	true	bucket FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv
bucket	`Folder*/*.csv`	false	bucket FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
bucket	`Folder*/*.csv`	true	bucket FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This section describes the resulting behavior of using a file list path in a Copy activity source.

Assume that you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	CONFIGURATION
bucket FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In dataset:** - Bucket: `bucket` - Folder path: `FolderA` **In Copy activity source:** - File list path: `bucket/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line, with the relative path to the path configured in the dataset.

Preserve metadata during copy

When you copy files from Amazon S3 to Azure Data Lake Storage Gen2 or Azure Blob storage, you can choose to preserve the file metadata along with data. Learn more from Preserve metadata.

Mapping data flow properties

When you're transforming data in mapping data flows, you can read files from Amazon S3 in the following formats:

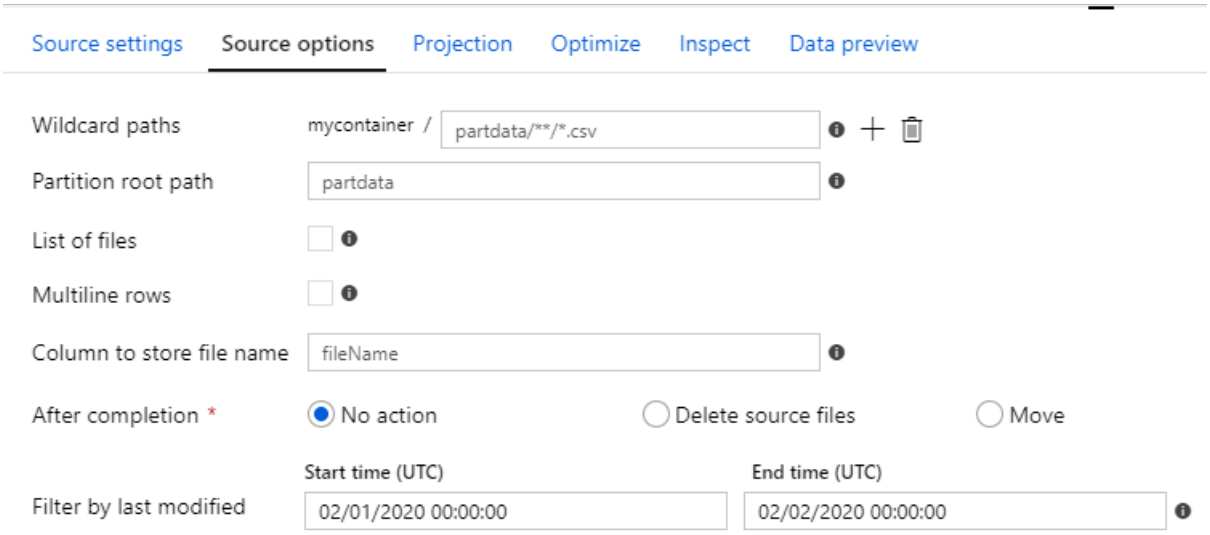
- Avro
- Delimited text
- Delta
- Excel
- JSON
- Parquet

Format specific settings are located in the documentation for that format. For more information, see Source transformation in mapping data flow.

> **NOTE**
> The Amazon S3 source transformation is only supported in the **Azure Synapse Analytics** workspace now.

Source transformation

In source transformation, you can read from a container, folder, or individual file in Amazon S3. Use the **Source options** tab to manage how the files are read.



Wildcard paths: Using a wildcard pattern will instruct the service to loop through each matching folder and file in a single source transformation. This is an effective way to process multiple files within a single flow. Add multiple wildcard matching patterns with the plus sign that appears when you hover over your existing wildcard pattern.

From your source container, choose a series of files that match a pattern. Only a container can be specified in the dataset. Your wildcard path must therefore also include your folder path from the root folder.

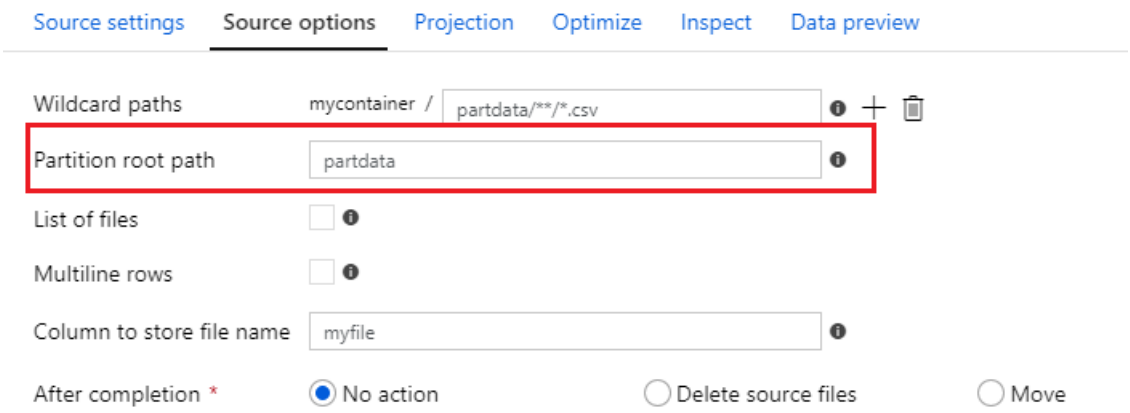
Wildcard examples:

- `*` Represents any set of characters.
- `**` Represents recursive directory nesting.

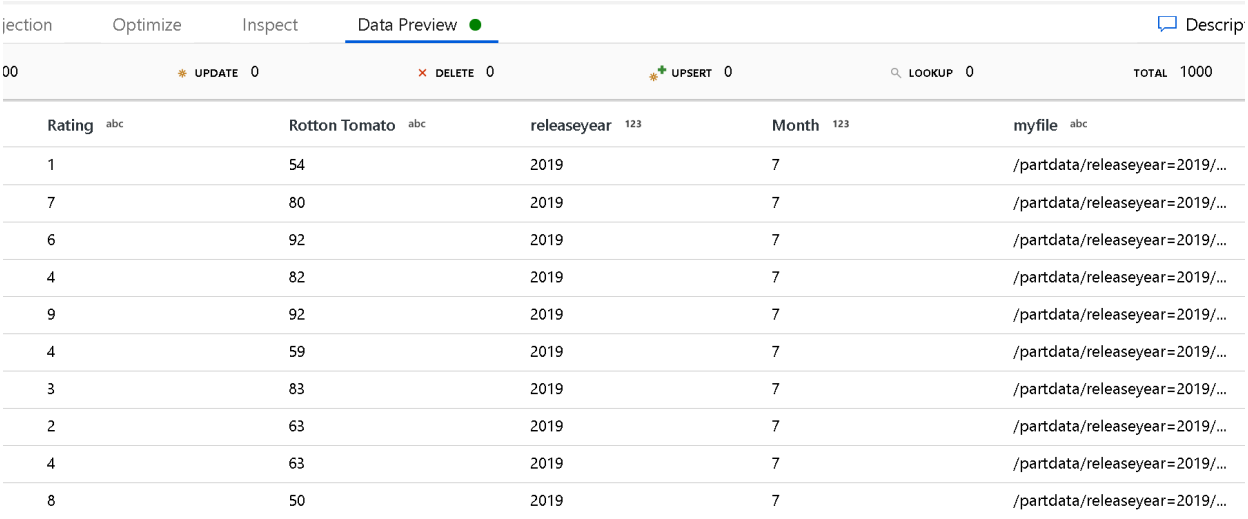
- `?` Replaces one character.
- `[]` Matches one or more characters in the brackets.
- `/data/sales/**/*.csv` Gets all .csv files under /data/sales.
- `/data/sales/20??/**/` Gets all files in the 20th century.
- `/data/sales/*/*/*.csv` Gets .csv files two levels under /data/sales.
- `/data/sales/2004/*/12/[XY]1?.csv` Gets all .csv files in December 2004 starting with X or Y prefixed by a two-digit number.

Partition root path: If you have partitioned folders in your file source with a `key=value` format (for example, `year=2019`), then you can assign the top level of that partition folder tree to a column name in your data flow's data stream.

First, set a wildcard to include all paths that are the partitioned folders plus the leaf files that you want to read.



Use the **Partition root path** setting to define what the top level of the folder structure is. When you view the contents of your data via a data preview, you'll see that the service will add the resolved partitions found in each of your folder levels.



Rating abc	Rotton Tomato abc	releaseyear 123	Month 123	myfile abc
1	54	2019	7	/partdata/releaseyear=2019/...
7	80	2019	7	/partdata/releaseyear=2019/...
6	92	2019	7	/partdata/releaseyear=2019/...
4	82	2019	7	/partdata/releaseyear=2019/...
9	92	2019	7	/partdata/releaseyear=2019/...
4	59	2019	7	/partdata/releaseyear=2019/...
3	83	2019	7	/partdata/releaseyear=2019/...
2	63	2019	7	/partdata/releaseyear=2019/...
4	63	2019	7	/partdata/releaseyear=2019/...
8	50	2019	7	/partdata/releaseyear=2019/...

List of files: This is a file set. Create a text file that includes a list of relative path files to process. Point to this text file.

Column to store file name: Store the name of the source file in a column in your data. Enter a new column name here to store the file name string.

After completion: Choose to do nothing with the source file after the data flow runs, delete the source file, or move the source file. The paths for the move are relative.

To move source files to another location post-processing, first select "Move" for file operation. Then, set the "from" directory. If you're not using any wildcards for your path, then the "from" setting will be the same folder as your source folder.

If you have a source path with wildcard, your syntax will look like this:

`/data/sales/20??/**/*.csv`

You can specify "from" as:

`/data/sales`

And you can specify "to" as:

`/backup/priorSales`

In this case, all files that were sourced under `/data/sales` are moved to `/backup/priorSales`.

> **NOTE**
> File operations run only when you start the data flow from a pipeline run (a pipeline debug or execution run) that uses the Execute Data Flow activity in a pipeline. File operations *do not* run in Data Flow debug mode.

Filter by last modified: You can filter which files you process by specifying a date range of when they were last modified. All datetimes are in UTC.

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity.

Delete activity properties

To learn details about the properties, check Delete activity.

Legacy models

> **NOTE**
> The following models are still supported as is for backward compatibility. We suggest that you use the new model mentioned earlier. The authoring UI has switched to generating the new model.

Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the dataset must be set to **AmazonS3Object**.	Yes
bucketName	The S3 bucket name. The wildcard filter is not supported.	Yes for the Copy or Lookup activity, no for the GetMetadata activity

PROPERTY	DESCRIPTION	REQUIRED

key	The name or wildcard filter of the S3 object key under the specified bucket. Applies only when the **prefix** property is not specified.
	
	The wildcard filter is supported for both the folder part and the file name part. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character).
	- Example 1:
	`"key": "rootfolder/subfolder/*.csv"`
	- Example 2:
	`"key": "rootfolder/subfolder/???20180427.txt"`
	See more example in Folder and file filter examples. Use `^` to escape if your actual folder or file name has a wildcard or this escape character inside.
		No

prefix	Prefix for the S3 object key. Objects whose keys start with this prefix are selected. Applies only when the **key** property is not specified.	No
version	The version of the S3 object, if S3 versioning is enabled. If a version is not specified, the latest version will be fetched.	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	Files are filtered based on the attribute: last modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z". Be aware that enabling this setting will affect the overall performance of data movement when you want to filter huge amounts of files. The properties can be **NULL**, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is **NULL**, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is NULL, the files whose last modified attribute is less than the datetime value will be selected.	No
modifiedDatetimeEnd	Files are filtered based on the attribute: last modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z". Be aware that enabling this setting will affect the overall performance of data movement when you want to filter huge amounts of files. The properties can be **NULL**, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is **NULL**, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is **NULL**, the files whose last modified attribute is less than the datetime value will be selected.	No

PROPERTY	DESCRIPTION	REQUIRED
format	If you want to copy files as is between file-based stores (binary copy), skip the format section in both input and output dataset definitions. If you want to parse or generate files with a specific format, the following file format types are supported: **TextFormat**, **JsonFormat**, **AvroFormat**, **OrcFormat**, **ParquetFormat**. Set the **type** property under **format** to one of these values. For more information, see the Text format, JSON format, Avro format, Orc format, and Parquet format sections.	No (only for binary copy scenario)
compression	Specify the type and level of compression for the data. For more information, see Supported file formats and compression codecs. Supported types are **GZip**, **Deflate**, **BZip2**, and **ZipDeflate**. Supported levels are **Optimal** and **Fastest**.	No

TIP

To copy all files under a folder, specify **bucketName** for the bucket and **prefix** for the folder part.

To copy a single file with a given name, specify **bucketName** for the bucket and **key** for the folder part plus file name.

To copy a subset of files under a folder, specify **bucketName** for the bucket and **key** for the folder part plus wildcard filter.

Example: using prefix

```
{
    "name": "AmazonS3Dataset",
    "properties": {
        "type": "AmazonS3Object",
        "linkedServiceName": {
            "referenceName": "<Amazon S3 linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "bucketName": "testbucket",
            "prefix": "testFolder/test",
            "modifiedDatetimeStart": "2018-12-01T05:00:00Z",
            "modifiedDatetimeEnd": "2018-12-01T06:00:00Z",
            "format": {
                "type": "TextFormat",
                "columnDelimiter": ",",
                "rowDelimiter": "\n"
            },
            "compression": {
                "type": "GZip",
                "level": "Optimal"
            }
        }
    }
}
```

Example: using key and version (optional)

```
{
    "name": "AmazonS3Dataset",
    "properties": {
        "type": "AmazonS3",
        "linkedServiceName": {
            "referenceName": "<Amazon S3 linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "bucketName": "testbucket",
            "key": "testFolder/testfile.csv.gz",
            "version": "XXXXXXXXXczm0CJajYkHf0_k6LhBmkcL",
            "format": {
                "type": "TextFormat",
                "columnDelimiter": ",",
                "rowDelimiter": "\n"
            },
            "compression": {
                "type": "GZip",
                "level": "Optimal"
            }
        }
    }
}
```

Legacy source model for the Copy activity

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the Copy activity source must be set to **FileSystemSource**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when **recursive** is set to **true** and the sink is a file-based store, an empty folder or subfolder will not be copied or created at the sink. Allowed values are **true** (default) and **false**.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromAmazonS3",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Amazon S3 input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "FileSystemSource",
                "recursive": true
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Next steps

For a list of data stores that the Copy activity supports as sources and sinks, see Supported data stores.

Copy data from Amazon S3 Compatible Storage by using Azure Data Factory or Synapse Analytics

8/19/2022 • 10 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to copy data from Amazon Simple Storage Service (Amazon S3) Compatible Storage. To learn more, read the introductory articles for Azure Data Factory and Synapse Analytics.

Supported capabilities

This Amazon S3 Compatible Storage connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②
GetMetadata activity	① ②
Delete activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

Specifically, this Amazon S3 Compatible Storage connector supports copying files as is or parsing files with the supported file formats and compression codecs. The connector uses AWS Signature Version 4 to authenticate requests to S3. You can use this Amazon S3 Compatible Storage connector to copy data from any S3-compatible storage provider. Specify the corresponding service URL in the linked service configuration.

Required permissions

To copy data from Amazon S3 Compatible Storage, make sure you've been granted the following permissions for Amazon S3 object operations: `s3:GetObject` and `s3:GetObjectVersion`.

If you use UI to author, additional `s3:ListAllMyBuckets` and `s3:ListBucket` / `s3:GetBucketLocation` permissions are required for operations like testing connection to linked service and browsing from root. If you don't want to grant these permissions, you can choose "Test connection to file path" or "Browse from specified path" options from the UI.

For the full list of Amazon S3 permissions, see Specifying Permissions in a Policy on the AWS site.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK

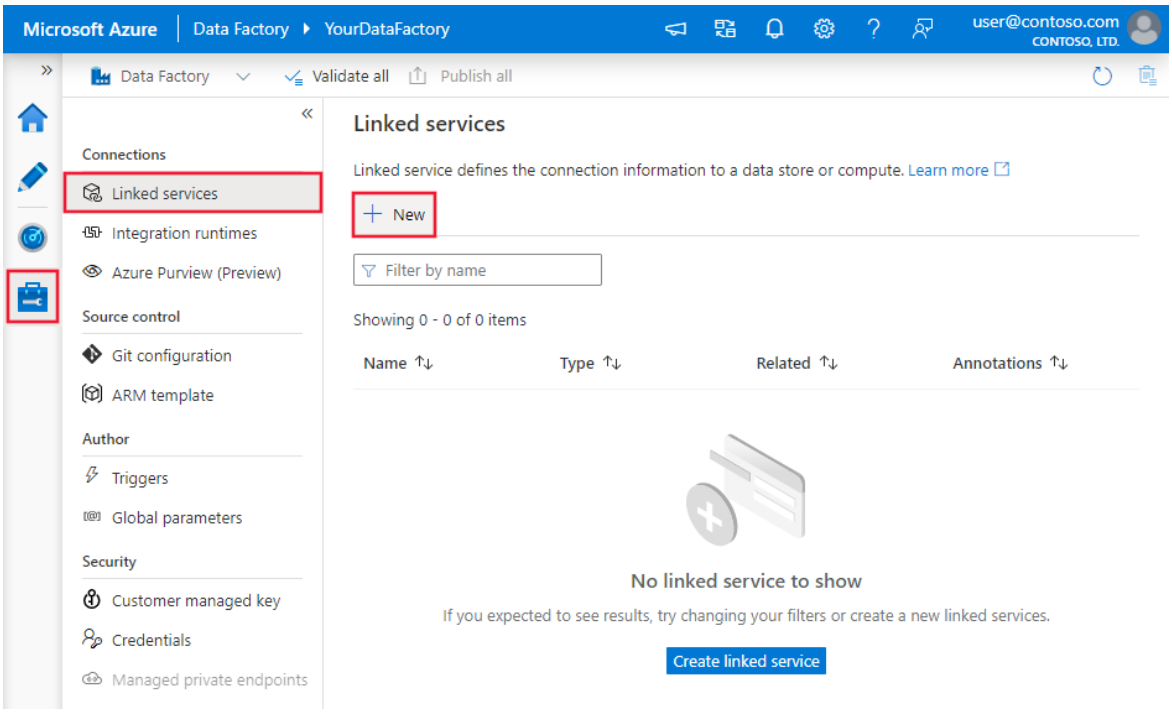
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Amazon S3 Compatible Storage using UI

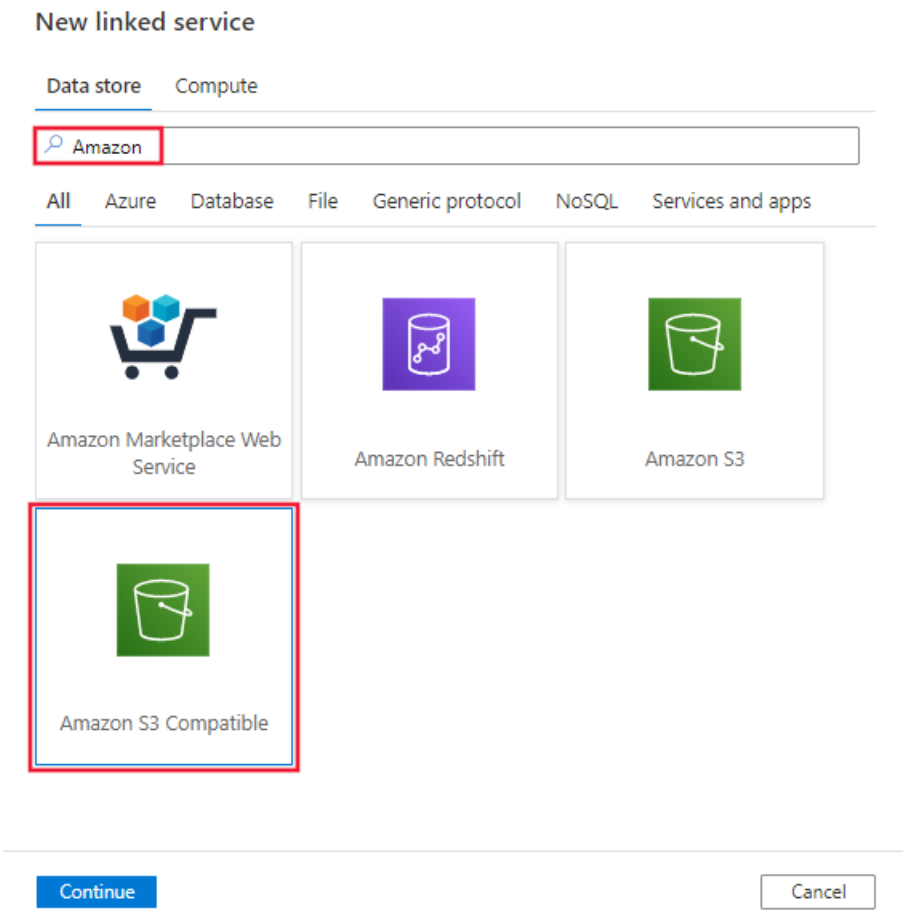
Use the following steps to create a linked service to Amazon S3 Compatible Storage in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

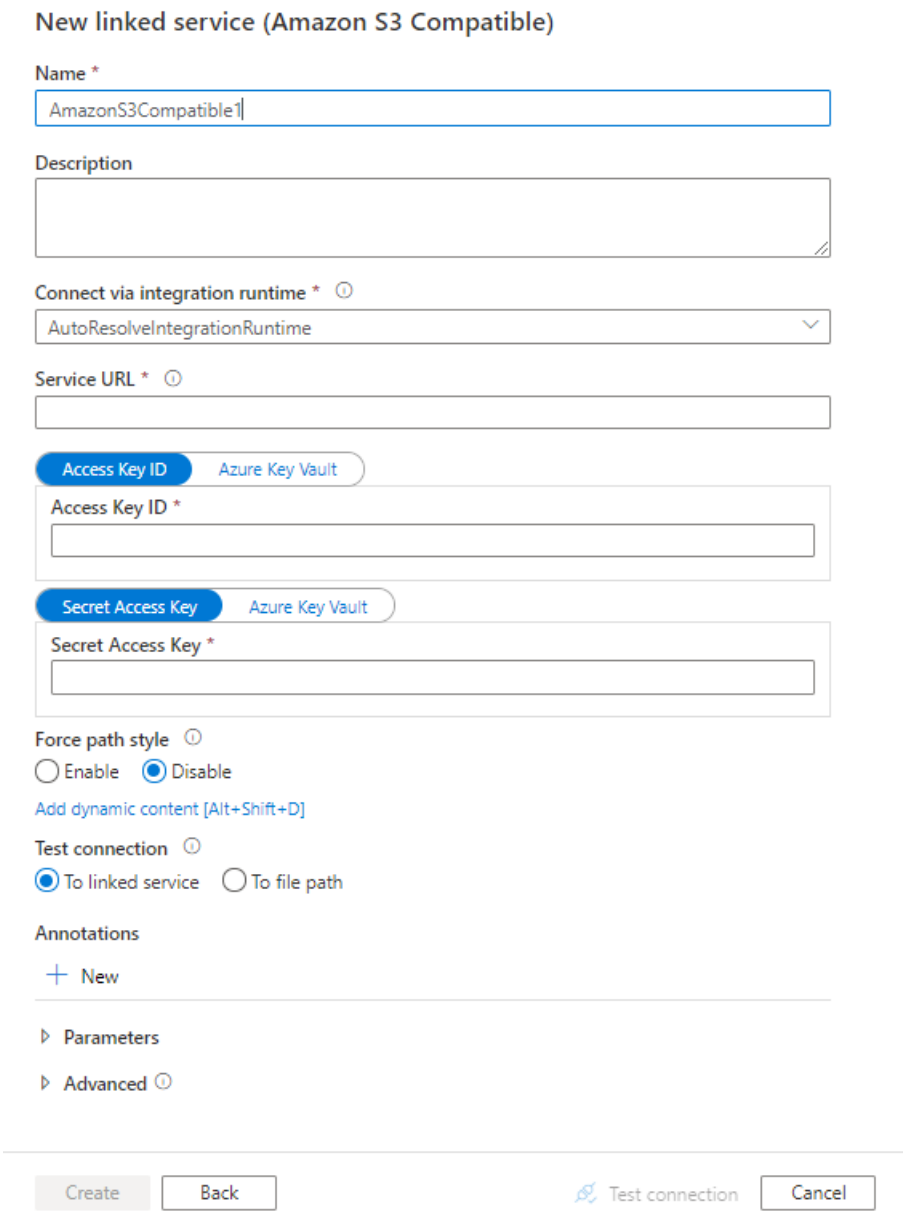
   - Azure Data Factory
   - Azure Synapse



2. Search for Amazon and select the Amazon S3 Compatible Storage connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to Amazon S3 Compatible Storage.

Linked service properties

The following properties are supported for an Amazon S3 Compatible linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **AmazonS3Compatible**.	Yes
accessKeyId	ID of the secret access key.	Yes
secretAccessKey	The secret access key itself. Mark this field as a **SecureString** to store it securely, or reference a secret stored in Azure Key Vault.	Yes

PROPERTY	DESCRIPTION	REQUIRED
serviceUrl	Specify the custom S3 endpoint `https://<service url>`.	No
forcePathStyle	Indicates whether to use S3 path-style access instead of virtual hosted-style access. Allowed values are: **false** (default), **true**. Check each data store's documentation on if path-style access is needed or not.	No
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

Example:

```
{
    "name": "AmazonS3CompatibleLinkedService",
    "properties": {
        "type": "AmazonS3Compatible",
        "typeProperties": {
            "accessKeyId": "<access key id>",
            "secretAccessKey": {
                "type": "SecureString",
                "value": "<secret access key>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Amazon S3 Compatible under `location` settings in a format-based

dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property under `location` in a dataset must be set to **AmazonS3CompatibleLocation**.	Yes
bucketName	The S3 Compatible Storage bucket name.	Yes
folderPath	The path to the folder under the given bucket. If you want to use a wildcard to filter the folder, skip this setting and specify that in the activity source settings.	No
fileName	The file name under the given bucket and folder path. If you want to use a wildcard to filter files, skip this setting and specify that in the activity source settings.	No
version	The version of the S3 Compatible Storage object, if S3 Compatible Storage versioning is enabled. If it's not specified, the latest version will be fetched.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<Amazon S3 Compatible Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AmazonS3CompatibleLocation",
                "bucketName": "bucketname",
                "folderPath": "folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties that the Amazon S3 Compatible Storage source supports.

Amazon S3 Compatible Storage as a source type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Amazon S3 Compatible Storage under `storeSettings` settings in a format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property under `storeSettings` must be set to **AmazonS3CompatibleReadSettings**.	Yes
Locate the files to copy:		
OPTION 1: static path	Copy from the given bucket or folder/file path specified in the dataset. If you want to copy all files from a bucket or folder, additionally specify `wildcardFileName` as `*`.	
OPTION 2: S3 Compatible Storage prefix - prefix	Prefix for the S3 Compatible Storage key name under the given bucket configured in a dataset to filter source S3 Compatible Storage files. S3 Compatible Storage keys whose names start with `bucket_in_dataset/this_prefix` are selected. It utilizes S3 Compatible Storage's service-side filter, which provides better performance than a wildcard filter. When you use prefix and choose to copy to file-based sink with preserving hierarchy, note the sub-path after the last "/" in prefix will be preserved. For example, you have source `bucket/folder/subfolder/file.txt`, and configure prefix as `folder/sub`, then the preserved file path is `subfolder/file.txt`.	No

PROPERTY	DESCRIPTION	REQUIRED
OPTION 3: wildcard - wildcardFolderPath	The folder path with wildcard characters under the given bucket configured in a dataset to filter source folders. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your folder name has a wildcard or this escape character inside. See more examples in Folder and file filter examples.	No
OPTION 3: wildcard - wildcardFileName	The file name with wildcard characters under the given bucket and folder path (or wildcard folder path) to filter source files. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your file name has a wildcard or this escape character inside. See more examples in Folder and file filter examples.	Yes
OPTION 4: a list of files - fileListPath	Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset. When you're using this option, do not specify a file name in the dataset. See more examples in File list examples.	No
Additional settings:		
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when **recursive** is set to **true** and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**. This property doesn't apply when you configure `fileListPath`.	No

PROPERTY	DESCRIPTION	REQUIRED
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No
modifiedDatetimeStart	Files are filtered based on the attribute: last modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to a UTC time zone in the format of "2018-12-01T05:00:00Z". The properties can be **NULL**, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is **NULL**, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is **NULL**, the files whose last modified attribute is less than the datetime value will be selected. This property doesn't apply when you configure `fileListPath`.	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. - When you use prefix, partition root path is sub-path before the last "/". For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromAmazonS3CompatibleStorage",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "AmazonS3CompatibleReadSettings",
                    "recursive": true,
                    "wildcardFolderPath": "myfolder*A",
                    "wildcardFileName": "*.csv"
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

BUCKET	KEY	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
bucket	`Folder*/*`	false	bucket FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv

BUCKET	KEY	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
bucket	`Folder*/*`	true	bucket FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv
bucket	`Folder*/*.csv`	false	bucket FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
bucket	`Folder*/*.csv`	true	bucket FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This section describes the resulting behavior of using a file list path in a Copy activity source.

Assume that you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	CONFIGURATION
bucket FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In dataset:** - Bucket: `bucket` - Folder path: `FolderA` **In Copy activity source:** - File list path: `bucket/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line, with the relative path to the path configured in the dataset.

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity.

Delete activity properties

To learn details about the properties, check Delete activity.

Next steps

For a list of data stores that the Copy activity supports as sources and sinks, see Supported data stores.

Transform data in Asana (Preview) using Azure Data Factory or Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Data Flow to transform data in Asana (Preview). To learn more, read the introductory article for Azure Data Factory or Azure Synapse Analytics.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Asana connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Mapping data flow (source/-)	①

① Azure integration runtime ② Self-hosted integration runtime

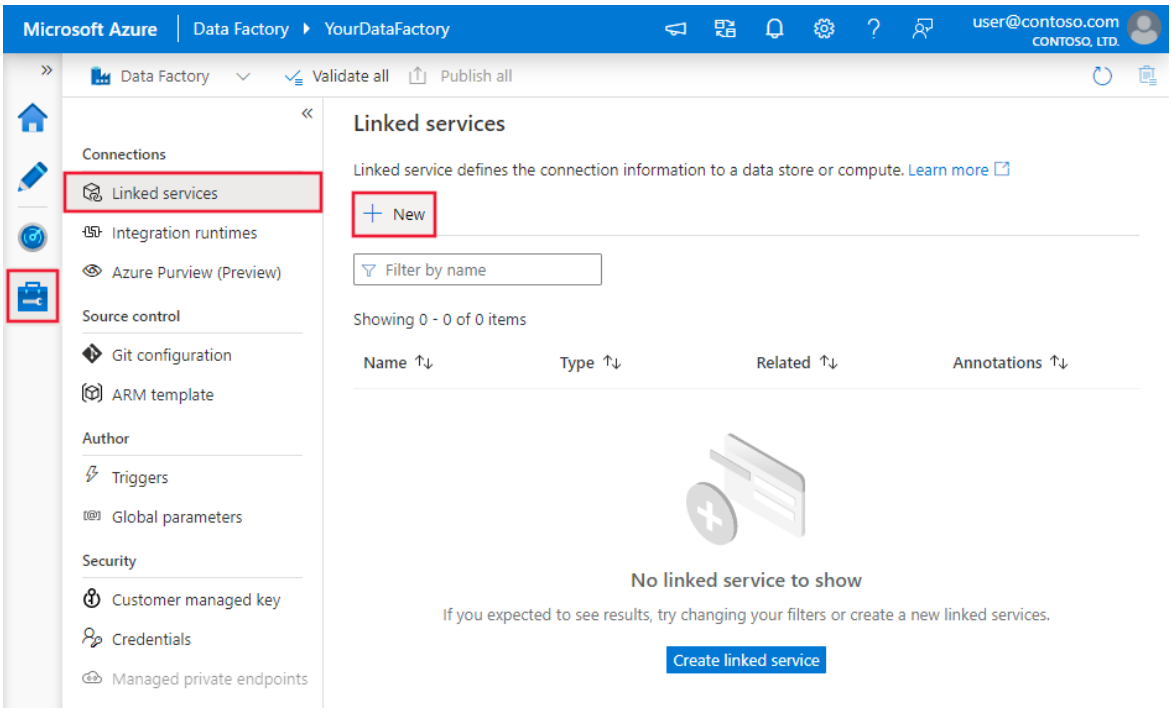
For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Create an Asana linked service using UI

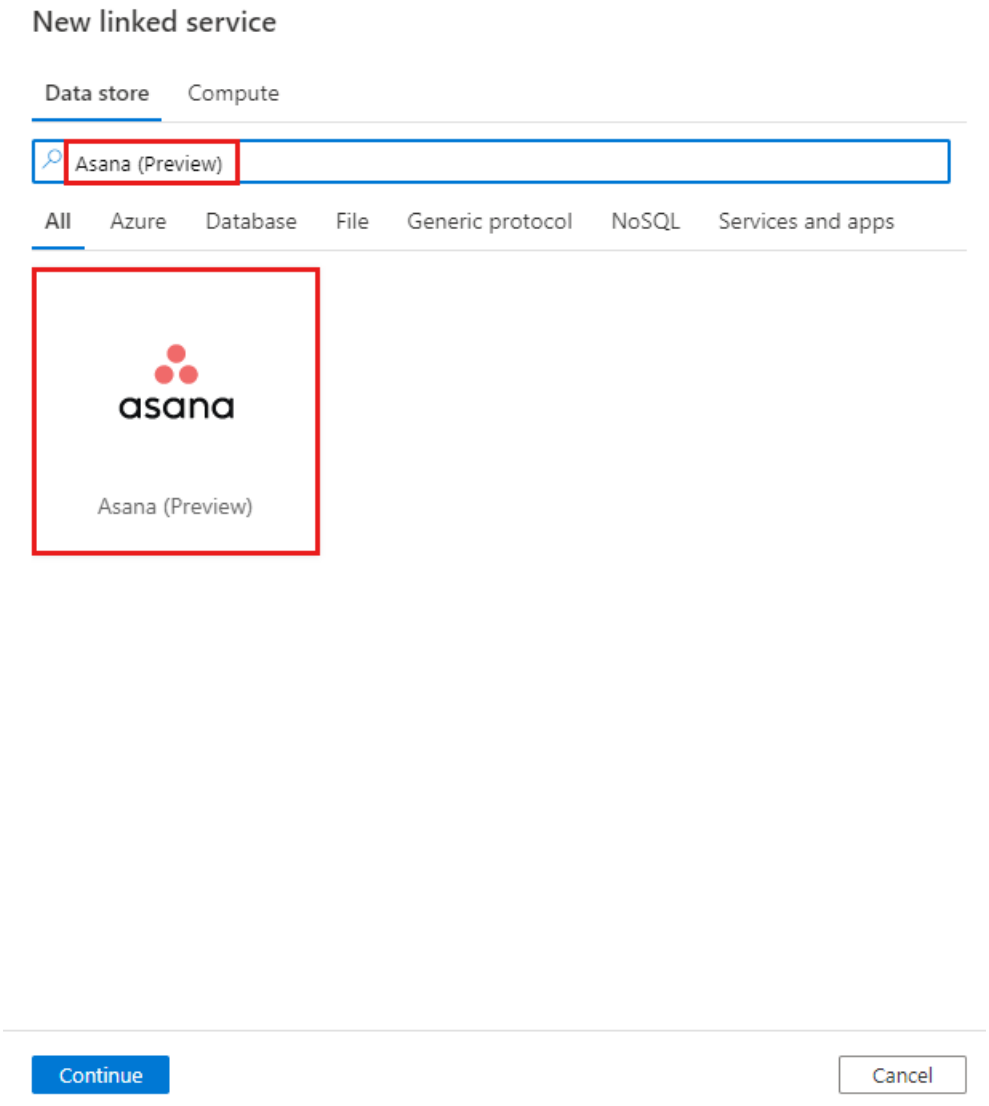
Use the following steps to create an Asana linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then select New:

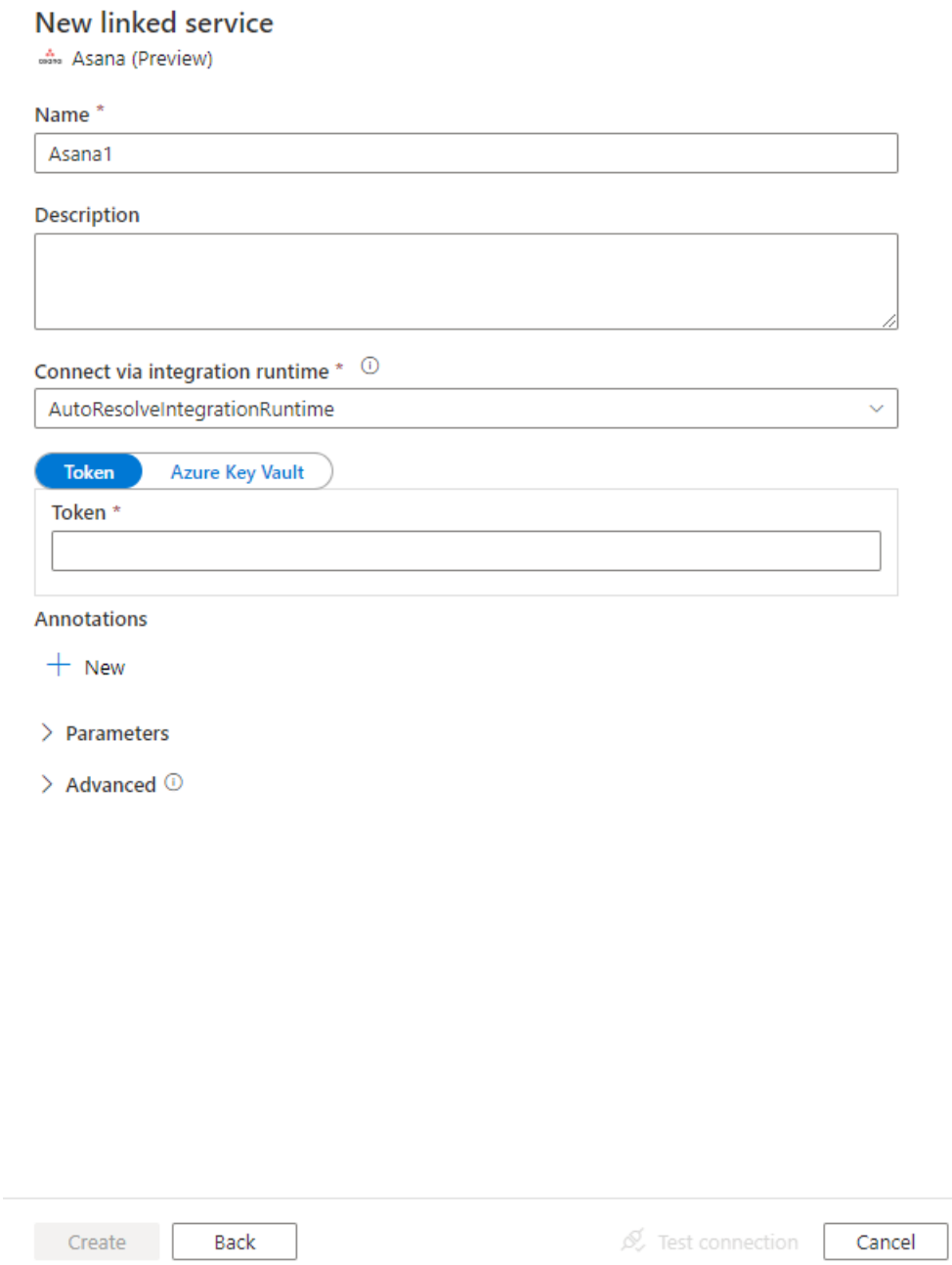
   - Azure Data Factory
   - Azure Synapse



2. Search for Asana (Preview) and select the Asana (Preview) connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide information about properties that are used to define Data Factory and Synapse pipeline entities specific to Asana.

Linked service properties

The following properties are supported for the Asana linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **Asana**.	Yes
apiToken	Specify an API token for the Asana. Mark this field as **SecureString** to store it securely. Or, you can reference a secret stored in Azure Key Vault.	Yes

Example:

```
{
    "name": "AsanaLinkedService",
    "properties": {
        "type": "Asana",
        "typeProperties": {
            "apiToken": {
                "type": "SecureString",
                "value": "<API token>"
            }
        }
    }
}
```

Mapping data flow properties

When transforming data in mapping data flow, you can read tables from Asana. For more information, see the source transformation in mapping data flows. You can only use an inline dataset as source type.

Source transformation

The below table lists the properties supported by Asana source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Workspace	The ID of the workspace in Asana.	Yes	String	workspaceId
Entity	The ID of the entity in Asana.	Yes	String	entityId

Entity Type	The type of the entity in Asana.	Yes	`teams`
			`portfolios`
			`projects`
				entityType

Asana source script examples

When you use Asana as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
    validateSchema: false,
    store: 'asana',
    format: 'rest',
    workspaceId: '9876543210',
    entityId: '1234567890',
    entityType: 'teams') ~> AsanaSource
```

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Avro format in Azure Data Factory and Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Follow this article when you want to **parse Avro files or write the data into Avro format**.

Avro format is supported for the following connectors: Amazon S3, Amazon S3 Compatible Storage, Azure Blob, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure Files, File System, FTP, Google Cloud Storage, HDFS, HTTP, Oracle Cloud Storage and SFTP.

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the Avro dataset.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **Avro**.	Yes
location	Location settings of the file(s). Each file-based connector has its own location type and supported properties under `location`. **See details in connector article -> Dataset properties section**.	Yes
avroCompressionCodec	The compression codec to use when writing to Avro files. When reading from Avro files, the service automatically determines the compression codec based on the file metadata. Supported types are "**none**" (default), "**deflate**", "**snappy**". Note currently Copy activity doesn't support Snappy when read/write Avro files.	No

NOTE

White space in column name is not supported for Avro files.

Below is an example of Avro dataset on Azure Blob Storage:

```
{
    "name": "AvroDataset",
    "properties": {
        "type": "Avro",
        "linkedServiceName": {
            "referenceName": "<Azure Blob Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "container": "containername",
                "folderPath": "folder/subfolder",
            },
            "avroCompressionCodec": "snappy"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the Avro source and sink.

Avro as source

The following properties are supported in the copy activity ***source*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **AvroSource**.	Yes
storeSettings	A group of properties on how to read data from a data store. Each file-based connector has its own supported read settings under `storeSettings`. **See details in connector article -> Copy activity properties section**.	No

Avro as sink

The following properties are supported in the copy activity ***sink*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **AvroSink**.	Yes
formatSettings	A group of properties. Refer to **Avro write settings** table below.	No
storeSettings	A group of properties on how to write data to a data store. Each file-based connector has its own supported write settings under `storeSettings`. **See details in connector article -> Copy activity properties section**.	No

Supported **Avro write settings** under `formatSettings`:

PROPERTY	DESCRIPTION	REQUIRED
type	The type of formatSettings must be set to **AvroWriteSettings**.	Yes
maxRowsPerFile	When writing data into a folder, you can choose to write to multiple files and specify the max rows per file.	No
fileNamePrefix	Applicable when `maxRowsPerFile` is configured. Specify the file name prefix when writing data to multiple files, resulted in this pattern: `<fileNamePrefix>_00000.<fileExtension>`. If not specified, file name prefix will be auto generated. This property does not apply when source is file-based store or partition-option-enabled data store.	No

Mapping data flow properties

In mapping data flows, you can read and write to avro format in the following data stores: Azure Blob Storage, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2 and SFTP, and you can read avro format in Amazon S3.

Source properties

The below table lists the properties supported by an avro source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Wild card paths	All files matching the wildcard path will be processed. Overrides the folder and file path set in the dataset.	no	String[]	wildcardPaths
Partition root path	For file data that is partitioned, you can enter a partition root path in order to read partitioned folders as columns	no	String	partitionRootPath
List of files	Whether your source is pointing to a text file that lists files to process	no	`true` or `false`	fileList

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Column to store file name	Create a new column with the source file name and path	no	String	rowUrlColumn
After completion	Delete or move the files after processing. File path starts from the container root	no	Delete: `true` or `false` Move: `['<from>', '<to>']`	purgeFiles moveFiles
Filter by last modified	Choose to filter files based upon when they were last altered	no	Timestamp	modifiedAfter modifiedBefore
Allow no files found	If true, an error is not thrown if no files are found	no	`true` or `false`	ignoreNoFilesFound

Sink properties

The below table lists the properties supported by an avro sink. You can edit these properties in the **Settings** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Clear the folder	If the destination folder is cleared prior to write	no	`true` or `false`	truncate
File name option	The naming format of the data written. By default, one file per partition in format `part-#####-tid-<guid>`	no	Pattern: String Per partition: String[] As data in column: String Output to single file: `['<fileName>']`	filePattern partitionFileNames rowUrlColumn partitionFileNames
Quote all	Enclose all values in quotes	no	`true` or `false`	quoteAll

Data type support

Copy activity

Avro complex data types are not supported (records, enums, arrays, maps, unions, and fixed) in Copy Activity.

Data flows

When working with Avro files in data flows, you can read and write complex data types, but be sure to clear the physical schema from the dataset first. In data flows, you can set your logical projection and derive columns that are complex structures, then auto-map those fields to an Avro file.

Next steps

- Copy activity overview
- Lookup activity

- GetMetadata activity

Copy and transform data in Azure Blob storage by using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 36 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy activity in Azure Data Factory and Azure Synapse pipelines to copy data from and to Azure Blob storage. It also describes how to use the Data Flow activity to transform data in Azure Blob storage. To learn more read the Azure Data Factory and the Azure Synapse Analytics introduction articles.

> **TIP**
> To learn about a migration scenario for a data lake or a data warehouse, see the article Migrate data from your data lake or data warehouse to Azure.

Supported capabilities

This Azure Blob storage connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/sink)	① ②	✓ Exclude storage account V1
Mapping data flow (source/sink)	①	✓ Exclude storage account V1
Lookup activity	① ②	✓ Exclude storage account V1
GetMetadata activity	① ②	✓ Exclude storage account V1
Delete activity	① ②	✓ Exclude storage account V1

① Azure integration runtime ② Self-hosted integration runtime

For the Copy activity, this Blob storage connector supports:

- Copying blobs to and from general-purpose Azure storage accounts and hot/cool blob storage.
- Copying blobs by using an account key, a service shared access signature (SAS), a service principal, or managed identities for Azure resource authentications.
- Copying blobs from block, append, or page blobs and copying data to only block blobs.
- Copying blobs as is, or parsing or generating blobs with supported file formats and compression codecs.
- Preserving file metadata during copy.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

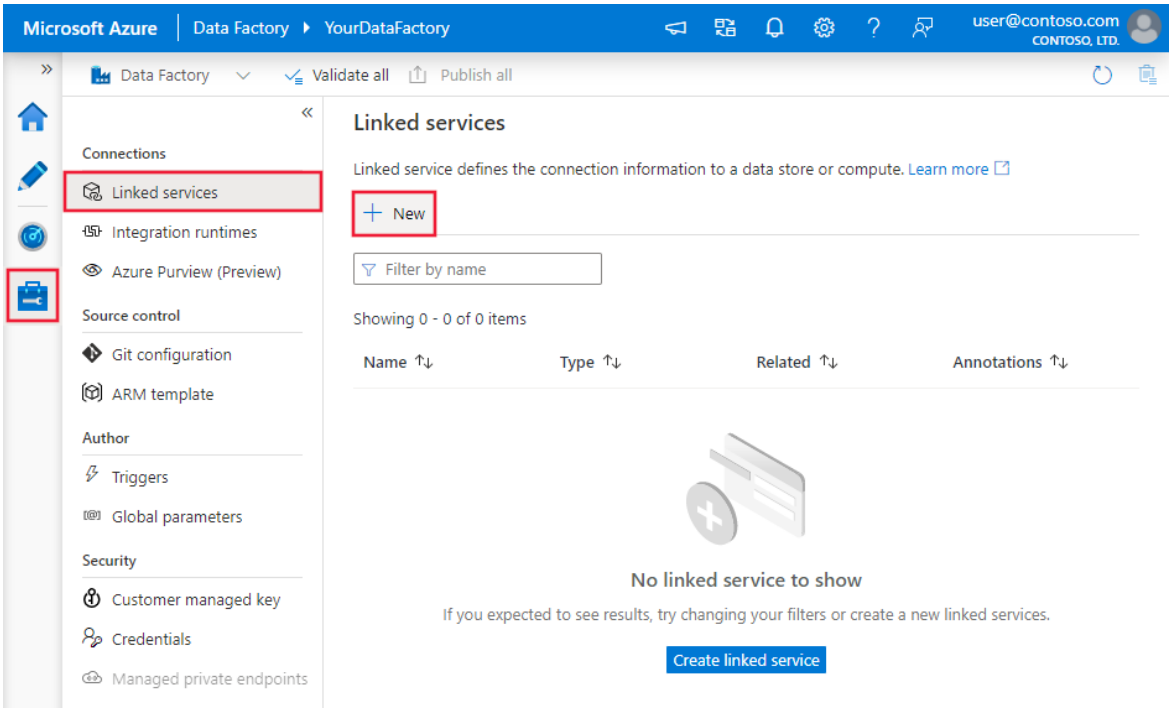
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create an Azure Blob Storage linked service using UI

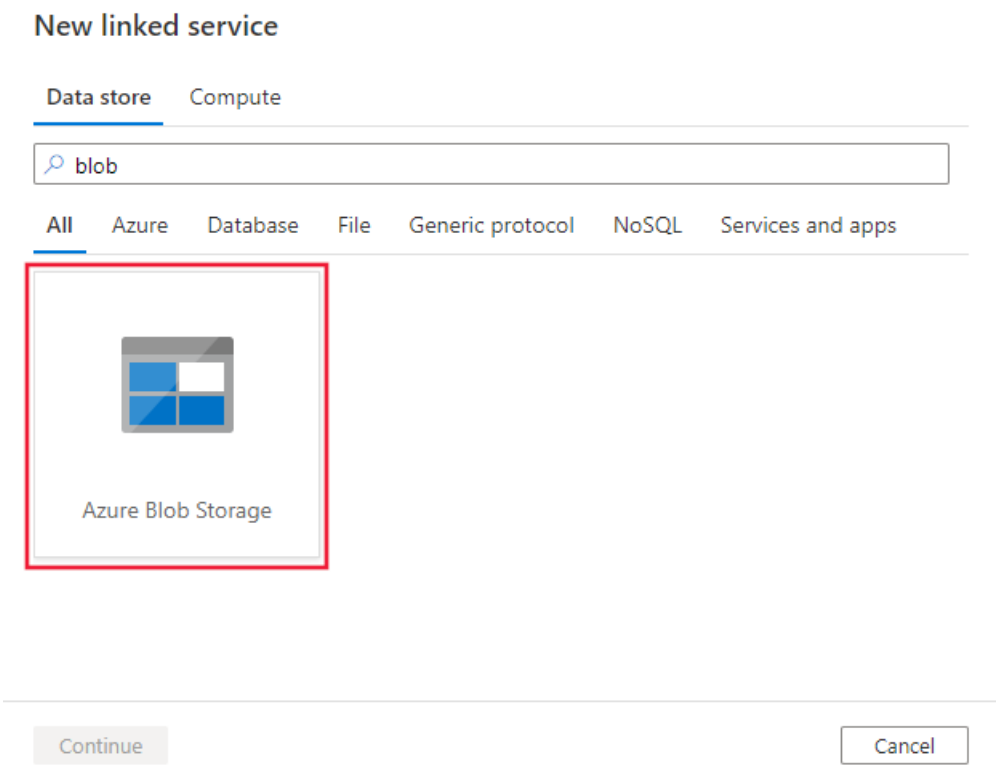
Use the following steps to create an Azure Blob Storage linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

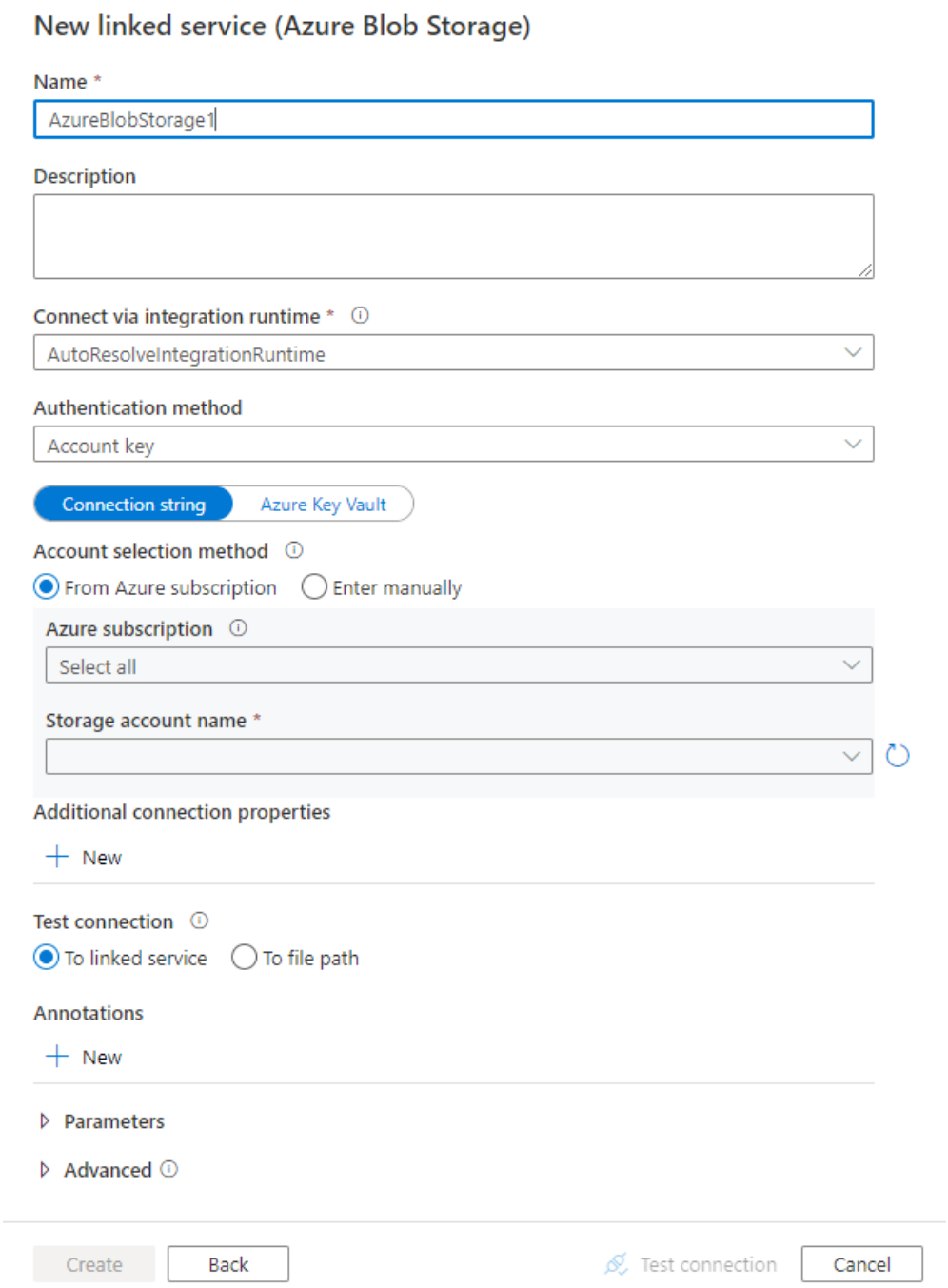
   - Azure Data Factory
   - Azure Synapse



2. Search for blob and select the Azure Blob Storage connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory and Synapse pipeline entities specific to Blob storage.

Linked service properties

This Blob storage connector supports the following authentication types. See the corresponding sections for details.

- Account key authentication
- Shared access signature authentication
- Service principal authentication
- System-assigned managed identity authentication
- User-assigned managed identity authentication

NOTE

- If want to use the public Azure integration runtime to connect to your Blob storage by leveraging the **Allow trusted Microsoft services to access this storage account** option enabled on Azure Storage firewall, you must use managed identity authentication. For more information about the Azure Storage firewalls settings, see Configure Azure Storage firewalls and virtual networks.
- When you use PolyBase or COPY statement to load data into Azure Synapse Analytics, if your source or staging Blob storage is configured with an Azure Virtual Network endpoint, you must use managed identity authentication as required by Azure Synapse. See the Managed identity authentication section for more configuration prerequisites.

NOTE

Azure HDInsight and Azure Machine Learning activities only support authentication that uses Azure Blob storage account keys.

Account key authentication

The following properties are supported for storage account key authentication in Azure Data Factory or Synapse pipelines:

PROPERTY	DESCRIPTION	REQUIRED
type	The `type` property must be set to `AzureBlobStorage` (suggested) or `AzureStorage` (see the following notes).	Yes
connectionString	Specify the information needed to connect to Storage for the `connectionString` property. You can also put the account key in Azure Key Vault and pull the `accountKey` configuration out of the connection string. For more information, see the following samples and the Store credentials in Azure Key Vault article.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

NOTE

A secondary Blob service endpoint is not supported when you're using account key authentication. You can use other authentication types.

> **NOTE**
>
> If you're using the `AzureStorage` type linked service, it's still supported as is. But we suggest that you use the new `AzureBlobStorage` linked service type going forward.

Example:

```
{
    "name": "AzureBlobStorageLinkedService",
    "properties": {
        "type": "AzureBlobStorage",
        "typeProperties": {
          "connectionString": "DefaultEndpointsProtocol=https;AccountName=<accountname>;AccountKey=
<accountkey>"
        },
        "connectVia": {
          "referenceName": "<name of Integration Runtime>",
          "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store the account key in Azure Key Vault

```
{
    "name": "AzureBlobStorageLinkedService",
    "properties": {
        "type": "AzureBlobStorage",
        "typeProperties": {
            "connectionString": "DefaultEndpointsProtocol=https;AccountName=<accountname>;",
            "accountKey": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Shared access signature authentication

A shared access signature provides delegated access to resources in your storage account. You can use a shared access signature to grant a client limited permissions to objects in your storage account for a specified time.

You don't have to share your account access keys. The shared access signature is a URI that encompasses in its query parameters all the information necessary for authenticated access to a storage resource. To access storage resources with the shared access signature, the client only needs to pass in the shared access signature to the appropriate constructor or method.

For more information about shared access signatures, see Shared access signatures: Understand the shared access signature model.

NOTE

- The service now supports both *service shared access signatures* and *account shared access signatures*. For more information about shared access signatures, see Grant limited access to Azure Storage resources using shared access signatures.
- In later dataset configurations, the folder path is the absolute path starting from the container level. You need to configure one aligned with the path in your SAS URI.

The following properties are supported for using shared access signature authentication:

PROPERTY	DESCRIPTION	REQUIRED
type	The `type` property must be set to `AzureBlobStorage` (suggested) or `AzureStorage` (see the following note).	Yes
sasUri	Specify the shared access signature URI to the Storage resources such as blob or container. Mark this field as `SecureString` to store it securely. You can also put the SAS token in Azure Key Vault to use auto-rotation and remove the token portion. For more information, see the following samples and Store credentials in Azure Key Vault.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

NOTE

If you're using the `AzureStorage` type linked service, it's still supported as is. But we suggest that you use the new `AzureBlobStorage` linked service type going forward.

Example:

```
{
    "name": "AzureBlobStorageLinkedService",
    "properties": {
        "type": "AzureBlobStorage",
        "typeProperties": {
            "sasUri": {
                "type": "SecureString",
                "value": "<SAS URI of the Azure Storage resource e.g.
https://<accountname>.blob.core.windows.net/?sv=<storage version>&st=<start time>&se=<expire time>&sr=
<resource>&sp=<permissions>&sip=<ip range>&spr=<protocol>&sig=<signature>>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store the account key in Azure Key Vault

```
{
    "name": "AzureBlobStorageLinkedService",
    "properties": {
        "type": "AzureBlobStorage",
        "typeProperties": {
            "sasUri": {
                "type": "SecureString",
                "value": "<SAS URI of the Azure Storage resource without token e.g.
https://<accountname>.blob.core.windows.net/>"
            },
            "sasToken": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName with value of SAS token e.g. ?sv=<storage version>&st=<start
time>&se=<expire time>&sr=<resource>&sp=<permissions>&sip=<ip range>&spr=<protocol>&sig=<signature>>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

When you create a shared access signature URI, consider the following points:

- Set appropriate read/write permissions on objects based on how the linked service (read, write, read/write) is used.
- Set **Expiry time** appropriately. Make sure that the access to Storage objects doesn't expire within the active period of the pipeline.
- The URI should be created at the right container or blob based on the need. A shared access signature URI to a blob allows the data factory or Synapse pipeline to access that particular blob. A shared access signature URI to a Blob storage container allows the data factory or Synapse pipeline to iterate through blobs in that container. To provide access to more or fewer objects later, or to update the shared access signature URI, remember to update the linked service with the new URI.

Service principal authentication

For general information about Azure Storage service principal authentication, see Authenticate access to Azure Storage using Azure Active Directory.

To use service principal authentication, follow these steps:

1. Register an application entity in Azure Active Directory (Azure AD) by following Register your application with an Azure AD tenant. Make note of these values, which you use to define the linked service:
   - Application ID
   - Application key
   - Tenant ID
2. Grant the service principal proper permission in Azure Blob storage. For more information on the roles, see Use the Azure portal to assign an Azure role for access to blob and queue data.
   - **As source**, in **Access control (IAM)**, grant at least the **Storage Blob Data Reader** role.
   - **As sink**, in **Access control (IAM)**, grant at least the **Storage Blob Data Contributor** role.

These properties are supported for an Azure Blob storage linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **AzureBlobStorage**.	Yes
serviceEndpoint	Specify the Azure Blob storage service endpoint with the pattern of `https://<accountName>.blob.core.windows.net/`.	Yes
accountKind	Specify the kind of your storage account. Allowed values are: **Storage** (general purpose v1), **StorageV2** (general purpose v2), **BlobStorage**, or **BlockBlobStorage**. When using Azure Blob linked service in data flow, managed identity or service principal authentication is not supported when account kind as empty or "Storage". Specify the proper account kind, choose a different authentication, or upgrade your storage account to general purpose v2.	No
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key. Mark this field as **SecureString** to store it securelyFactory, or reference a secret stored in Azure Key Vault.	Yes
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. Retrieve it by hovering over the upper-right corner of the Azure portal.	Yes

PROPERTY	DESCRIPTION	REQUIRED
azureCloudType	For service principal authentication, specify the type of Azure cloud environment, to which your Azure Active Directory application is registered. Allowed values are **AzurePublic**, **AzureChina**, **AzureUsGovernment**, and **AzureGermany**. By default, the data factory or Synapse pipeline's cloud environment is used.	No
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

NOTE

- If your blob account enables soft delete, service principal authentication is not supported in Data Flow.
- If you access the blob storage through private endpoint using Data Flow, note when service principal authentication is used Data Flow connects to the ADLS Gen2 endpoint instead of Blob endpoint. Make sure you create the corresponding private endpoint in your data factory or Synapse workspace to enable access.

NOTE

Service principal authentication is supported only by the "AzureBlobStorage" type linked service, not the previous "AzureStorage" type linked service.

Example:

```
{
    "name": "AzureBlobStorageLinkedService",
    "properties": {
        "type": "AzureBlobStorage",
        "typeProperties": {
            "serviceEndpoint": "https://<accountName>.blob.core.windows.net/",
            "accountKind": "StorageV2",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

System-assigned managed identity authentication

A data factory or Synapse pipeline can be associated with a system-assigned managed identity for Azure resources, which represents that resource for authentication to other Azure services. You can directly use this system-assigned managed identity for Blob storage authentication, which is similar to using your own service principal. It allows this designated resource to access and copy data from or to Blob storage. To learn more about managed identities for Azure resources, see Managed identities for Azure resources

For general information about Azure Storage authentication, see Authenticate access to Azure Storage using Azure Active Directory. To use managed identities for Azure resource authentication, follow these steps:

1. Retrieve system-assigned managed identity information by copying the value of the system-assigned managed identity object ID generated along with your factory or Synapse workspace.
2. Grant the managed identity permission in Azure Blob storage. For more information on the roles, see Use the Azure portal to assign an Azure role for access to blob and queue data.
   - **As source**, in **Access control (IAM)**, grant at least the **Storage Blob Data Reader** role.
   - **As sink**, in **Access control (IAM)**, grant at least the **Storage Blob Data Contributor** role.

These properties are supported for an Azure Blob storage linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **AzureBlobStorage**.	Yes
serviceEndpoint	Specify the Azure Blob storage service endpoint with the pattern of `https://<accountName>.blob.core.windows.net/`.	Yes
accountKind	Specify the kind of your storage account. Allowed values are: **Storage** (general purpose v1), **StorageV2** (general purpose v2), **BlobStorage**, or **BlockBlobStorage**. When using Azure Blob linked service in data flow, managed identity or service principal authentication is not supported when account kind as empty or "Storage". Specify the proper account kind, choose a different authentication, or upgrade your storage account to general purpose v2.	No
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

Example:

```
{
    "name": "AzureBlobStorageLinkedService",
    "properties": {
        "type": "AzureBlobStorage",
        "typeProperties": {
            "serviceEndpoint": "https://<accountName>.blob.core.windows.net/",
            "accountKind": "StorageV2"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

User-assigned managed identity authentication

A data factory can be assigned with one or multiple user-assigned managed identities. You can use this user-assigned managed identity for Blob storage authentication, which allows to access and copy data from or to Blob storage. To learn more about managed identities for Azure resources, see Managed identities for Azure resources

For general information about Azure storage authentication, see Authenticate access to Azure Storage using Azure Active Directory. To use user-assigned managed identity authentication, follow these steps:

1. Create one or multiple user-assigned managed identities and grant permission in Azure Blob storage. For more information on the roles, see Use the Azure portal to assign an Azure role for access to blob and queue data.
    - **As source**, in **Access control (IAM)**, grant at least the **Storage Blob Data Reader** role.
    - **As sink**, in **Access control (IAM)**, grant at least the **Storage Blob Data Contributor** role.
2. Assign one or multiple user-assigned managed identities to your data factory and create credentials for each user-assigned managed identity.

These properties are supported for an Azure Blob storage linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **AzureBlobStorage**.	Yes
serviceEndpoint	Specify the Azure Blob storage service endpoint with the pattern of `https://<accountName>.blob.core.windows.net/`.	Yes

PROPERTY	DESCRIPTION	REQUIRED
accountKind	Specify the kind of your storage account. Allowed values are: **Storage** (general purpose v1), **StorageV2** (general purpose v2), **BlobStorage**, or **BlockBlobStorage**. When using Azure Blob linked service in data flow, managed identity or service principal authentication is not supported when account kind as empty or "Storage". Specify the proper account kind, choose a different authentication, or upgrade your storage account to general purpose v2.	No
credentials	Specify the user-assigned managed identity as the credential object.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

Example:

```
{
    "name": "AzureBlobStorageLinkedService",
    "properties": {
        "type": "AzureBlobStorage",
        "typeProperties": {
            "serviceEndpoint": "https://<accountName>.blob.core.windows.net/",
            "accountKind": "StorageV2",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

IMPORTANT

If you use PolyBase or COPY statement to load data from Blob storage (as a source or as staging) into Azure Synapse Analytics, when you use managed identity authentication for Blob storage, make sure you also follow steps 1 to 3 in this guidance. Those steps will register your server with Azure AD and assign the Storage Blob Data Contributor role to your server. Data Factory handles the rest. If you configure Blob storage with an Azure Virtual Network endpoint, you also need to have **Allow trusted Microsoft services to access this storage account** turned on under Azure Storage account **Firewalls and Virtual networks** settings menu as required by Azure Synapse.

NOTE

- If your blob account enables soft delete, system-assigned/user-assigned managed identity authentication is not supported in Data Flow.
- If you access the blob storage through private endpoint using Data Flow, note when system-assigned/user-assigned managed identity authentication is used Data Flow connects to the ADLS Gen2 endpoint instead of Blob endpoint. Make sure you create the corresponding private endpoint in ADF to enable access.

NOTE

System-assigned/user-assigned managed identity authentication is supported only by the "AzureBlobStorage" type linked service, not the previous "AzureStorage" type linked service.

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Azure Blob storage under `location` settings in a format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the location in the dataset must be set to **AzureBlobStorageLocation**.	Yes
container	The blob container.	Yes
folderPath	The path to the folder under the given container. If you want to use a wildcard to filter the folder, skip this setting and specify that in activity source settings.	No
fileName	The file name under the given container and folder path. If you want to use wildcard to filter files, skip this setting and specify that in activity source settings.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<Azure Blob Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "container": "containername",
                "folderPath": "folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties that the Blob storage source and sink support.

Blob storage as a source type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Azure Blob storage under `storeSettings` settings in a format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property under `storeSettings` must be set to **AzureBlobStorageReadSettings**.	Yes
Locate the files to copy:		
OPTION 1: static path	Copy from the given container or folder/file path specified in the dataset. If you want to copy all blobs from a container or folder, additionally specify `wildcardFileName` as `*`.	

PROPERTY	DESCRIPTION	REQUIRED
OPTION 2: blob prefix - prefix	Prefix for the blob name under the given container configured in a dataset to filter source blobs. Blobs whose names start with `container_in_dataset/this_prefix` are selected. It utilizes the service-side filter for Blob storage, which provides better performance than a wildcard filter. When you use prefix and choose to copy to file-based sink with preserving hierarchy, note the sub-path after the last "/" in prefix will be preserved. For example, you have source `container/folder/subfolder/file.txt`, and configure prefix as `folder/sub`, then the preserved file path is `subfolder/file.txt`.	No
OPTION 3: wildcard - wildcardFolderPath	The folder path with wildcard characters under the given container configured in a dataset to filter source folders. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your folder name has wildcard or this escape character inside. See more examples in Folder and file filter examples.	No
OPTION 3: wildcard - wildcardFileName	The file name with wildcard characters under the given container and folder path (or wildcard folder path) to filter source files. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your file name has a wildcard or this escape character inside. See more examples in Folder and file filter examples.	Yes
OPTION 4: a list of files - fileListPath	Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset. When you're using this option, do not specify a file name in the dataset. See more examples in File list examples.	No
Additional settings:		

PROPERTY	DESCRIPTION	REQUIRED
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when **recursive** is set to **true** and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**. This property doesn't apply when you configure `fileListPath`.	No
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No
modifiedDatetimeStart	Files are filtered based on the attribute: last modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to a UTC time zone in the format of "2018-12-01T05:00:00Z". The properties can be **NULL**, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is **NULL**, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is **NULL**, the files whose last modified attribute is less than the datetime value will be selected. This property doesn't apply when you configure `fileListPath`.	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. - When you use prefix, partition root path is sub-path before the last "/". For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

NOTE

For Parquet/delimited text format, the **BlobSource** type for the Copy activity source mentioned in the next section is still supported as is for backward compatibility. We suggest that you use the new model until the authoring UI has switched to generating these new types.

Example:

```
"activities":[
    {
        "name": "CopyFromBlob",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "AzureBlobStorageReadSettings",
                    "recursive": true,
                    "wildcardFolderPath": "myfolder*A",
                    "wildcardFileName": "*.csv"
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

NOTE

The `$logs` container, which is automatically created when Storage Analytics is enabled for a storage account, isn't shown when a container listing operation is performed via the UI. The file path must be provided directly for your data factory or Synapse pipeline to consume files from the `$logs` container.

Blob storage as a sink type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- JSON format
- ORC format
- Parquet format

The following properties are supported for Azure Blob storage under `storeSettings` settings in a format-based copy sink:

PROPERTY	DESCRIPTION	REQUIRED

PROPERTY	DESCRIPTION	REQUIRED
type	The `type` property under `storeSettings` must be set to `AzureBlobStorageWriteSettings`.	Yes
copyBehavior	Defines the copy behavior when the source is files from a file-based data store. Allowed values are: - **PreserveHierarchy (default)**: Preserves the file hierarchy in the target folder. The relative path of the source file to the source folder is identical to the relative path of the target file to the target folder. - **FlattenHierarchy**: All files from the source folder are in the first level of the target folder. The target files have autogenerated names. - **MergeFiles**: Merges all files from the source folder to one file. If the file or blob name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.	No
blockSizeInMB	Specify the block size, in megabytes, used to write data to block blobs. Learn more about Block Blobs. Allowed value is *between 4 MB and 100 MB*. By default, the service automatically determines the block size based on your source store type and data. For nonbinary copy into Blob storage, the default block size is 100 MB so it can fit in (at most) 4.95 TB of data. It might be not optimal when your data is not large, especially when you use the self-hosted integration runtime with poor network connections that result in operation timeout or performance issues. You can explicitly specify a block size, while ensuring that `blockSizeInMB*50000` is big enough to store the data. Otherwise, the Copy activity run will fail.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

PROPERTY	DESCRIPTION	REQUIRED
metadata	Set custom metadata when copy to sink. Each object under the `metadata` array represents an extra column. The `name` defines the metadata key name, and the `value` indicates the data value of that key. If preserve attributes feature is used, the specified metadata will union/overwrite with the source file metadata. Allowed data values are: - `$$LASTMODIFIED`: a reserved variable indicates to store the source files' last modified time. Apply to file-based source with binary format only. - **Expression** - **Static value**	No

Example:

```
"activities":[
    {
        "name": "CopyFromBlob",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Parquet output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "ParquetSink",
                "storeSettings":{
                    "type": "AzureBlobStorageWriteSettings",
                    "copyBehavior": "PreserveHierarchy",
                    "metadata": [
                        {
                            "name": "testKey1",
                            "value": "value1"
                        },
                        {
                            "name": "testKey2",
                            "value": "value2"
                        },
                        {
                            "name": "lastModifiedKey",
                            "value": "$$LASTMODIFIED"
                        }
                    ]
                }
            }
        }
    }
]
```

Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`container/Folder*`	(empty, use default)	false	container FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`container/Folder*`	(empty, use default)	true	container FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv
`container/Folder*`	`*.csv`	false	container FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`container/Folder*`	`*.csv`	true	container FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This section describes the resulting behavior of using a file list path in the Copy activity source.

Assume that you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	CONFIGURATION
container FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In dataset:** - Container: `container` - Folder path: `FolderA` **In Copy activity source:** - File list path: `container/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line, with the relative path to the path configured in the dataset.

Some recursive and copyBehavior examples

This section describes the resulting behavior of the Copy operation for different combinations of **recursive** and

copyBehavior values.

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET
true	preserveHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder, Folder1, is created with the same structure as the source: Folder1 File1 File2 Subfolder1 File3 File4 File5
true	flattenHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder, Folder1, is created with the following structure: Folder1 autogenerated name for File1 autogenerated name for File2 autogenerated name for File3 autogenerated name for File4 autogenerated name for File5
true	mergeFiles	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder, Folder1, is created with the following structure: Folder1 File1 + File2 + File3 + File4 + File5 contents are merged into one file with an autogenerated file name.
false	preserveHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder, Folder1, is created with the following structure: Folder1 File1 File2 Subfolder1 with File3, File4, and File5 is not picked up.

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET

false	flattenHierarchy	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target folder, Folder1, is created with the following structure:
			
			Folder1
			  autogenerated name for File1
			  autogenerated name for File2
			
			Subfolder1 with File3, File4, and File5 is not picked up.

false	mergeFiles	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target folder, Folder1, is created with the following structure:
			
			Folder1
			  File1 + File2 contents are merged into one file with an autogenerated file name. autogenerated name for File1
			
			Subfolder1 with File3, File4, and File5 is not picked up.

Preserving metadata during copy

When you copy files from Amazon S3, Azure Blob storage, or Azure Data Lake Storage Gen2 to Azure Data Lake Storage Gen2 or Azure Blob storage, you can choose to preserve the file metadata along with data. Learn more from Preserve metadata.

Mapping data flow properties

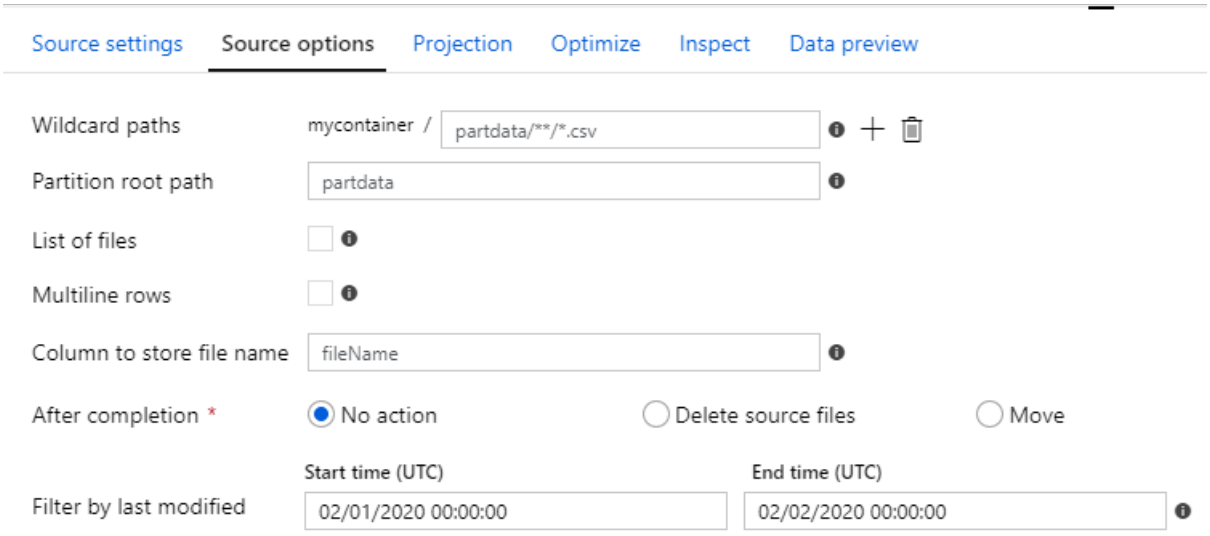
When you're transforming data in mapping data flows, you can read and write files from Azure Blob storage in the following formats:

- Avro
- Delimited text
- Delta
- Excel
- JSON
- Parquet

Format specific settings are located in the documentation for that format. For more information, see Source transformation in mapping data flow and Sink transformation in mapping data flow.

Source transformation

In source transformation, you can read from a container, folder, or individual file in Azure Blob storage. Use the **Source options** tab to manage how the files are read.



Wildcard paths: Using a wildcard pattern will instruct the service to loop through each matching folder and file in a single source transformation. This is an effective way to process multiple files within a single flow. Add multiple wildcard matching patterns with the plus sign that appears when you hover over your existing wildcard pattern.

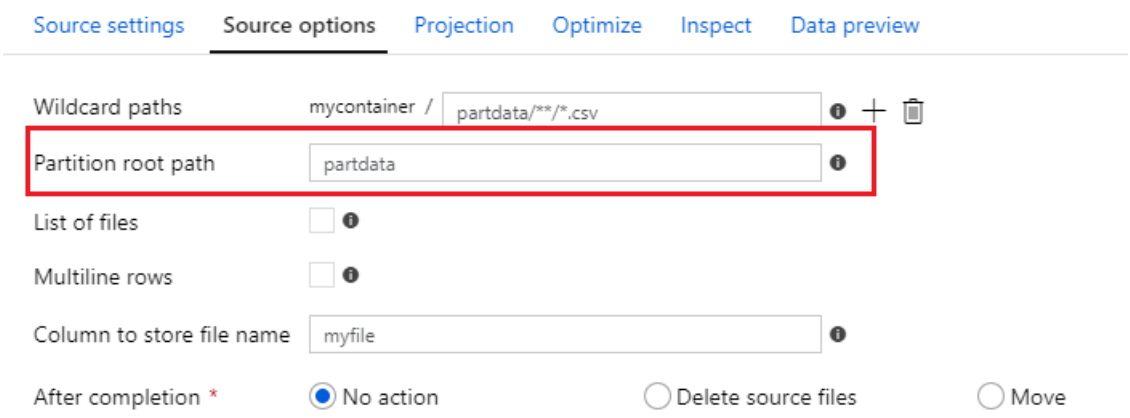
From your source container, choose a series of files that match a pattern. Only a container can be specified in the dataset. Your wildcard path must therefore also include your folder path from the root folder.

Wildcard examples:

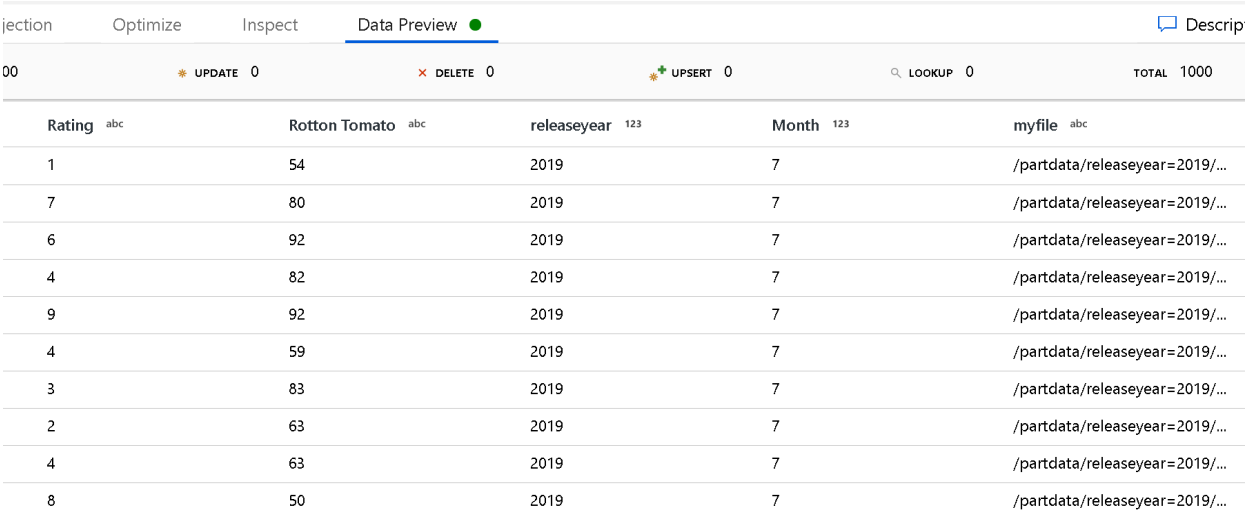
- `*` Represents any set of characters.
- `**` Represents recursive directory nesting.
- `?` Replaces one character.
- `[]` Matches one or more characters in the brackets.
- `/data/sales/**/*.csv` Gets all .csv files under /data/sales.
- `/data/sales/20??/**/` Gets all files in the 20th century.
- `/data/sales/*/*/*.csv` Gets .csv files two levels under /data/sales.
- `/data/sales/2004/*/12/[XY]1?.csv` Gets all .csv files in December 2004 starting with X or Y prefixed by a two-digit number.

Partition root path: If you have partitioned folders in your file source with a `key=value` format (for example, `year=2019`), then you can assign the top level of that partition folder tree to a column name in your data flow's data stream.

First, set a wildcard to include all paths that are the partitioned folders plus the leaf files that you want to read.



Use the **Partition root path** setting to define what the top level of the folder structure is. When you view the contents of your data via a data preview, you'll see that the service will add the resolved partitions found in each of your folder levels.



Rating abc	Rotton Tomato abc	releaseyear 123	Month 123	myfile abc
1	54	2019	7	/partdata/releaseyear=2019/...
7	80	2019	7	/partdata/releaseyear=2019/...
6	92	2019	7	/partdata/releaseyear=2019/...
4	82	2019	7	/partdata/releaseyear=2019/...
9	92	2019	7	/partdata/releaseyear=2019/...
4	59	2019	7	/partdata/releaseyear=2019/...
3	83	2019	7	/partdata/releaseyear=2019/...
2	63	2019	7	/partdata/releaseyear=2019/...
4	63	2019	7	/partdata/releaseyear=2019/...
8	50	2019	7	/partdata/releaseyear=2019/...

List of files: This is a file set. Create a text file that includes a list of relative path files to process. Point to this text file.

Column to store file name: Store the name of the source file in a column in your data. Enter a new column name here to store the file name string.

After completion: Choose to do nothing with the source file after the data flow runs, delete the source file, or move the source file. The paths for the move are relative.

To move source files to another location post-processing, first select "Move" for file operation. Then, set the "from" directory. If you're not using any wildcards for your path, then the "from" setting will be the same folder as your source folder.

If you have a source path with wildcard, your syntax will look like this:

`/data/sales/20??/**/*.csv`

You can specify "from" as:

`/data/sales`

And you can specify "to" as:

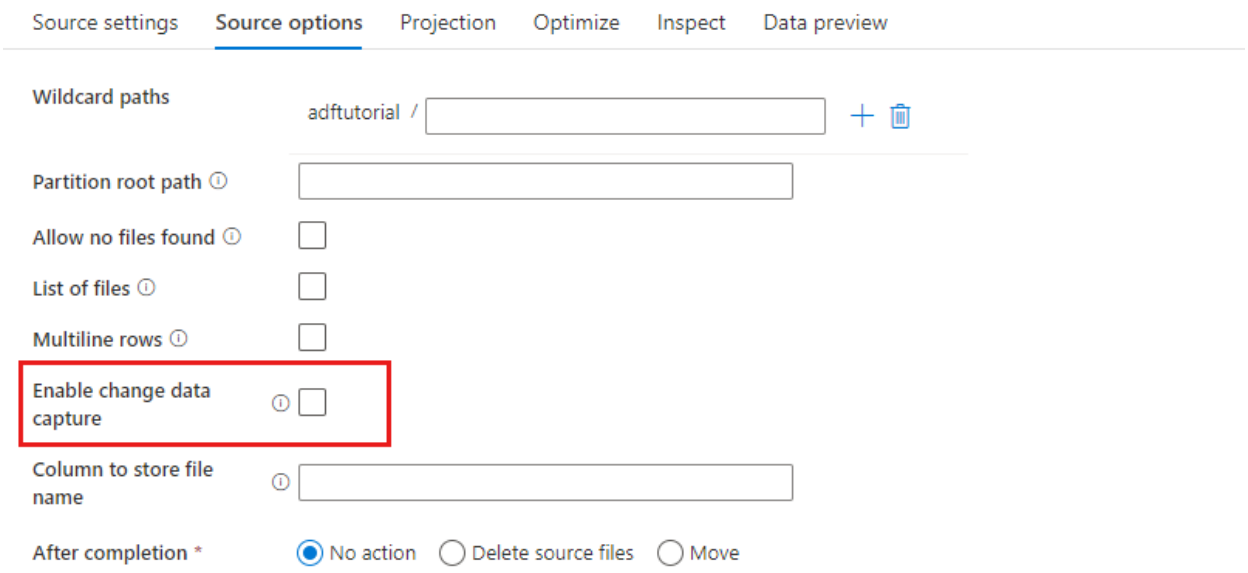
`/backup/priorSales`

In this case, all files that were sourced under `/data/sales` are moved to `/backup/priorSales`.

> **NOTE**
>
> File operations run only when you start the data flow from a pipeline run (a pipeline debug or execution run) that uses the Execute Data Flow activity in a pipeline. File operations *do not* run in Data Flow debug mode.

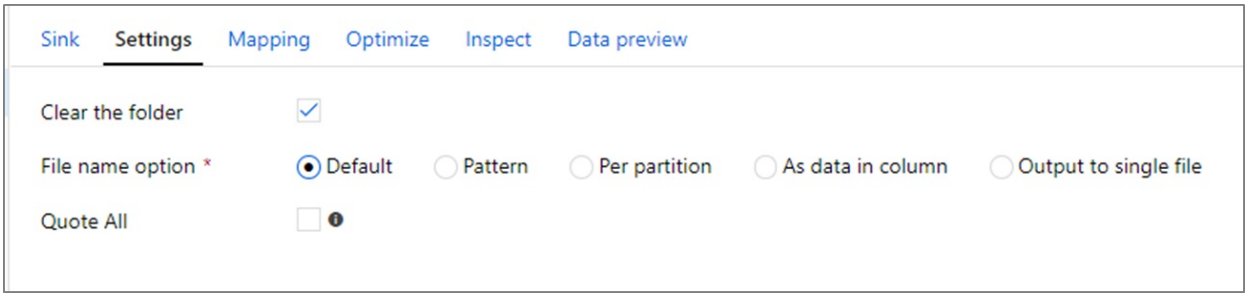
Filter by last modified: You can filter which files you process by specifying a date range of when they were last modified. All datetimes are in UTC.

Enable change data capture: If true, you will get new or changed files only from the last run. Initial load of full snapshot data will always be gotten in the first run, followed by capturing new or changed files only in next runs. For more details, see Change data capture.



Sink properties

In the sink transformation, you can write to either a container or a folder in Azure Blob storage. Use the **Settings** tab to manage how the files get written.



Clear the folder: Determines whether or not the destination folder gets cleared before the data is written.

File name option: Determines how the destination files are named in the destination folder. The file name options are:

- **Default**: Allow Spark to name files based on PART defaults.
- **Pattern**: Enter a pattern that enumerates your output files per partition. For example, `loans[n].csv` will create `loans1.csv`, `loans2.csv`, and so on.
- **Per partition**: Enter one file name per partition.
- **As data in column**: Set the output file to the value of a column. The path is relative to the dataset container, not the destination folder. If you have a folder path in your dataset, it will be overridden.
- **Output to a single file**: Combine the partitioned output files into a single named file. The path is relative to the dataset folder. Be aware that the merge operation can possibly fail based on node size. We don't

recommend this option for large datasets.

Quote all: Determines whether to enclose all values in quotation marks.

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity.

Delete activity properties

To learn details about the properties, check Delete activity.

Legacy models

> **NOTE**
> The following models are still supported as is for backward compatibility. We suggest that you use the new model mentioned earlier. The authoring UI has switched to generating the new model.

Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED
type	The `type` property of the dataset must be set to `AzureBlob`.	Yes
folderPath	Path to the container and folder in Blob storage. A wildcard filter is supported for the path, excluding container name. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your folder name has a wildcard or this escape character inside. An example is: `myblobcontainer/myblobfolder/`. See more examples in Folder and file filter examples.	Yes for the Copy or Lookup activity, No for the GetMetadata activity

PROPERTY	DESCRIPTION	REQUIRED
fileName	Name or wildcard filter for the blobs under the specified `folderPath` value. If you don't specify a value for this property, the dataset points to all blobs in the folder. For the filter, allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). - Example 1: `"fileName": "*.csv"` - Example 2: `"fileName": "???20180427.txt"` Use `^` to escape if your file name has a wildcard or this escape character inside. When `fileName` isn't specified for an output dataset and `preserveHierarchy` isn't specified in the activity sink, the Copy activity automatically generates the blob name with the following pattern: "*Data.[activity run ID GUID].[GUID if FlattenHierarchy].[format if configured].[compression if configured]*". For example: "Data.0a405f8a-93ff-4c6f-b3be-f69616f1df7a.txt.gz". If you copy from a tabular source by using a table name instead of a query, the name pattern is `[table name].[format].[compression if configured]`. For example: "MyTable.csv".	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	Files are filtered based on the attribute: last modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z". Be aware that enabling this setting will affect the overall performance of data movement when you want to filter huge amounts of files. The properties can be `NULL`, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is `NULL`, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is `NULL`, the files whose last modified attribute is less than the datetime value will be selected.	No
modifiedDatetimeEnd	Files are filtered based on the attribute: last modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z". Be aware that enabling this setting will affect the overall performance of data movement when you want to filter huge amounts of files. The properties can be `NULL`, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is `NULL`, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is `NULL`, the files whose last modified attribute is less than the datetime value will be selected.	No

PROPERTY	DESCRIPTION	REQUIRED
format	If you want to copy files as is between file-based stores (binary copy), skip the format section in both the input and output dataset definitions. If you want to parse or generate files with a specific format, the following file format types are supported: **TextFormat**, **JsonFormat**, **AvroFormat**, **OrcFormat**, and **ParquetFormat**. Set the **type** property under **format** to one of these values. For more information, see the Text format, JSON format, Avro format, Orc format, and Parquet format sections.	No (only for binary copy scenario)
compression	Specify the type and level of compression for the data. For more information, see Supported file formats and compression codecs. Supported types are **GZip**, **Deflate**, **BZip2**, and **ZipDeflate**. Supported levels are **Optimal** and **Fastest**.	No

TIP

To copy all blobs under a folder, specify **folderPath** only.
To copy a single blob with a given name, specify **folderPath** for the folder part and **fileName** for the file name.
To copy a subset of blobs under a folder, specify **folderPath** for the folder part and **fileName** with a wildcard filter.

Example:

```
{
    "name": "AzureBlobDataset",
    "properties": {
        "type": "AzureBlob",
        "linkedServiceName": {
            "referenceName": "<Azure Blob storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "folderPath": "mycontainer/myfolder",
            "fileName": "*",
            "modifiedDatetimeStart": "2018-12-01T05:00:00Z",
            "modifiedDatetimeEnd": "2018-12-01T06:00:00Z",
            "format": {
                "type": "TextFormat",
                "columnDelimiter": ",",
                "rowDelimiter": "\n"
            },
            "compression": {
                "type": "GZip",
                "level": "Optimal"
            }
        }
    }
}
```

Legacy source model for the Copy activity

PROPERTY	DESCRIPTION	REQUIRED
type	The `type` property of the Copy activity source must be set to `BlobSource`.	Yes
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when `recursive` is set to `true` and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are `true` (default) and `false`.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromBlob",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Azure Blob input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "BlobSource",
                "recursive": true
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Legacy sink model for the Copy activity

PROPERTY	DESCRIPTION	REQUIRED
type	The `type` property of the Copy activity sink must be set to `BlobSink`.	Yes
copyBehavior	Defines the copy behavior when the source is files from a file-based data store. Allowed values are: - **PreserveHierarchy (default)**: Preserves the file hierarchy in the target folder. The relative path of source file to source folder is identical to the relative path of target file to target folder. - **FlattenHierarchy**: All files from the source folder are in the first level of the target folder. The target files have autogenerated names. - **MergeFiles**: Merges all files from the source folder to one file. If the file or blob name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToBlob",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Azure Blob output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "BlobSink",
                "copyBehavior": "PreserveHierarchy"
            }
        }
    }
]
```

Change data capture (preview)

Azure Data Factory can get new or changed files only from Azure Blob Storage by enabling **Enable change data capture (Preview)** in the mapping data flow source transformation. With this connector option, you can read new or updated files only and apply transformations before loading transformed data into destination datasets of your choice.

Make sure you keep the pipeline and activity name unchanged, so that the checkpoint can always be recorded from the last run to get changes from there. If you change your pipeline name or activity name, the checkpoint will be reset, and you will start from the beginning in the next run.

When you debug the pipeline, the **Enable change data capture (Preview)** works as well. Be aware that the checkpoint will be reset when you refresh your browser during the debug run. After you are satisfied with the result from debug run, you can publish and trigger the pipeline. It will always start from the beginning regardless of the previous checkpoint recorded by debug run.

In the monitoring section, you always have the chance to rerun a pipeline. When you are doing so, the changes are always gotten from the checkpoint record in your selected pipeline run.

Next steps

For a list of data stores that the Copy activity supports as sources and sinks, see Supported data stores.

Copy data to an Azure Cognitive Search index using Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data into Azure Cognitive Search index. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Azure Cognitive Search connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (-/sink)	① ②	✓

① Azure integration runtime ② Self-hosted integration runtime

You can copy data from any supported source data store into search index. For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

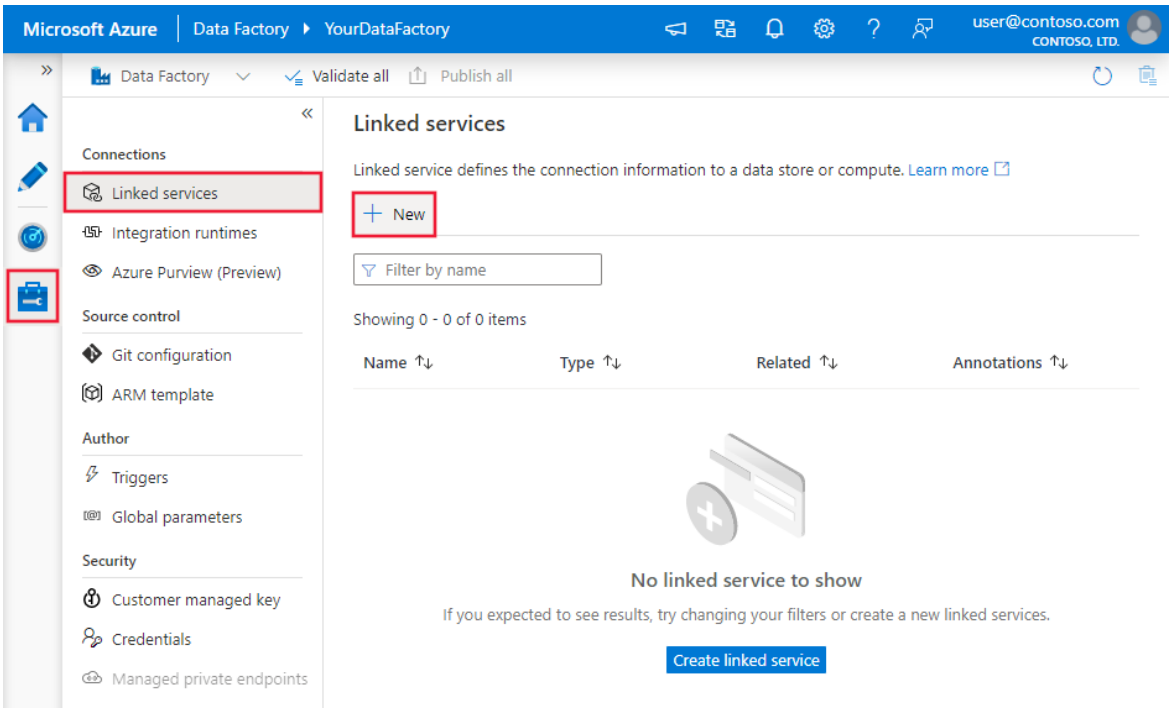
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Azure Search using UI

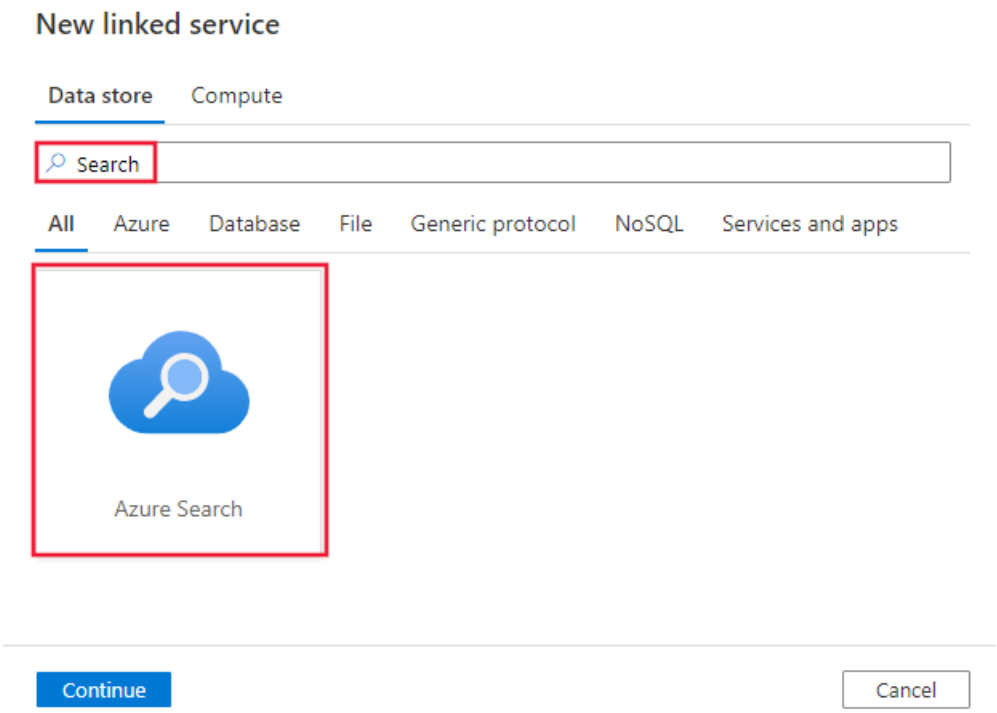
Use the following steps to create a linked service to Azure Search in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

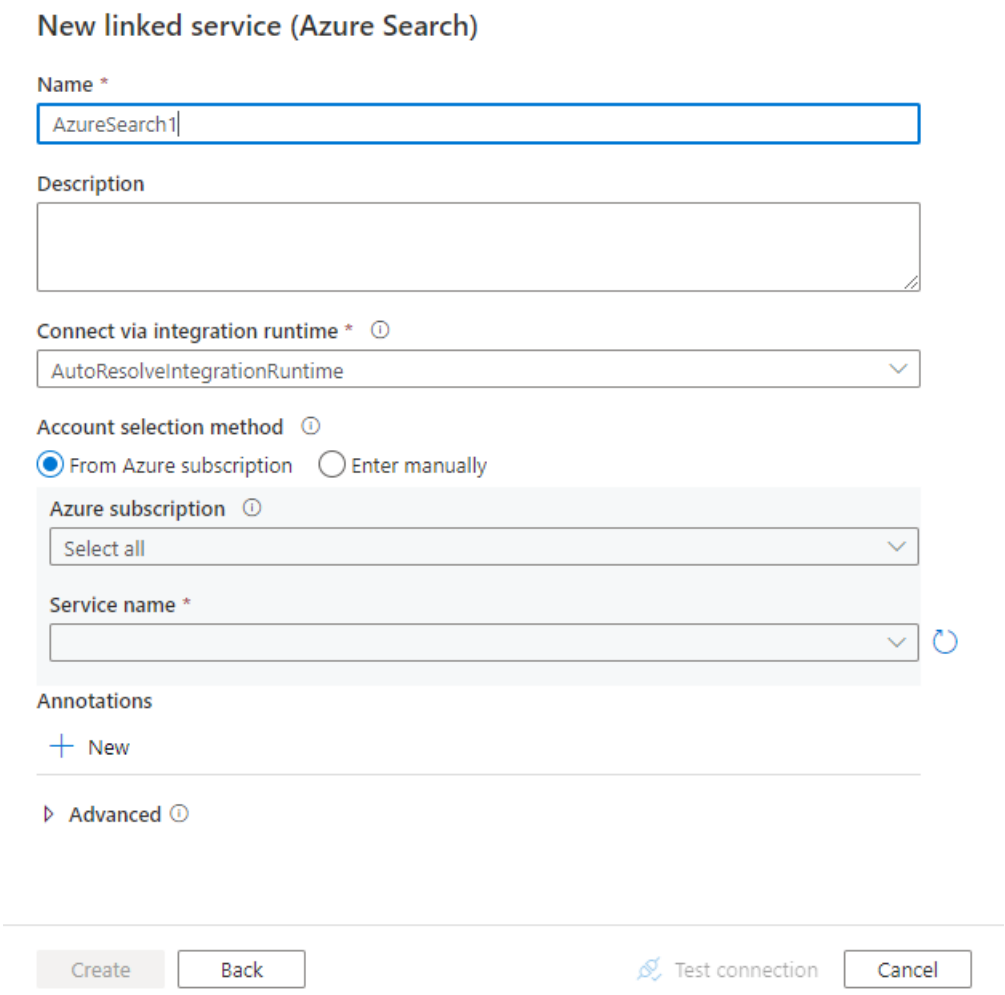
   - Azure Data Factory
   - Azure Synapse



2. Search for Search and select the Azure Search connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Azure Cognitive Search connector.

Linked service properties

The following properties are supported for Azure Cognitive Search linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **AzureSearch**	Yes
url	URL for the search service.	Yes
key	Admin key for the search service. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is located in private network). If not specified, it uses the default Azure Integration Runtime.	No

> **IMPORTANT**
>
> When copying data from a cloud data store into search index, in Azure Cognitive Search linked service, you need to refer an Azure Integration Runtime with explicit region in connactVia. Set the region as the one where your search service resides. Learn more from Azure Integration Runtime.

Example:

```
{
    "name": "AzureSearchLinkedService",
    "properties": {
        "type": "AzureSearch",
        "typeProperties": {
            "url": "https://<service>.search.windows.net",
            "key": {
                "type": "SecureString",
                "value": "<AdminKey>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Azure Cognitive Search dataset.

To copy data into Azure Cognitive Search, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **AzureSearchIndex**	Yes
indexName	Name of the search index. The service does not create the index. The index must exist in Azure Cognitive Search.	Yes

Example:

```
{
    "name": "AzureSearchIndexDataset",
    "properties": {
        "type": "AzureSearchIndex",
        "typeProperties" : {
            "indexName": "products"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Azure Cognitive Search linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Azure Cognitive Search source.

Azure Cognitive Search as sink

To copy data into Azure Cognitive Search, set the source type in the copy activity to **AzureSearchIndexSink**. The following properties are supported in the copy activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **AzureSearchIndexSink**	Yes
writeBehavior	Specifies whether to merge or replace when a document already exists in the index. See the WriteBehavior property. Allowed values are: **Merge** (default), and **Upload**.	No
writeBatchSize	Uploads data into the search index when the buffer size reaches writeBatchSize. See the WriteBatchSize property for details. Allowed values are: integer 1 to 1,000; default is 1000.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

WriteBehavior property

AzureSearchSink upserts when writing data. In other words, when writing a document, if the document key already exists in the search index, Azure Cognitive Search updates the existing document rather than throwing a conflict exception.

The AzureSearchSink provides the following two upsert behaviors (by using AzureSearch SDK):

- **Merge**: combine all the columns in the new document with the existing one. For columns with null value in the new document, the value in the existing one is preserved.
- **Upload**: The new document replaces the existing one. For columns not specified in the new document, the value is set to null whether there is a non-null value in the existing document or not.

The default behavior is **Merge**.

WriteBatchSize Property

Azure Cognitive Search service supports writing documents as a batch. A batch can contain 1 to 1,000 Actions. An action handles one document to perform the upload/merge operation.

Example:

```
"activities":[
    {
        "name": "CopyToAzureSearch",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Azure Cognitive Search output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureSearchIndexSink",
                "writeBehavior": "Merge"
            }
        }
    }
]
```

Data type support

The following table specifies whether an Azure Cognitive Search data type is supported or not.

AZURE COGNITIVE SEARCH DATA TYPE	SUPPORTED IN AZURE COGNITIVE SEARCH SINK
String	Y
Int32	Y
Int64	Y
Double	Y
Boolean	Y
DataTimeOffset	Y
String Array	N
GeographyPoint	N

Currently other data types e.g. ComplexType are not supported. For a full list of Azure Cognitive Search supported data types, see Supported data types (Azure Cognitive Search).

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy and transform data in Azure Cosmos DB (SQL API) by using Azure Data Factory

8/19/2022 • 18 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory to copy data from and to Azure Cosmos DB (SQL API), and use Data Flow to transform data in Azure Cosmos DB (SQL API). To learn more, read the introductory articles for Azure Data Factory and Azure Synapse Analytics.

> **NOTE**
> This connector only support Cosmos DB SQL API. For MongoDB API, refer to connector for Azure Cosmos DB's API for MongoDB. Other API types are not supported now.

Supported capabilities

This Azure Cosmos DB (SQL API) connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/sink)	① ②	✓
Mapping data flow (source/sink)	①	✓
Lookup activity	① ②	✓

① Azure integration runtime ② Self-hosted integration runtime

For Copy activity, this Azure Cosmos DB (SQL API) connector supports:

- Copy data from and to the Azure Cosmos DB SQL API using key, service principal, or managed identities for Azure resources authentications.
- Write to Azure Cosmos DB as **insert** or **upsert**.
- Import and export JSON documents as-is, or copy data from or to a tabular dataset. Examples include a SQL database and a CSV file. To copy documents as-is to or from JSON files or to or from another Azure Cosmos DB collection, see Import and export JSON documents.

Data Factory and Synapse pipelines integrate with the Azure Cosmos DB bulk executor library to provide the best performance when you write to Azure Cosmos DB.

> **TIP**
> The Data Migration video walks you through the steps of copying data from Azure Blob storage to Azure Cosmos DB. The video also describes performance-tuning considerations for ingesting data to Azure Cosmos DB in general.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

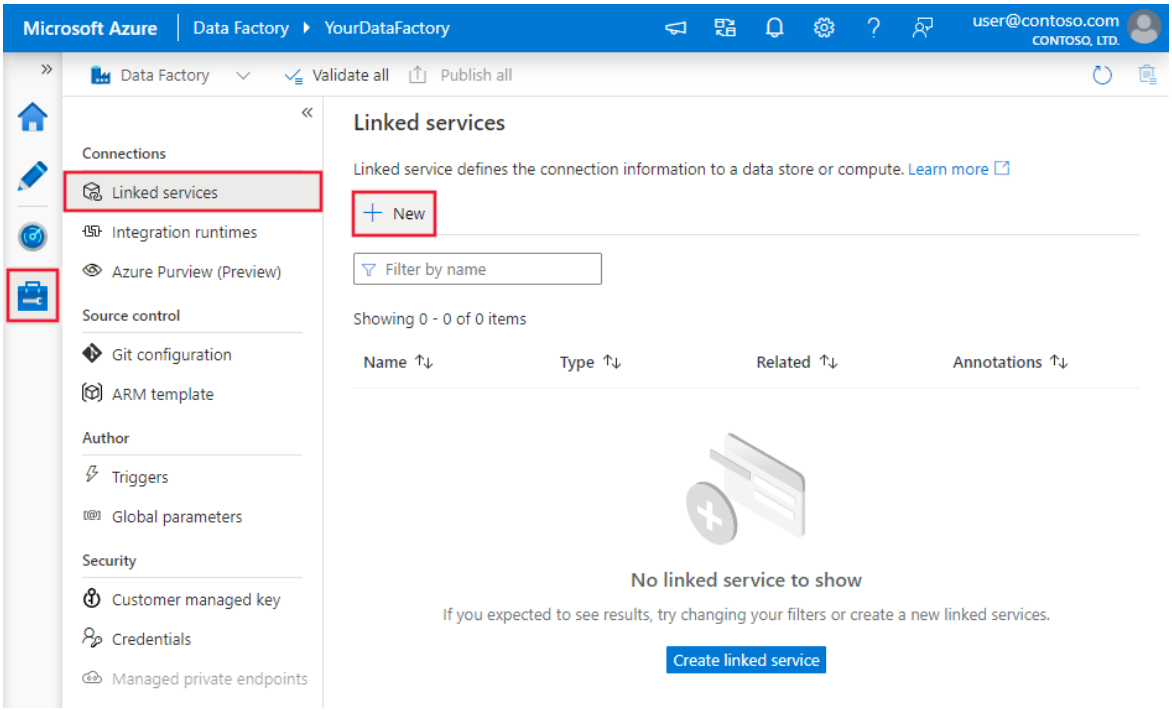
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Azure Cosmos DB using UI

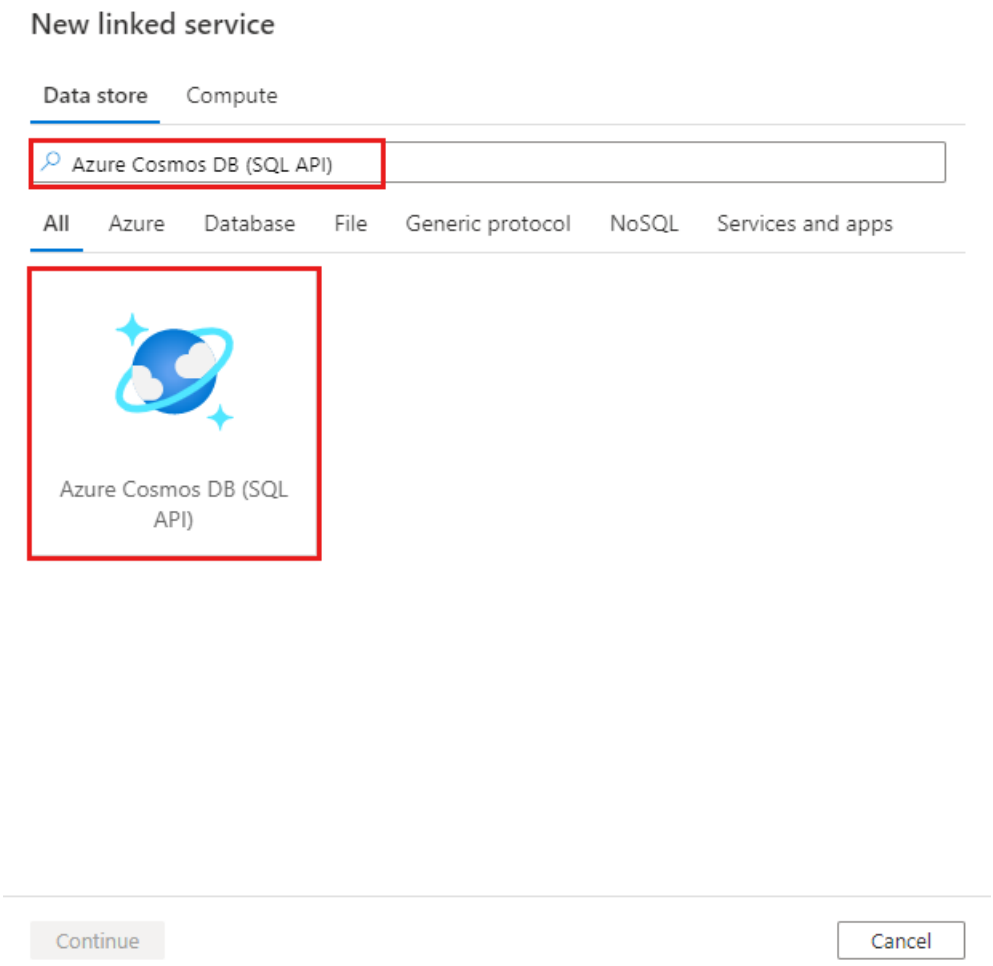
Use the following steps to create a linked service to Azure Cosmos DB in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

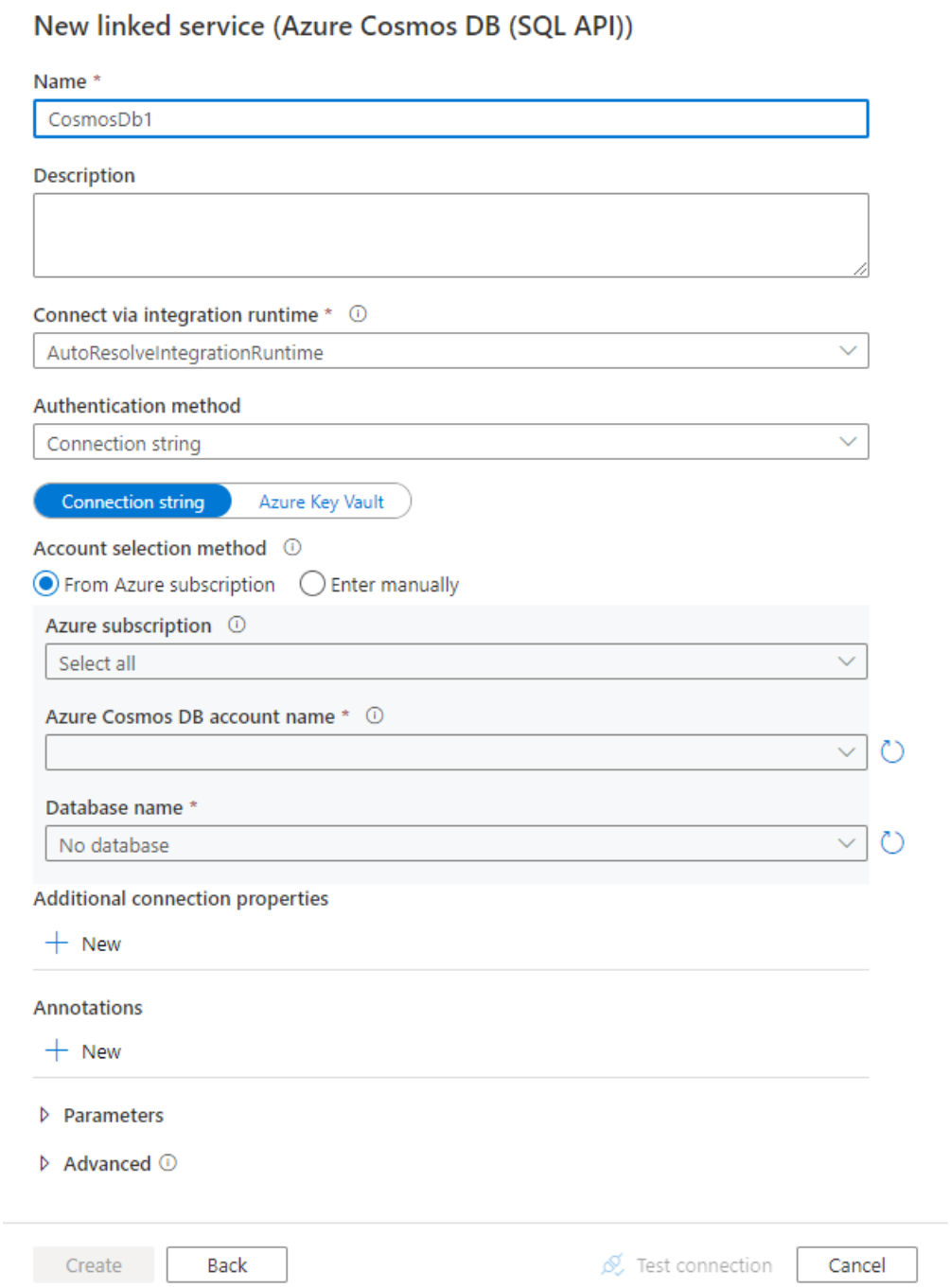
   - Azure Data Factory
   - Azure Synapse



2. Search for Azure Cosmos DB (SQL API) and select the Azure Cosmos DB (SQL API) connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties you can use to define entities that are specific to Azure Cosmos DB (SQL API).

Linked service properties

The Azure Cosmos DB (SQL API) connector supports the following authentication types. See the corresponding sections for details:

- Key authentication
- Service principal authentication
- System-assigned managed identity authentication
- User-assigned managed identity authentication

Key authentication

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **CosmosDb**.	Yes
connectionString	Specify information that's required to connect to the Azure Cosmos DB database. **Note**: You must specify database information in the connection string as shown in the examples that follow. You can also put account key in Azure Key Vault and pull the `accountKey` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to use to connect to the data store. You can use the Azure Integration Runtime or a self-hosted integration runtime (if your data store is located in a private network). If this property isn't specified, the default Azure Integration Runtime is used.	No

Example

```
{
    "name": "CosmosDbSQLAPILinkedService",
    "properties": {
        "type": "CosmosDb",
        "typeProperties": {
            "connectionString": "AccountEndpoint=<EndpointUrl>;AccountKey=<AccessKey>;Database=<Database>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store account key in Azure Key Vault

```
{
    "name": "CosmosDbSQLAPILinkedService",
    "properties": {
        "type": "CosmosDb",
        "typeProperties": {
            "connectionString": "AccountEndpoint=<EndpointUrl>;Database=<Database>",
            "accountKey": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Service principal authentication

> **NOTE**
> Currently, the service principal authentication is not supported in data flow.

To use service principal authentication, follow these steps.

1. Register an application entity in Azure Active Directory (Azure AD) by following the steps in Register your application with an Azure AD tenant. Make note of the following values, which you use to define the linked service:
   - Application ID
   - Application key
   - Tenant ID
2. Grant the service principal proper permission. See examples on how permission works in Cosmos DB from Access control lists on files and directories. More specifically, create a role definition, and assign the role to the service principle via service principle object ID.

These properties are supported for the linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **CosmosDb**.	Yes
accountEndpoint	Specify the account endpoint URL for the Azure Cosmos DB.	Yes
database	Specify the name of the database.	Yes
servicePrincipalId	Specify the application's client ID.	Yes

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalCredentialType	The credential type to use for service principal authentication. Allowed values are **ServicePrincipalKey** and **ServicePrincipalCert**.	Yes
servicePrincipalCredential	The service principal credential. When you use **ServicePrincipalKey** as the credential type, specify the application's key. Mark this field as **SecureString** to store it securely, or reference a secret stored in Azure Key Vault. When you use **ServicePrincipalCert** as the credential, reference a certificate in Azure Key Vault, and ensure the certificate content type is **PKCS #12**.	Yes
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. Retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Yes
azureCloudType	For service principal authentication, specify the type of Azure cloud environment to which your Azure Active Directory application is registered. Allowed values are **AzurePublic**, **AzureChina**, **AzureUsGovernment**, and **AzureGermany**. By default, the service's cloud environment is used.	No
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

Example: using service principal key authentication

You can also store service principal key in Azure Key Vault.

```
{
    "name": "CosmosDbSQLAPILinkedService",
    "properties": {
        "type": "CosmosDb",
        "typeProperties": {
            "accountEndpoint": "<account endpoint>",
            "database": "<database name>",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalCredentialType": "ServicePrincipalKey",
            "servicePrincipalCredential": {
                "type": "SecureString",
                "value": "<service principal key>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: using service principal certificate authentication

```
{
    "name": "CosmosDbSQLAPILinkedService",
    "properties": {
        "type": "CosmosDb",
        "typeProperties": {
            "accountEndpoint": "<account endpoint>",
            "database": "<database name>",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalCredentialType": "ServicePrincipalCert",
            "servicePrincipalCredential": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<AKV reference>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<certificate name in AKV>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

System-assigned managed identity authentication

> **NOTE**
> Currently, the system-assigned managed identity authentication is not supported in data flow.

A data factory or Synapse pipeline can be associated with a system-assigned managed identity for Azure resources, which represents this specific service instance. You can directly use this managed identity for Cosmos DB authentication, similar to using your own service principal. It allows this designated resource to access and copy data to or from your Cosmos DB.

To use system-assigned managed identities for Azure resource authentication, follow these steps.

1. Retrieve the system-assigned managed identity information by copying the value of the **managed identity object ID** generated along with your service.

2. Grant the system-assigned managed identity proper permission. See examples on how permission works in Cosmos DB from Access control lists on files and directories. More specifically, create a role definition, and assign the role to the system-assigned managed identity.

These properties are supported for the linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **CosmosDb**.	Yes
accountEndpoint	Specify the account endpoint URL for the Azure Cosmos DB.	Yes
database	Specify the name of the database.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

Example:

```
{
    "name": "CosmosDbSQLAPILinkedService",
    "properties": {
        "type": "CosmosDb",
        "typeProperties": {
            "accountEndpoint": "<account endpoint>",
            "database": "<database name>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

User-assigned managed identity authentication

> **NOTE**
> Currently, the user-assigned managed identity authentication is not supported in data flow.

A data factory or Synapse pipeline can be associated with a user-assigned managed identities, which represents this specific service instance. You can directly use this managed identity for Cosmos DB authentication, similar to using your own service principal. It allows this designated resource to access and copy data to or from your Cosmos DB.

To use user-assigned managed identities for Azure resource authentication, follow these steps.

1. Create one or multiple user-assigned managed identities and grant the user-assigned managed identity

proper permission. See examples on how permission works in Cosmos DB from Access control lists on files and directories. More specifically, create a role definition, and assign the role to the user-assigned managed identity.

2. Assign one or multiple user-assigned managed identities to your data factory and create credentials for each user-assigned managed identity.

These properties are supported for the linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **CosmosDb**.	Yes
accountEndpoint	Specify the account endpoint URL for the Azure Cosmos DB.	Yes
database	Specify the name of the database.	Yes
credentials	Specify the user-assigned managed identity as the credential object.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

Example:

```
{
    "name": "CosmosDbSQLAPILinkedService",
    "properties": {
        "type": "CosmosDb",
        "typeProperties": {
            "accountEndpoint": "<account endpoint>",
            "database": "<database name>",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties that are available for defining datasets, see Datasets and linked services.

The following properties are supported for Azure Cosmos DB (SQL API) dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the dataset must be set to **CosmosDbSqlApiCollection**.	Yes
collectionName	The name of the Azure Cosmos DB document collection.	Yes

If you use "DocumentDbCollection" type dataset, it is still supported as-is for backward compatibility for Copy and Lookup activity, it's not supported for Data Flow. You are suggested to use the new model going forward.

Example

```
{
    "name": "CosmosDbSQLAPIDataset",
    "properties": {
        "type": "CosmosDbSqlApiCollection",
        "linkedServiceName":{
            "referenceName": "<Azure Cosmos DB linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [],
        "typeProperties": {
            "collectionName": "<collection name>"
        }
    }
}
```

Copy Activity properties

This section provides a list of properties that the Azure Cosmos DB (SQL API) source and sink support. For a full list of sections and properties that are available for defining activities, see Pipelines.

Azure Cosmos DB (SQL API) as source

To copy data from Azure Cosmos DB (SQL API), set the **source** type in Copy Activity to **DocumentDbCollectionSource**.

The following properties are supported in the Copy Activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the copy activity source must be set to **CosmosDbSqlApiSource**.	Yes
query	Specify the Azure Cosmos DB query to read data. Example: `SELECT c.BusinessEntityID, c.Name.First AS FirstName, c.Name.Middle AS MiddleName, c.Name.Last AS LastName, c.Suffix, c.EmailPromotion FROM c WHERE c.ModifiedDate > \"2009-01-01T00:00:00\"`	No If not specified, this SQL statement is executed: `select <columns defined in structure> from mycollection`

PROPERTY	DESCRIPTION	REQUIRED
preferredRegions	The preferred list of regions to connect to when retrieving data from Cosmos DB.	No
pageSize	The number of documents per page of the query result. Default is "-1" which means uses the service side dynamic page size up to 1000.	No
detectDatetime	Whether to detect datetime from the string values in the documents. Allowed values are: **true** (default), **false**.	No

If you use "DocumentDbCollectionSource" type source, it is still supported as-is for backward compatibility. You are suggested to use the new model going forward which provide richer capabilities to copy data from Cosmos DB.

Example

```
"activities":[
    {
        "name": "CopyFromCosmosDBSQLAPI",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Cosmos DB SQL API input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "CosmosDbSqlApiSource",
                "query": "SELECT c.BusinessEntityID, c.Name.First AS FirstName, c.Name.Middle AS MiddleName,
c.Name.Last AS LastName, c.Suffix, c.EmailPromotion FROM c WHERE c.ModifiedDate > \"2009-01-01T00:00:00\"",
                "preferredRegions": [
                    "East US"
                ]
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

When copying data from Cosmos DB, unless you want to export JSON documents as-is, the best practice is to specify the mapping in copy activity. The service honors the mapping you specified on the activity - if a row doesn't contain a value for a column, a null value is provided for the column value. If you don't specify a mapping, the service infers the schema by using the first row in the data. If the first row doesn't contain the full schema, some columns will be missing in the result of the activity operation.

Azure Cosmos DB (SQL API) as sink

To copy data to Azure Cosmos DB (SQL API), set the **sink** type in Copy Activity to **DocumentDbCollectionSink**.

The following properties are supported in the Copy Activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the Copy Activity sink must be set to **CosmosDbSqlApiSink**.	Yes
writeBehavior	Describes how to write data to Azure Cosmos DB. Allowed values: **insert** and **upsert**. The behavior of **upsert** is to replace the document if a document with the same ID already exists; otherwise, insert the document. **Note**: The service automatically generates an ID for a document if an ID isn't specified either in the original document or by column mapping. This means that you must ensure that, for **upsert** to work as expected, your document has an ID.	No (the default is **insert**)
writeBatchSize	The service uses the Azure Cosmos DB bulk executor library to write data to Azure Cosmos DB. The **writeBatchSize** property controls the size of documents the service provides to the library. You can try increasing the value for **writeBatchSize** to improve performance and decreasing the value if your document size being large - see below tips.	No (the default is **10,000**)
disableMetricsCollection	The service collects metrics such as Cosmos DB RUs for copy performance optimization and recommendations. If you are concerned with this behavior, specify `true` to turn it off.	No (default is `false`)
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

TIP

To import JSON documents as-is, refer to Import or export JSON documents section; to copy from tabular-shaped data, refer to Migrate from relational database to Cosmos DB.

> **TIP**
> Cosmos DB limits single request's size to 2MB. The formula is Request Size = Single Document Size * Write Batch Size. If you hit error saying **"Request size is too large."**, **reduce the** `writeBatchSize` **value** in copy sink configuration.

If you use "DocumentDbCollectionSink" type source, it is still supported as-is for backward compatibility. You are suggested to use the new model going forward which provide richer capabilities to copy data from Cosmos DB.

Example

```
"activities":[
    {
        "name": "CopyToCosmosDBSQLAPI",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Document DB output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "CosmosDbSqlApiSink",
                "writeBehavior": "upsert"
            }
        }
    }
]
```

Schema mapping

To copy data from Azure Cosmos DB to tabular sink or reversed, refer to schema mapping.

Mapping data flow properties

When transforming data in mapping data flow, you can read and write to collections in Cosmos DB. For more information, see the source transformation and sink transformation in mapping data flows.

> **NOTE**
> The Azure Cosmos DB serverless is not supported in mapping data flow.

Source transformation

Settings specific to Azure Cosmos DB are available in the **Source Options** tab of the source transformation.

Include system columns: If true, `id`, `_ts`, and other system columns will be included in your data flow metadata from CosmosDB. When updating collections, it is important to include this so that you can grab the existing row ID.

Page size: The number of documents per page of the query result. Default is "-1" which uses the service

dynamic page up to 1000.

Throughput: Set an optional value for the number of RUs you'd like to apply to your CosmosDB collection for each execution of this data flow during the read operation. Minimum is 400.

Preferred regions: Choose the preferred read regions for this process.

Change feed: If true, you will get data from Azure Cosmos DB change feed which is a persistent record of changes to a container in the order they occur from last run automatically. When you set it true, do not set both **Infer drifted column types** and **Allow schema drift** as true at the same time. For more details, see Azure Cosmos DB change feed).

Start from beginning: If true, you will get initial load of full snapshot data in the first run, followed by capturing changed data in next runs. If false, the initial load will be skipped in the first run, followed by capturing changed data in next runs. The setting is aligned with the same setting name in Cosmos DB reference. For more details, see Azure Cosmos DB change feed.

Sink transformation

Settings specific to Azure Cosmos DB are available in the **Settings** tab of the sink transformation.

Update method: Determines what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an alter-row transformation is required to tag rows for those actions. For updates, upserts and deletes, a key column or columns must be set to determine which row to alter.

Collection action: Determines whether to recreate the destination collection prior to writing.

- None: No action will be done to the collection.
- Recreate: The collection will get dropped and recreated

Batch size: An integer that represents how many objects are being written to Cosmos DB collection in each batch. Usually, starting with the default batch size is sufficient. To further tune this value, note:

- Cosmos DB limits single request's size to 2MB. The formula is "Request Size = Single Document Size * Batch Size". If you hit error saying "Request size is too large", reduce the batch size value.
- The larger the batch size, the better throughput the service can achieve, while make sure you allocate enough RUs to empower your workload.

Partition key: Enter a string that represents the partition key for your collection. Example: `/movies/title`

Throughput: Set an optional value for the number of RUs you'd like to apply to your CosmosDB collection for each execution of this data flow. Minimum is 400.

Write throughput budget: An integer that represents the RUs you want to allocate for this Data Flow write operation, out of the total throughput allocated to the collection.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Import and export JSON documents

You can use this Azure Cosmos DB (SQL API) connector to easily:

- Copy documents between two Azure Cosmos DB collections as-is.
- Import JSON documents from various sources to Azure Cosmos DB, including from Azure Blob storage, Azure Data Lake Store, and other file-based stores that the service supports.
- Export JSON documents from an Azure Cosmos DB collection to various file-based stores.

To achieve schema-agnostic copy:

- When you use the Copy Data tool, select the **Export as-is to JSON files or Cosmos DB collection** option.
- When you use activity authoring, choose JSON format with the corresponding file store for source or sink.

Migrate from relational database to Cosmos DB

When migrating from a relational database e.g. SQL Server to Azure Cosmos DB, copy activity can easily map tabular data from source to flatten JSON documents in Cosmos DB. In some cases, you may want to redesign the data model to optimize it for the NoSQL use-cases according to Data modeling in Azure Cosmos DB, for example, to de-normalize the data by embedding all of the related sub-items within one JSON document. For such case, refer to this article with a walk-through on how to achieve it using the copy activity.

Azure Cosmos DB change feed

Azure Data Factory can get data from Azure Cosmos DB change feed by enabling it in the mapping data flow source transformation. With this connector option, you can read change feeds and apply transformations before loading transformed data into destination datasets of your choice. You do not have to use Azure functions to read the change feed and then write custom transformations. You can use this option to move data from one container to another, prepare change feed driven material views for fit purpose or automate container backup or recovery based on change feed, and enable many more such use cases using visual drag and drop capability of Azure Data Factory.

Make sure you keep the pipeline and activity name unchanged, so that the checkpoint can be recorded by ADF for you to get changed data from the last run automatically. If you change your pipeline name or activity name, the checkpoint will be reset, which leads you to start from beginning or get changes from now in the next run.

When you debug the pipeline, this feature works the same. Be aware that the checkpoint will be reset when you refresh your browser during the debug run. After you are satisfied with the pipeline result from debug run, you can go ahead to publish and trigger the pipeline. At the moment when you first time trigger your published pipeline, it automatically restarts from the beginning or gets changes from now on.

In the monitoring section, you always have the chance to rerun a pipeline. When you are doing so, the changed data is always captured from the previous checkpoint of your selected pipeline run.

Next steps

For a list of data stores that Copy Activity supports as sources and sinks, see supported data stores.

Copy data to or from Azure Cosmos DB's API for MongoDB using Azure Data Factory or Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from and to Azure Cosmos DB's API for MongoDB. The article builds on Copy Activity, which presents a general overview of Copy Activity.

> **NOTE**
> This connector only supports copy data to/from Azure Cosmos DB's API for MongoDB. For SQL API, refer to Cosmos DB SQL API connector. Other API types are not supported now.

Supported capabilities

This Azure Cosmos DB's API for MongoDB connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/sink)	① ②	✓

① Azure integration runtime ② Self-hosted integration runtime

You can copy data from Azure Cosmos DB's API for MongoDB to any supported sink data store, or copy data from any supported source data store to Azure Cosmos DB's API for MongoDB. For a list of data stores that Copy Activity supports as sources and sinks, see Supported data stores and formats.

You can use the Azure Cosmos DB's API for MongoDB connector to:

- Copy data from and to the Azure Cosmos DB's API for MongoDB.
- Write to Azure Cosmos DB as **insert** or **upsert**.
- Import and export JSON documents as-is, or copy data from or to a tabular dataset. Examples include a SQL database and a CSV file. To copy documents as-is to or from JSON files or to or from another Azure Cosmos DB collection, see Import or export JSON documents.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

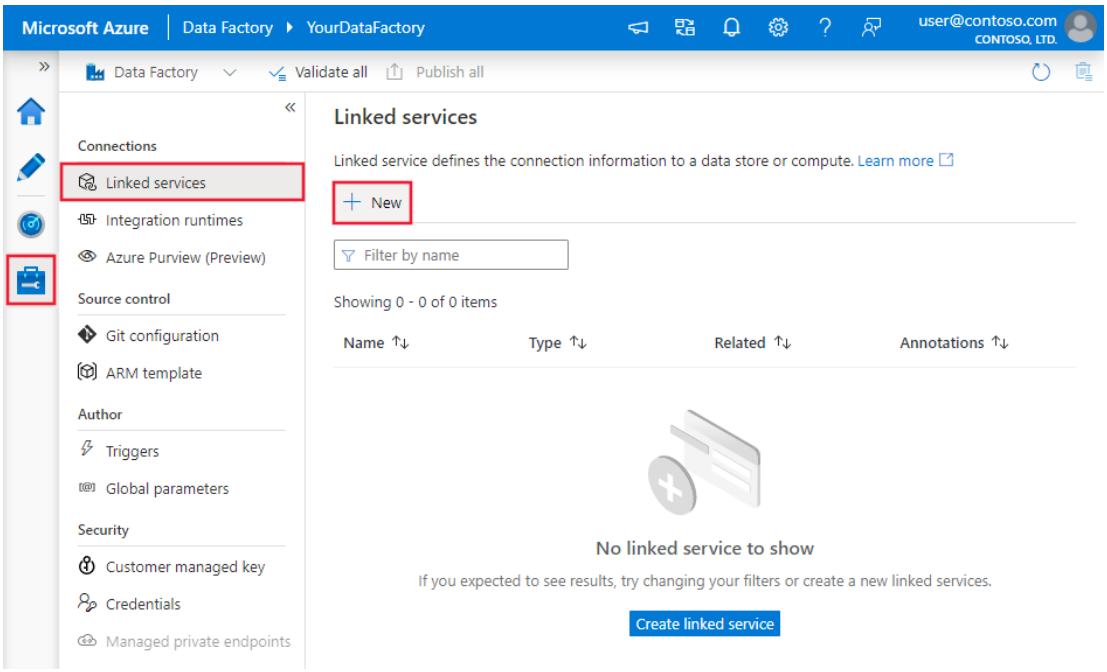
Create a linked service to Azure Cosmos DB's API for MongoDB using

UI

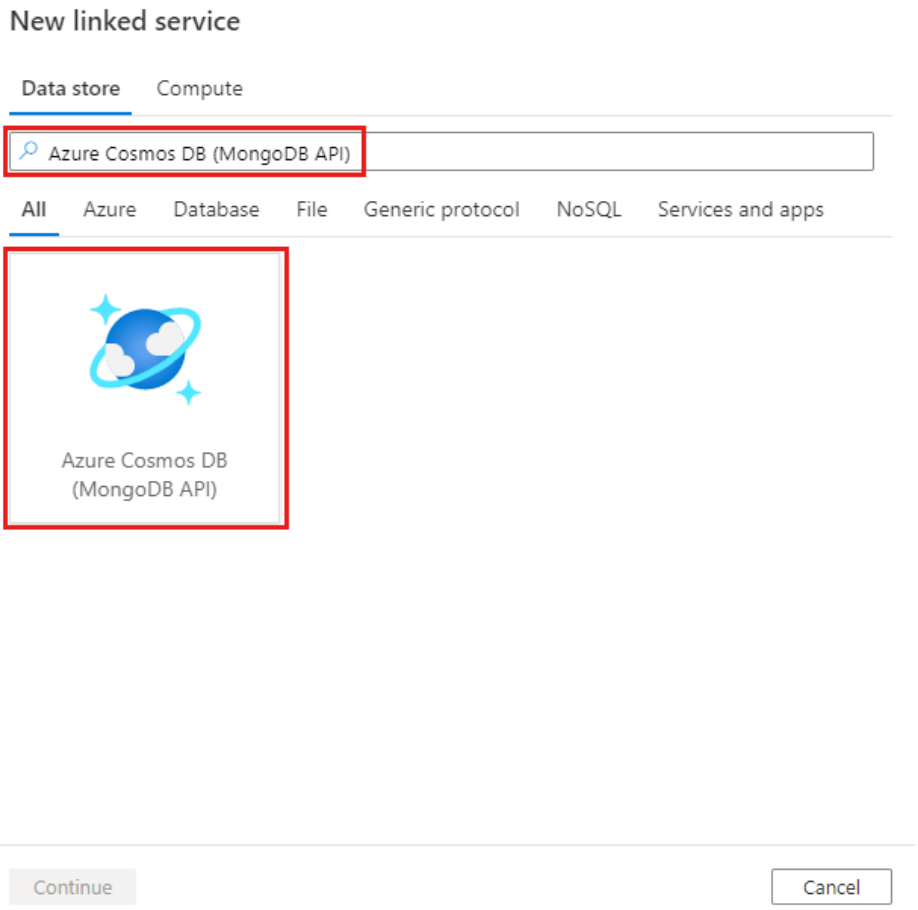
Use the following steps to create a linked service to Azure Cosmos DB's API for MongoDB in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

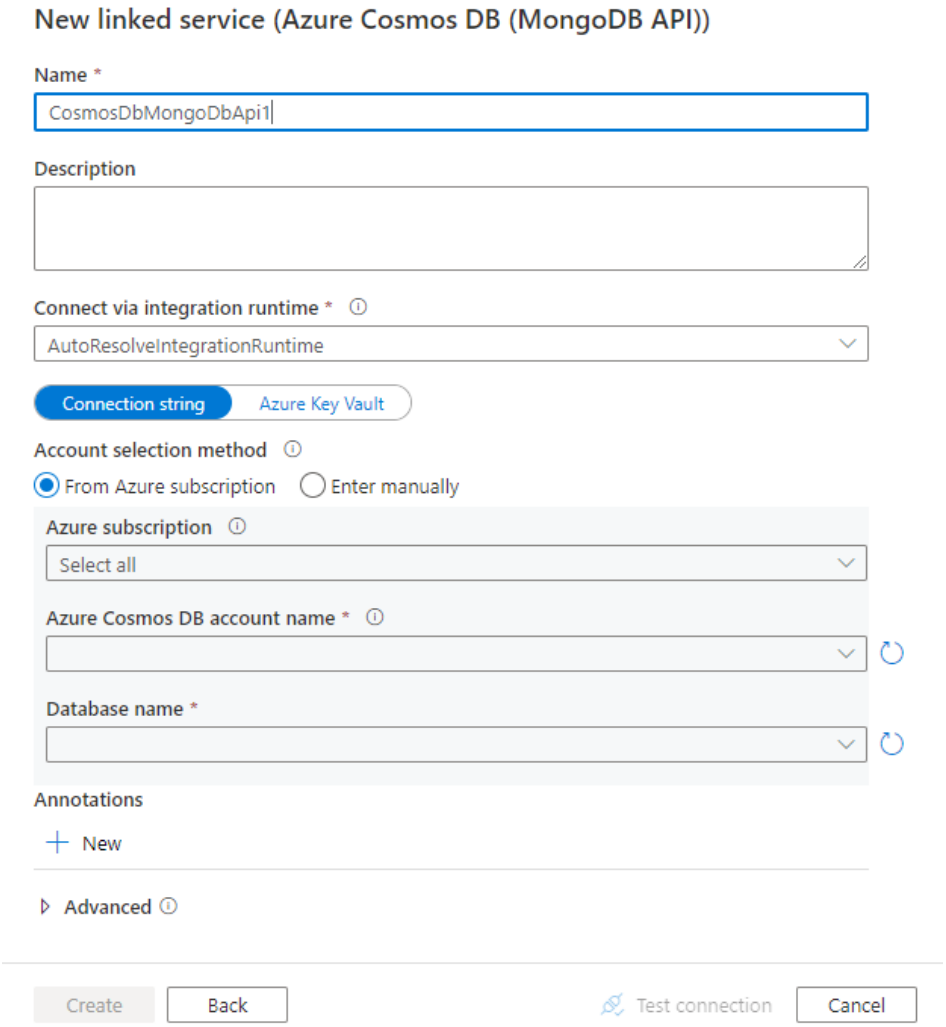
   - Azure Data Factory
   - Azure Synapse



2. Search for Azure Cosmos DB (MongoDB API) and select the Azure Cosmos DB's API for MongoDB connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties you can use to define Data Factory entities that are specific to Azure Cosmos DB's API for MongoDB.

Linked service properties

The following properties are supported for the Azure Cosmos DB's API for MongoDB linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **CosmosDbMongoDbApi**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	Specify the connection string for your Azure Cosmos DB's API for MongoDB. You can find it in the Azure portal -> your Cosmos DB blade -> primary or secondary connection string. For 3.2 server version, the string pattern is `mongodb://<cosmosdb-name>:<password>@<cosmosdb-name>.documents.azure.com:10255/?ssl=true&replicaSet=globaldb`. For 3.6+ server versions, the string pattern is `mongodb://<cosmosdb-name>:<password>@<cosmosdb-name>.mongo.cosmos.azure.com:10255/?ssl=true&replicaSet=globaldb&retrywrites=false&maxIdleTimeMS=120000&appName=@<cosmos name>@`. You can also put a password in Azure Key Vault and pull the `password` configuration out of the connection string. Refer to Store credentials in Azure Key Vault with more details.	Yes
database	Name of the database that you want to access.	Yes
isServerVersionAbove32	Specify whether the server version is above 3.2. Allowed values are **true** and **false**(default). This will determine the driver to use in the service.	Yes
connectVia	The Integration Runtime to use to connect to the data store. You can use the Azure Integration Runtime or a self-hosted integration runtime (if your data store is located in a private network). If this property isn't specified, the default Azure Integration Runtime is used.	No

Example

```
{
    "name": "CosmosDbMongoDBAPILinkedService",
    "properties": {
        "type": "CosmosDbMongoDbApi",
        "typeProperties": {
            "connectionString": "mongodb://<cosmosdb-name>:<password>@<cosmosdb-
name>.documents.azure.com:10255/?ssl=true&replicaSet=globaldb",
            "database": "myDatabase",
            "isServerVersionAbove32": "false"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties that are available for defining datasets, see Datasets and linked services. The following properties are supported for Azure Cosmos DB's API for MongoDB dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the dataset must be set to **CosmosDbMongoDbApiCollection**.	Yes
collectionName	The name of the Azure Cosmos DB collection.	Yes

Example

```
{
    "name": "CosmosDbMongoDBAPIDataset",
    "properties": {
        "type": "CosmosDbMongoDbApiCollection",
        "typeProperties": {
            "collectionName": "<collection name>"
        },
        "schema": [],
        "linkedServiceName":{
            "referenceName": "<Azure Cosmos DB's API for MongoDB linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy Activity properties

This section provides a list of properties that the Azure Cosmos DB's API for MongoDB source and sink support.

For a full list of sections and properties that are available for defining activities, see Pipelines.

Azure Cosmos DB's API for MongoDB as source

The following properties are supported in the Copy Activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the copy activity source must be set to **CosmosDbMongoDbApiSource**.	Yes
filter	Specifies selection filter using query operators. To return all documents in a collection, omit this parameter or pass an empty document ({}).	No
cursorMethods.project	Specifies the fields to return in the documents for projection. To return all fields in the matching documents, omit this parameter.	No
cursorMethods.sort	Specifies the order in which the query returns matching documents. Refer to cursor.sort().	No

PROPERTY	DESCRIPTION	REQUIRED
cursorMethods.limit	Specifies the maximum number of documents the server returns. Refer to cursor.limit().	No
cursorMethods.skip	Specifies the number of documents to skip and from where MongoDB begins to return results. Refer to cursor.skip().	No
batchSize	Specifies the number of documents to return in each batch of the response from MongoDB instance. In most cases, modifying the batch size will not affect the user or the application. Cosmos DB limits each batch cannot exceed 40MB in size, which is the sum of the batchSize number of documents' size, so decrease this value if your document size being large.	No (the default is **100**)

TIP

ADF support consuming BSON document in **Strict mode**. Make sure your filter query is in Strict mode instead of Shell mode. More description can be found at MongoDB manual.

Example

```
"activities":[
    {
        "name": "CopyFromCosmosDBMongoDBAPI",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Azure Cosmos DB's API for MongoDB input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "CosmosDbMongoDbApiSource",
                "filter": "{datetimeData: {$gte: ISODate(\"2018-12-11T00:00:00.000Z\"),$lt: ISODate(\"2018-
12-12T00:00:00.000Z\")}, _id: ObjectId(\"5acd7c3d0000000000000000\") }",
                "cursorMethods": {
                    "project": "{ _id : 1, name : 1, age: 1, datetimeData: 1 }",
                    "sort": "{ age : 1 }",
                    "skip": 3,
                    "limit": 3
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Azure Cosmos DB's API for MongoDB as sink

The following properties are supported in the Copy Activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the Copy Activity sink must be set to **CosmosDbMongoDbApiSink**.	Yes
writeBehavior	Describes how to write data to Azure Cosmos DB. Allowed values: **insert** and **upsert**. The behavior of **upsert** is to replace the document if a document with the same `_id` already exists; otherwise, insert the document. **Note**: The service automatically generates an `_id` for a document if an `_id` isn't specified either in the original document or by column mapping. This means that you must ensure that, for **upsert** to work as expected, your document has an ID.	No (the default is **insert**)
writeBatchSize	The **writeBatchSize** property controls the size of documents to write in each batch. You can try increasing the value for **writeBatchSize** to improve performance and decreasing the value if your document size being large.	No (the default is **10,000**)
writeBatchTimeout	The wait time for the batch insert operation to finish before it times out. The allowed value is timespan.	No (the default is **00:30:00** - 30 minutes)

TIP

To import JSON documents as-is, refer to Import or export JSON documents section; to copy from tabular-shaped data, refer to Schema mapping.

Example

```
"activities":[
    {
        "name": "CopyToCosmosDBMongoDBAPI",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Document DB output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "CosmosDbMongoDbApiSink",
                "writeBehavior": "upsert"
            }
        }
    }
]
```

Import and export JSON documents

You can use this Azure Cosmos DB connector to easily:

- Copy documents between two Azure Cosmos DB collections as-is.
- Import JSON documents from various sources to Azure Cosmos DB, including from MongoDB, Azure Blob storage, Azure Data Lake Store, and other file-based stores that the service supports.
- Export JSON documents from an Azure Cosmos DB collection to various file-based stores.

To achieve schema-agnostic copy:

- When you use the Copy Data tool, select the **Export as-is to JSON files or Cosmos DB collection** option.
- When you use activity authoring, choose JSON format with the corresponding file store for source or sink.

Schema mapping

To copy data from Azure Cosmos DB's API for MongoDB to tabular sink or reversed, refer to schema mapping.

Specifically for writing into Cosmos DB, to make sure you populate Cosmos DB with the right object ID from your source data, for example, you have an "id" column in SQL database table and want to use the value of that as the document ID in MongoDB for insert/upsert, you need to set the proper schema mapping according to MongoDB strict mode definition (`_id.$oid`) as the following:

After copy activity execution, below BSON ObjectId is generated in sink:

```
{
    "_id": ObjectId("592e07800000000000000000")
}
```

Next steps

For a list of data stores that Copy Activity supports as sources and sinks, see supported data stores.

Copy data to or from Azure Data Explorer using Azure Data Factory or Synapse Analytics

8/19/2022 • 12 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article describes how to use the copy activity in Azure Data Factory and Synapse Analytics pipelines to copy data to or from Azure Data Explorer. It builds on the copy activity overview article, which offers a general overview of copy activity.

> **TIP**
> To learn more about Azure Data Explorer integration with the service generally read Integrate Azure Data Explorer.

Supported capabilities

This Azure Data Explorer connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Mapping data flow (source/sink)	①
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

You can copy data from any supported source data store to Azure Data Explorer. You can also copy data from Azure Data Explorer to any supported sink data store. For a list of data stores that the copy activity supports as sources or sinks, see the Supported data stores table.

> **NOTE**
> Copying data to or from Azure Data Explorer through an on-premises data store by using self-hosted integration runtime is supported in version 3.14 and later.

With the Azure Data Explorer connector, you can do the following:

- Copy data by using Azure Active Directory (Azure AD) application token authentication with a **service principal**.
- As a source, retrieve data by using a KQL (Kusto) query.
- As a sink, append data to a destination table.

Getting started

TIP

For a walkthrough of Azure Data Explorer connector, see Copy data to/from Azure Data Explorer and Bulk copy from a database to Azure Data Explorer.

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

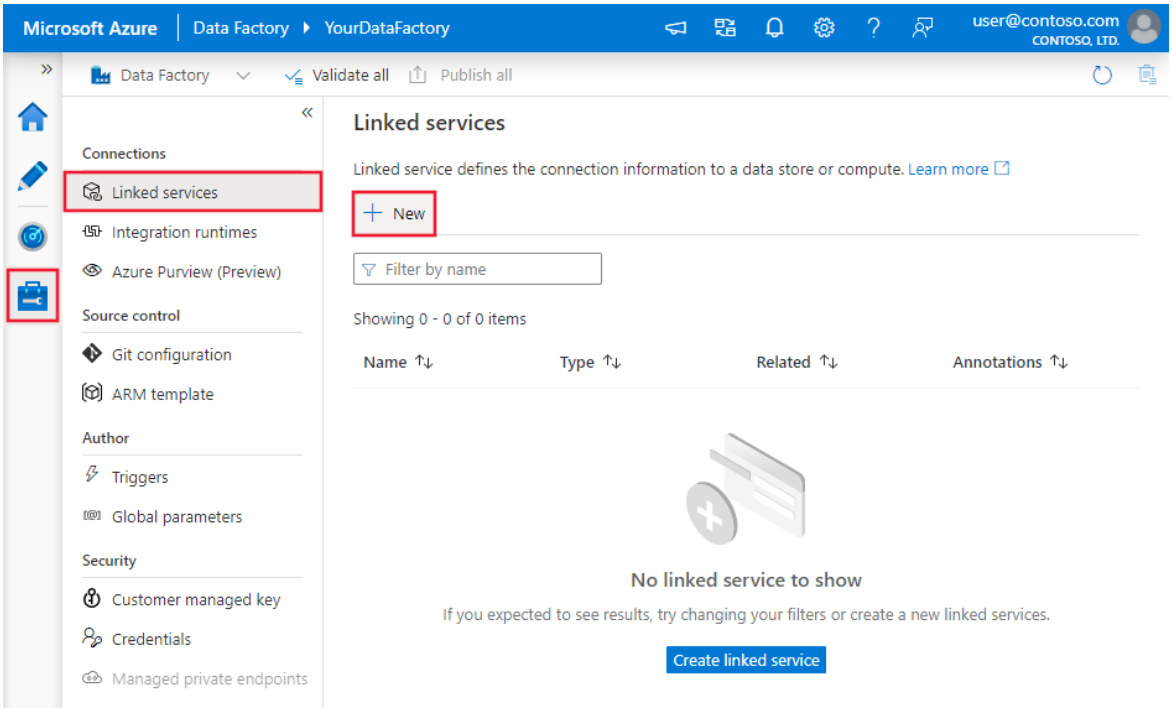
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Azure Data Explorer using UI

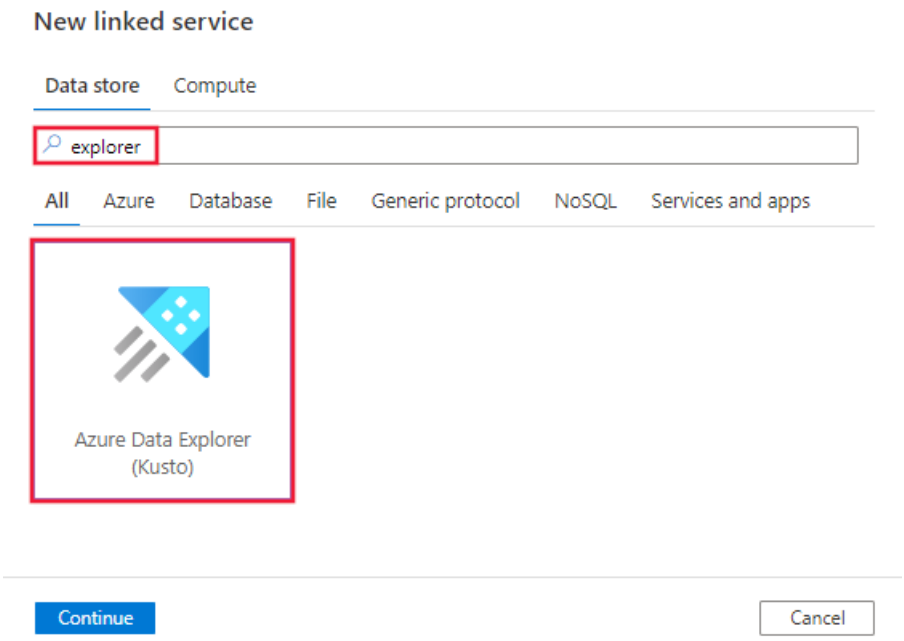
Use the following steps to create a linked service to Azure Data Explorer in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

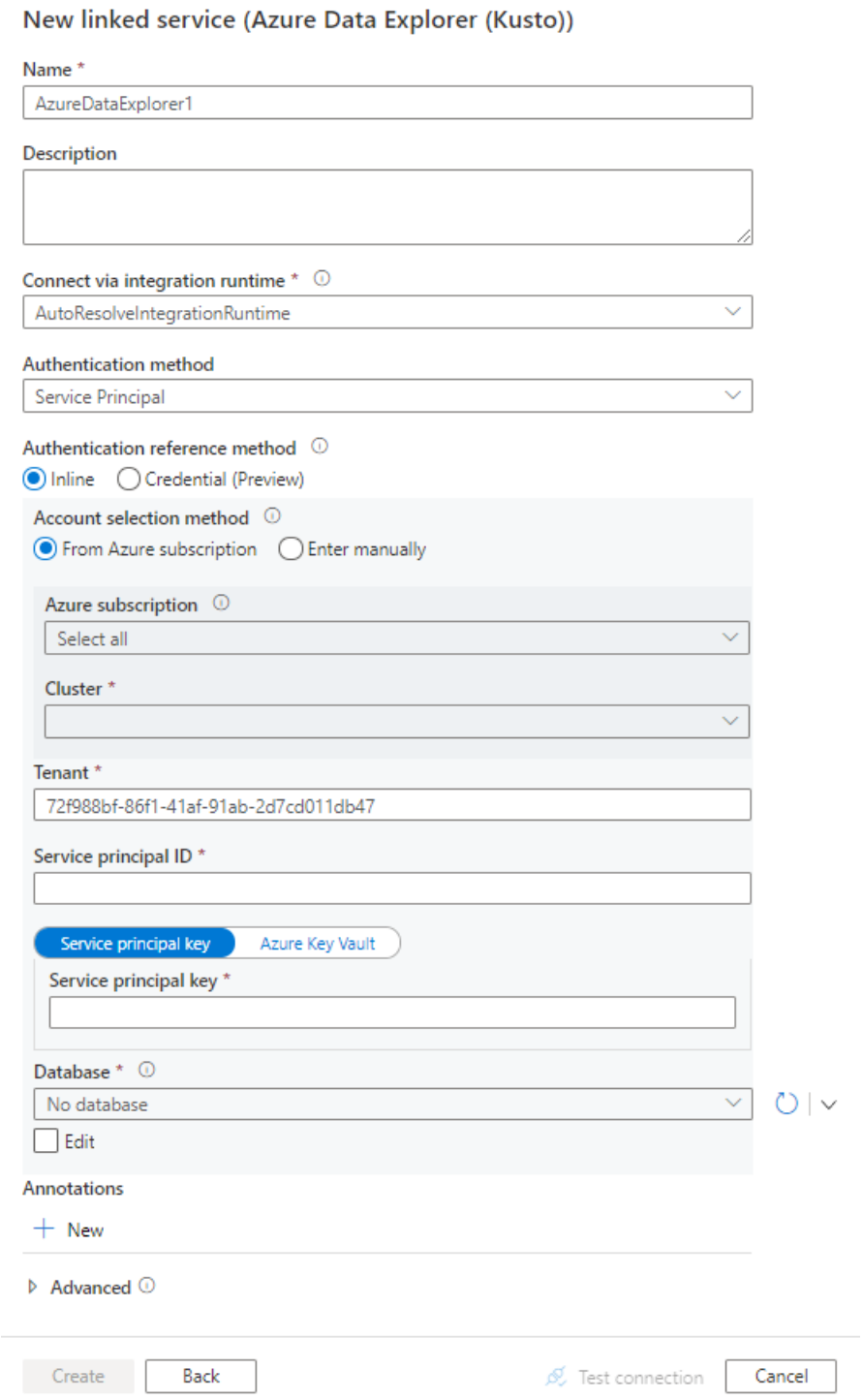
   - Azure Data Factory
   - Azure Synapse



2. Search for Explorer and select the Azure Data Explorer (Kusto) connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to Azure Data Explorer connector.

Linked service properties

The Azure Data Explorer connector supports the following authentication types. See the corresponding sections for details:

- Service principal authentication
- System-assigned managed identity authentication
- User-assigned managed identity authentication

Service principal authentication

To use service principal authentication, follow these steps to get a service principal and to grant permissions:

1. Register an application entity in Azure Active Directory by following the steps in Register your application with an Azure AD tenant. Make note of the following values, which you use to define the linked service:
   - Application ID
   - Application key
   - Tenant ID
2. Grant the service principal the correct permissions in Azure Data Explorer. See Manage Azure Data Explorer database permissions for detailed information about roles and permissions and about managing permissions. In general, you must:
   - **As source**, grant at least the **Database viewer** role to your database
   - **As sink**, grant at least the **Database ingestor** role to your database

> **NOTE**
> When you use the UI to author, by default your login user account is used to list Azure Data Explorer clusters, databases, and tables. You can choose to list the objects using the service principal by clicking the dropdown next to the refresh button, or manually enter the name if you don't have permission for these operations.

The following properties are supported for the Azure Data Explorer linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **AzureDataExplorer**.	Yes
endpoint	Endpoint URL of the Azure Data Explorer cluster, with the format as `https://<clusterName>.<regionName>.kusto.windows.net`.	Yes
database	Name of database.	Yes
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. This is known as "Authority ID" in Kusto connection string. Retrieve it by hovering the mouse pointer in the upper-right corner of the Azure portal.	Yes
servicePrincipalId	Specify the application's client ID. This is known as "AAD application client ID" in Kusto connection string.	Yes
servicePrincipalKey	Specify the application's key. This is known as "AAD application key" in Kusto connection string. Mark this field as a **SecureString** to store it securely, or reference secure data stored in Azure Key Vault.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

Example: using service principal key authentication

```
{
    "name": "AzureDataExplorerLinkedService",
    "properties": {
        "type": "AzureDataExplorer",
        "typeProperties": {
            "endpoint": "https://<clusterName>.<regionName>.kusto.windows.net ",
            "database": "<database name>",
            "tenant": "<tenant name/id e.g. microsoft.onmicrosoft.com>",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            }
        }
    }
}
```

System-assigned managed identity authentication

To learn more about managed identities for Azure resources, see Managed identities for Azure resources.

To use system-assigned managed identity authentication, follow these steps to grant permissions:

1. Retrieve the managed identity information by copying the value of the **managed identity object ID** generated along with your factory or Synapse workspace.
2. Grant the managed identity the correct permissions in Azure Data Explorer. See Manage Azure Data Explorer database permissions for detailed information about roles and permissions and about managing permissions. In general, you must:
   - **As source**, grant at least the **Database viewer** role to your database
   - **As sink**, grant at least the **Database ingestor** role to your database

> **NOTE**
> When you use the UI to author, your login user account is used to list Azure Data Explorer clusters, databases, and tables. Manually enter the name if you don't have permission for these operations.

The following properties are supported for the Azure Data Explorer linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **AzureDataExplorer**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
endpoint	Endpoint URL of the Azure Data Explorer cluster, with the format as `https://<clusterName>.<regionName>.kusto.windows.net`.	Yes
database	Name of database.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

Example: using system-assigned managed identity authentication

```
{
    "name": "AzureDataExplorerLinkedService",
    "properties": {
        "type": "AzureDataExplorer",
        "typeProperties": {
            "endpoint": "https://<clusterName>.<regionName>.kusto.windows.net ",
            "database": "<database name>",
        }
    }
}
```

User-assigned managed identity authentication

To learn more about managed identities for Azure resources, see Managed identities for Azure resources

To use user-assigned managed identity authentication, follow these steps:

1. Create one or multiple user-assigned managed identities and grant permission in Azure Data Explorer. See Manage Azure Data Explorer database permissions for detailed information about roles and permissions and about managing permissions. In general, you must:
   - **As source**, grant at least the **Database viewer** role to your database
   - **As sink**, grant at least the **Database ingestor** role to your database
2. Assign one or multiple user-assigned managed identities to your data factory or Synapse workspace, and create credentials for each user-assigned managed identity.

The following properties are supported for the Azure Data Explorer linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **AzureDataExplorer**.	Yes
endpoint	Endpoint URL of the Azure Data Explorer cluster, with the format as `https://<clusterName>.<regionName>.kusto.windows.net`.	Yes

PROPERTY	DESCRIPTION	REQUIRED
database	Name of database.	Yes
credentials	Specify the user-assigned managed identity as the credential object.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

Example: using user-assigned managed identity authentication

```
{
    "name": "AzureDataExplorerLinkedService",
    "properties": {
        "type": "AzureDataExplorer",
        "typeProperties": {
            "endpoint": "https://<clusterName>.<regionName>.kusto.windows.net ",
            "database": "<database name>",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see Datasets. This section lists properties that the Azure Data Explorer dataset supports.

To copy data to Azure Data Explorer, set the type property of the dataset to **AzureDataExplorerTable**.

The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **AzureDataExplorerTable**.	Yes
table	The name of the table that the linked service refers to.	Yes for sink; No for source

Dataset properties example:

```
{
   "name": "AzureDataExplorerDataset",
    "properties": {
        "type": "AzureDataExplorerTable",
        "typeProperties": {
            "table": "<table name>"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Azure Data Explorer linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see Pipelines and activities. This section provides a list of properties that Azure Data Explorer sources and sinks support.

Azure Data Explorer as source

To copy data from Azure Data Explorer, set the **type** property in the Copy activity source to **AzureDataExplorerSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the copy activity source must be set to: **AzureDataExplorerSource**	Yes
query	A read-only request given in a KQL format. Use the custom KQL query as a reference.	Yes
queryTimeout	The wait time before the query request times out. Default value is 10 min (00:10:00); allowed max value is 1 hour (01:00:00).	No
noTruncation	Indicates whether to truncate the returned result set. By default, result is truncated after 500,000 records or 64 megabytes (MB). Truncation is strongly recommended to ensure the correct behavior of the activity.	No

> **NOTE**
>
> By default, Azure Data Explorer source has a size limit of 500,000 records or 64 MB. To retrieve all the records without truncation, you can specify `set notruncation;` at the beginning of your query. For more information, see Query limits.

Example:

```
"activities":[
    {
        "name": "CopyFromAzureDataExplorer",
        "type": "Copy",
        "typeProperties": {
            "source": {
                "type": "AzureDataExplorerSource",
                "query": "TestTable1 | take 10",
                "queryTimeout": "00:10:00"
            },
            "sink": {
                "type": "<sink type>"
            }
        },
        "inputs": [
            {
                "referenceName": "<Azure Data Explorer input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ]
    }
]
```

Azure Data Explorer as sink

To copy data to Azure Data Explorer, set the type property in the copy activity sink to **AzureDataExplorerSink**. The following properties are supported in the copy activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the copy activity sink must be set to: **AzureDataExplorerSink**.	Yes
ingestionMappingName	Name of a pre-created mapping on a Kusto table. To map the columns from source to Azure Data Explorer (which applies to all supported source stores and formats, including CSV/JSON/Avro formats), you can use the copy activity column mapping (implicitly by name or explicitly as configured) and/or Azure Data Explorer mappings.	No
additionalProperties	A property bag which can be used for specifying any of the ingestion properties which aren't being set already by the Azure Data Explorer Sink. Specifically, it can be useful for specifying ingestion tags. Learn more from Azure Data Explore data ingestion doc.	No

Example:

```
"activities":[
    {
        "name": "CopyToAzureDataExplorer",
        "type": "Copy",
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureDataExplorerSink",
                "ingestionMappingName": "<optional Azure Data Explorer mapping name>",
                "additionalProperties": {<additional settings for data ingestion>}
            }
        },
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Azure Data Explorer output dataset name>",
                "type": "DatasetReference"
            }
        ]
    }
]
```

Mapping data flow properties

When transforming data in mapping data flow, you can read from and write to tables in Azure Data Explorer. For more information, see the source transformation and sink transformation in mapping data flows. You can choose to use an Azure Data Explorer dataset or an inline dataset as source and sink type.

Source transformation

The below table lists the properties supported by Azure Data Explorer source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Table	If you select Table as input, data flow will fetch all the data from the table specified in the Azure Data Explorer dataset or in the source options when using inline dataset.	No	String	*(for inline dataset only)* tableName
Query	A read-only request given in a KQL format. Use the custom KQL query as a reference.	No	String	query

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Timeout	The wait time before the query request times out. Default is '172000' (2 days)	No	Integer	timeout

Azure Data Explorer source script examples

When you use Azure Data Explorer dataset as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 query: 'table | take 10',
 format: 'query') ~> AzureDataExplorerSource
```

If you use inline dataset, the associated data flow script is:

```
source(allowSchemaDrift: true,
    validateSchema: false,
    format: 'query',
    query: 'table | take 10',
    store: 'azuredataexplorer') ~> AzureDataExplorerSource
```

Sink transformation

The below table lists the properties supported by Azure Data Explorer sink. You can edit these properties in the **Settings** tab. When using inline dataset, you will see additional settings, which are the same as the properties described in dataset properties section.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Table action	Determines whether to recreate or remove all rows from the destination table prior to writing. - **None**: No action will be done to the table. - **Recreate**: The table will get dropped and recreated. Required if creating a new table dynamically. - **Truncate**: All rows from the target table will get removed.	No	`true` or `false`	recreate truncate

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Pre and Post SQL scripts	Specify multiple Kusto control commands scripts that will execute before (pre-processing) and after (post-processing) data is written to your sink database.	No	String	preSQLs; postSQLs
Timeout	The wait time before the query request times out. Default is '172000' (2 days)	No	Integer	timeout

Azure Data Explorer sink script examples

When you use Azure Data Explorer dataset as sink type, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
 validateSchema: false,
 format: 'table',
 preSQLs:['pre SQL scripts'],
 postSQLs:['post SQL script'],
 skipDuplicateMapInputs: true,
 skipDuplicateMapOutputs: true) ~> AzureDataExplorerSink
```

If you use inline dataset, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    format: 'table',
    store: 'azuredataexplorer',
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> AzureDataExplorerSink
```

Lookup activity properties

For more information about the properties, see Lookup activity.

Next steps

- For a list of data stores that the copy activity supports as sources and sinks, see supported data stores.
- Learn more about how to copy data from Azure Data Factory and Synapse Analytics to Azure Data Explorer.

Copy data to or from Azure Data Lake Storage Gen1 using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 28 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to copy data to and from Azure Data Lake Storage Gen1. To learn more, read the introductory article for Azure Data Factory or Azure Synapse Analytics.

Supported capabilities

This Azure Data Lake Storage Gen1 connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Mapping data flow (source/sink)	①
Lookup activity	① ②
GetMetadata activity	① ②
Delete activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

Specifically, with this connector you can:

- Copy files by using one of the following methods of authentication: service principal or managed identities for Azure resources.
- Copy files as is or parse or generate files with the supported file formats and compression codecs.
- Preserve ACLs when copying into Azure Data Lake Storage Gen2.

> **IMPORTANT**
> If you copy data by using the self-hosted integration runtime, configure the corporate firewall to allow outbound traffic to `<ADLS account name>.azuredatalakestore.net` and `login.microsoftonline.com/<tenant>/oauth2/token` on port 443. The latter is the Azure Security Token Service that the integration runtime needs to communicate with to get the access token.

Get started

> **TIP**
> For a walk-through of how to use the Azure Data Lake Store connector, see Load data into Azure Data Lake Store.

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

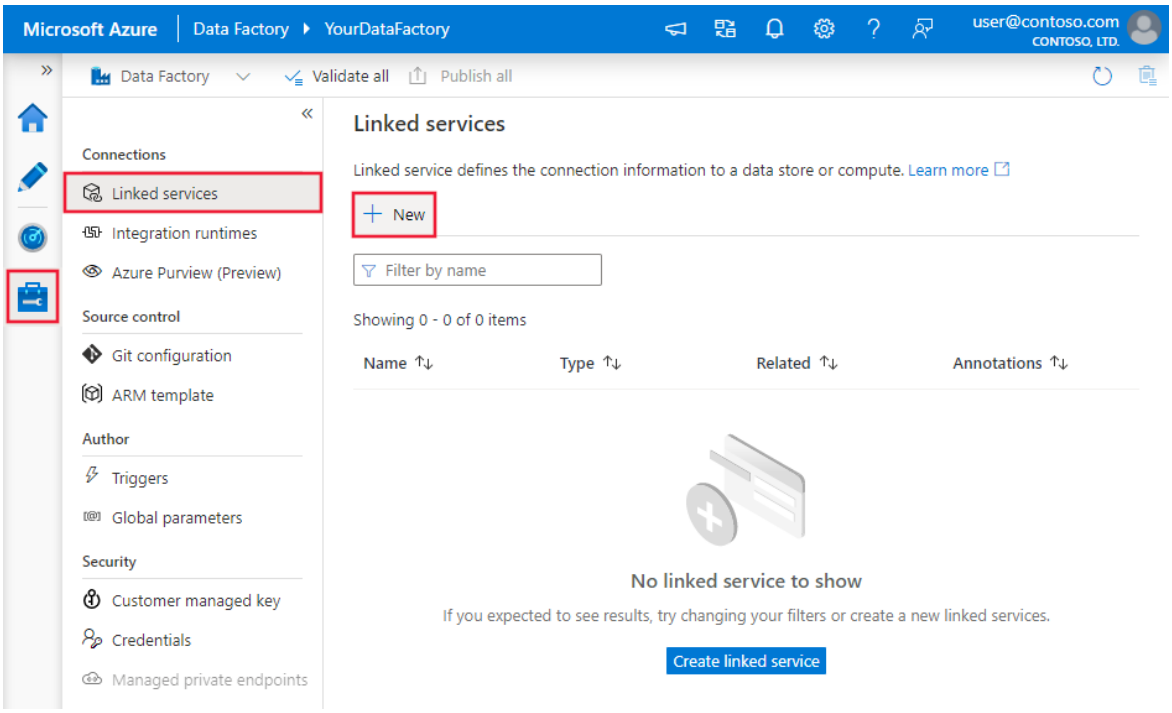
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Azure Data Lake Storage Gen1 using UI

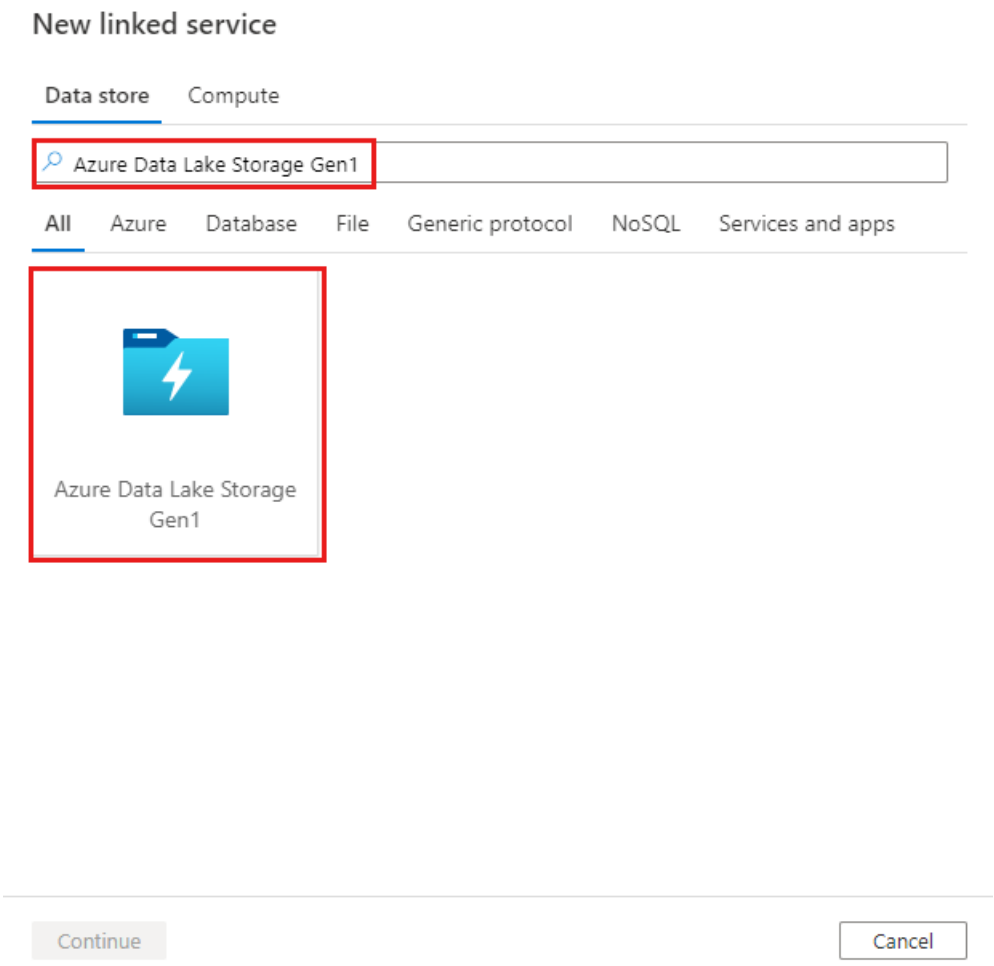
Use the following steps to create a linked service to Azure Data Lake Storage Gen1 in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

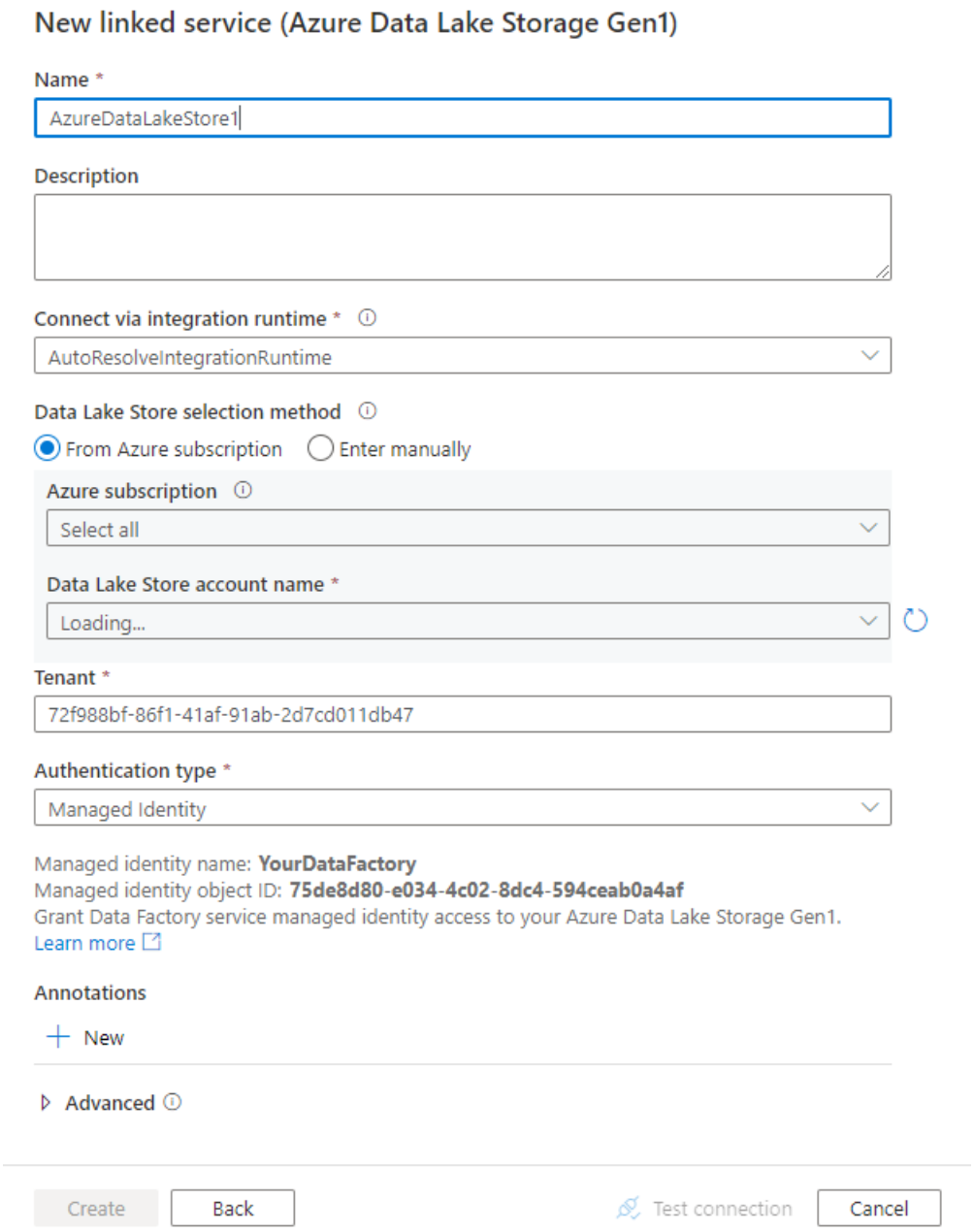
   - Azure Data Factory
   - Azure Synapse



2. Search for Azure Data Lake Storage Gen1 and select the Azure Data Lake Storage Gen1 connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide information about properties that are used to define entities specific to Azure Data Lake Store Gen1.

Linked service properties

The following properties are supported for the Azure Data Lake Store linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The `type` property must be set to **AzureDataLakeStore**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
dataLakeStoreUri	Information about the Azure Data Lake Store account. This information takes one of the following formats: `https://[accountname].azuredatalakestore.net/webhdfs/v1` or `adl://[accountname].azuredatalakestore.net/`.	Yes
subscriptionId	The Azure subscription ID to which the Data Lake Store account belongs.	Required for sink
resourceGroupName	The Azure resource group name to which the Data Lake Store account belongs.	Required for sink
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is located in a private network. If this property isn't specified, the default Azure integration runtime is used.	No

Use service principal authentication

To use service principal authentication, follow these steps.

1. Register an application entity in Azure Active Directory and grant it access to Data Lake Store. For detailed steps, see Service-to-service authentication. Make note of the following values, which you use to define the linked service:
   - Application ID
   - Application key
   - Tenant ID
2. Grant the service principal proper permission. See examples on how permission works in Data Lake Storage Gen1 from Access control in Azure Data Lake Storage Gen1.
   - **As source**: In **Data explorer** > **Access**, grant at least **Execute** permission for ALL upstream folders including the root, along with **Read** permission for the files to copy. You can choose to add to **This folder and all children** for recursive, and add as **an access permission and a default permission entry**. There's no requirement on account-level access control (IAM).
   - **As sink**: In **Data explorer** > **Access**, grant at least **Execute** permission for ALL upstream folders including the root, along with **Write** permission for the sink folder. You can choose to add to **This folder and all children** for recursive, and add as **an access permission and a default permission entry**.

The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalId	Specify the application's client ID.	Yes

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalKey	Specify the application's key. Mark this field as a `SecureString` to store it securely, or reference a secret stored in Azure Key Vault.	Yes
tenant	Specify the tenant information, such as domain name or tenant ID, under which your application resides. You can retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Yes
azureCloudType	For service principal authentication, specify the type of Azure cloud environment to which your Azure Active Directory application is registered. Allowed values are **AzurePublic**, **AzureChina**, **AzureUsGovernment**, and **AzureGermany**. By default, the service's cloud environment is used.	No

Example:

```
{
    "name": "AzureDataLakeStoreLinkedService",
    "properties": {
        "type": "AzureDataLakeStore",
        "typeProperties": {
            "dataLakeStoreUri": "https://<accountname>.azuredatalakestore.net/webhdfs/v1",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>",
            "subscriptionId": "<subscription of ADLS>",
            "resourceGroupName": "<resource group of ADLS>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Use system-assigned managed identity authentication

A data factory or Synapse workspace can be associated with a system-assigned managed identity, which represents the service for authentication. You can directly use this system-assigned managed identity for Data Lake Store authentication, similar to using your own service principal. It allows this designated resource to access and copy data to or from Data Lake Store.

To use system-assigned managed identity authentication, follow these steps.

1. Retrieve the system-assigned managed identity information by copying the value of the "Service Identity Application ID" generated along with your factory or Synapse workspace.
2. Grant the system-assigned managed identity access to Data Lake Store. See examples on how permission

works in Data Lake Storage Gen1 from Access control in Azure Data Lake Storage Gen1.

- **As source**: In **Data explorer** > **Access**, grant at least **Execute** permission for ALL upstream folders including the root, along with **Read** permission for the files to copy. You can choose to add to **This folder and all children** for recursive, and add as **an access permission and a default permission entry**. There's no requirement on account-level access control (IAM).
- **As sink**: In **Data explorer** > **Access**, grant at least **Execute** permission for ALL upstream folders including the root, along with **Write** permission for the sink folder. You can choose to add to **This folder and all children** for recursive, and add as **an access permission and a default permission entry**.

You don't need to specify any properties other than the general Data Lake Store information in the linked service.

Example:

```
{
    "name": "AzureDataLakeStoreLinkedService",
    "properties": {
        "type": "AzureDataLakeStore",
        "typeProperties": {
            "dataLakeStoreUri": "https://<accountname>.azuredatalakestore.net/webhdfs/v1",
            "subscriptionId": "<subscription of ADLS>",
            "resourceGroupName": "<resource group of ADLS>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Use user-assigned managed identity authentication

A data factory can be assigned with one or multiple user-assigned managed identities. You can use this user-assigned managed identity for Blob storage authentication, which allows to access and copy data from or to Data Lake Store. To learn more about managed identities for Azure resources, see Managed identities for Azure resources

To use user-assigned managed identity authentication, follow these steps:

1. Create one or multiple user-assigned managed identities and grant access to Azure Data Lake. See examples on how permission works in Data Lake Storage Gen1 from Access control in Azure Data Lake Storage Gen1.

   - **As source**: In **Data explorer** > **Access**, grant at least **Execute** permission for ALL upstream folders including the root, along with **Read** permission for the files to copy. You can choose to add to **This folder and all children** for recursive, and add as **an access permission and a default permission entry**. There's no requirement on account-level access control (IAM).
   - **As sink**: In **Data explorer** > **Access**, grant at least **Execute** permission for ALL upstream folders including the root, along with **Write** permission for the sink folder. You can choose to add to **This folder and all children** for recursive, and add as **an access permission and a default permission entry**.
2. Assign one or multiple user-assigned managed identities to your data factory and create credentials for each user-assigned managed identity.

The following property is supported:

PROPERTY	DESCRIPTION	REQUIRED
credentials	Specify the user-assigned managed identity as the credential object.	Yes

Example:

```
{
    "name": "AzureDataLakeStoreLinkedService",
    "properties": {
        "type": "AzureDataLakeStore",
        "typeProperties": {
            "dataLakeStoreUri": "https://<accountname>.azuredatalakestore.net/webhdfs/v1",
            "subscriptionId": "<subscription of ADLS>",
            "resourceGroupName": "<resource group of ADLS>",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Azure Data Lake Store Gen1 under `location` settings in the format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `location` in the dataset must be set to **AzureDataLakeStoreLocation**.	Yes
folderPath	The path to a folder. If you want to use a wildcard to filter folders, skip this setting and specify it in activity source settings.	No

PROPERTY	DESCRIPTION	REQUIRED
fileName	The file name under the given folderPath. If you want to use a wildcard to filter files, skip this setting and specify it in activity source settings.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<ADLS Gen1 linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AzureDataLakeStoreLocation",
                "folderPath": "root/folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see Pipelines. This section provides a list of properties supported by Azure Data Lake Store source and sink.

Azure Data Lake Store as source

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Azure Data Lake Store Gen1 under `storeSettings` settings in the format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `storeSettings` must be set to **AzureDataLakeStoreReadSettings**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
Locate the files to copy:		
OPTION 1: static path	Copy from the given folder/file path specified in the dataset. If you want to copy all files from a folder, additionally specify `wildcardFileName` as `*`.	
OPTION 2: name range - listAfter	Retrieve the folders/files whose name is after this value alphabetically (exclusive). It utilizes the service-side filter for ADLS Gen1, which provides better performance than a wildcard filter. The service applies this filter to the path defined in dataset, and only one entity level is supported. See more examples in Name range filter examples.	No
OPTION 2: name range - listBefore	Retrieve the folders/files whose name is before this value alphabetically (inclusive). It utilizes the service-side filter for ADLS Gen1, which provides better performance than a wildcard filter. The service applies this filter to the path defined in dataset, and only one entity level is supported. See more examples in Name range filter examples.	No
OPTION 3: wildcard - wildcardFolderPath	The folder path with wildcard characters to filter source folders. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual folder name has wildcard or this escape char inside. See more examples in Folder and file filter examples.	No
OPTION 3: wildcard - wildcardFileName	The file name with wildcard characters under the given folderPath/wildcardFolderPath to filter source files. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual file name has wildcard or this escape char inside. See more examples in Folder and file filter examples.	Yes

PROPERTY	DESCRIPTION	REQUIRED
OPTION 4: a list of files - fileListPath	Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset. When using this option, do not specify file name in dataset. See more examples in File list examples.	No
Additional settings:		
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when recursive is set to true and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**. This property doesn't apply when you configure `fileListPath`.	No
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	Files filter based on the attribute: Last Modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z". The properties can be NULL, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal with the datetime value will be selected. When `modifiedDatetimeEnd` has datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value will be selected. This property doesn't apply when you configure `fileListPath`.	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromADLSGen1",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "AzureDataLakeStoreReadSettings",
                    "recursive": true,
                    "wildcardFolderPath": "myfolder*A",
                    "wildcardFileName": "*.csv"
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Azure Data Lake Store as sink

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- JSON format
- ORC format
- Parquet format

The following properties are supported for Azure Data Lake Store Gen1 under `storeSettings` settings in the format-based copy sink:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `storeSettings` must be set to **AzureDataLakeStoreWriteSettings**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
copyBehavior	Defines the copy behavior when the source is files from a file-based data store. Allowed values are: - **PreserveHierarchy (default)**: Preserves the file hierarchy in the target folder. The relative path of the source file to the source folder is identical to the relative path of the target file to the target folder. - **FlattenHierarchy**: All files from the source folder are in the first level of the target folder. The target files have autogenerated names. - **MergeFiles**: Merges all files from the source folder to one file. If the file name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.	No
expiryDateTime	Specifies the expiry time of the written files. The time is applied to the UTC time in the format of "2020-03-01T08:00:00Z". By default it is NULL, which means the written files are never expired.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToADLSGen1",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Parquet output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "ParquetSink",
                "storeSettings":{
                    "type": "AzureDataLakeStoreWriteSettings",
                    "copyBehavior": "PreserveHierarchy"
                }
            }
        }
    }
]
```

Name range filter examples

This section describes the resulting behavior of name range filters.

SAMPLE SOURCE STRUCTURE	CONFIGURATION	RESULT

root
  a
  file.csv
  ax
  file2.csv
  ax.csv
  b
  file3.csv
  bx.csv
  c
  file4.csv
  cx.csv
	In dataset:
	- Folder path: `root`
	
	In copy activity source:
	- List after: `a`
	- List before: `b`
		Then the following files will be copied:
		
		root
		  ax
		  file2.csv
		  ax.csv
		  b
		  file3.csv

Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`Folder*`	(Empty, use default)	false	FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	(Empty, use default)	true	FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv
`Folder*`	`*.csv`	false	FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	`*.csv`	true	FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This section describes the resulting behavior of using file list path in copy activity source.

Assuming you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	CONFIGURATION
root FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In dataset:** - Folder path: `root/FolderA` **In copy activity source:** - File list path: `root/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line with the relative path to the path configured in the dataset.

Examples of behavior of the copy operation

This section describes the resulting behavior of the copy operation for different combinations of `recursive` and `copyBehavior` values.

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET
true	preserveHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the same structure as the source: Folder1 File1 File2 Subfolder1 File3 File4 File5.
true	flattenHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the following structure: Folder1 autogenerated name for File1 autogenerated name for File2 autogenerated name for File3 autogenerated name for File4 autogenerated name for File5
true	mergeFiles	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the following structure: Folder1 File1 + File2 + File3 + File4 + File5 contents are merged into one file, with an autogenerated file name.

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET

false	preserveHierarchy	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target Folder1 is created with the following structure:
			
			Folder1
			  File1
			  File2
			
			Subfolder1 with File3, File4, and File5 aren't picked up.

false	flattenHierarchy	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target Folder1 is created with the following structure:
			
			Folder1
			  autogenerated name for File1
			  autogenerated name for File2
			
			Subfolder1 with File3, File4, and File5 aren't picked up.

false	mergeFiles	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target Folder1 is created with the following structure:
			
			Folder1
			  File1 + File2 contents are merged into one file with autogenerated file name. autogenerated name for File1
			
			Subfolder1 with File3, File4, and File5 aren't picked up.

Preserve ACLs to Data Lake Storage Gen2

> **TIP**
> To copy data from Azure Data Lake Storage Gen1 into Gen2 in general, see Copy data from Azure Data Lake Storage Gen1 to Gen2 for a walk-through and best practices.

If you want to replicate the access control lists (ACLs) along with data files when you upgrade from Data Lake Storage Gen1 to Data Lake Storage Gen2, see Preserve ACLs from Data Lake Storage Gen1.

Mapping data flow properties

When you're transforming data in mapping data flows, you can read and write files from Azure Data Lake Storage Gen1 in the following formats:

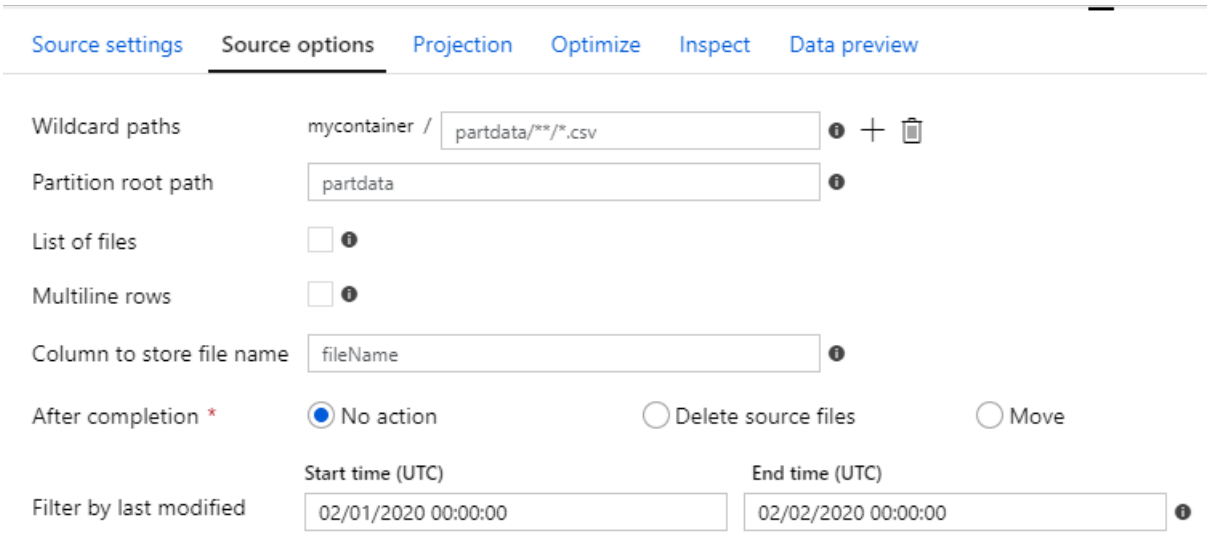
- Avro
- Delimited text

- Excel
- JSON
- ORC format
- Parquet

Format-specific settings are located in the documentation for that format. For more information, see Source transformation in mapping data flow and Sink transformation in mapping data flow.

Source transformation

In the source transformation, you can read from a container, folder, or individual file in Azure Data Lake Storage Gen1. The **Source options** tab lets you manage how the files get read.



Wildcard path: Using a wildcard pattern will instruct the service to loop through each matching folder and file in a single Source transformation. This is an effective way to process multiple files within a single flow. Add multiple wildcard matching patterns with the + sign that appears when hovering over your existing wildcard pattern.

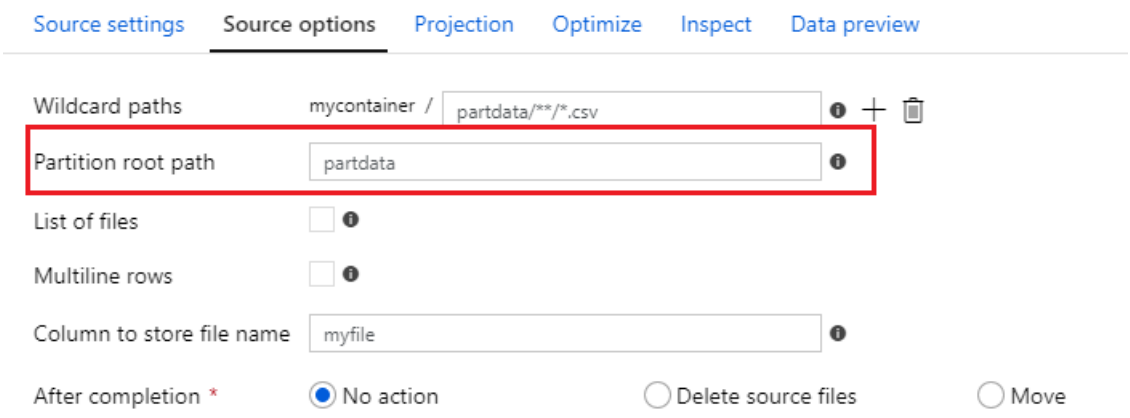
From your source container, choose a series of files that match a pattern. Only container can be specified in the dataset. Your wildcard path must therefore also include your folder path from the root folder.

Wildcard examples:

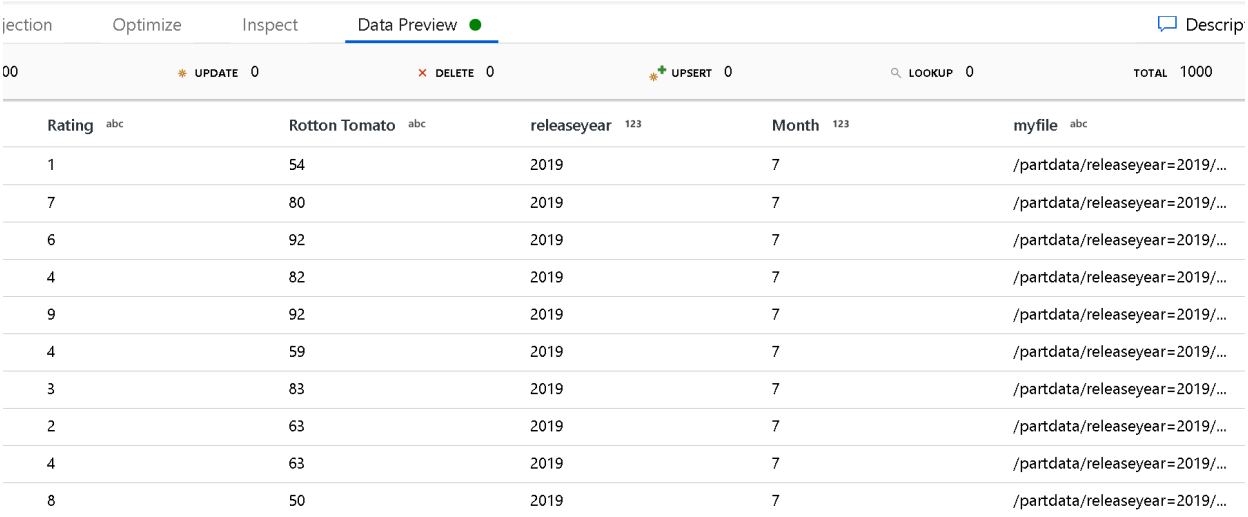
- `*` Represents any set of characters
- `**` Represents recursive directory nesting
- `?` Replaces one character
- `[]` Matches one of more characters in the brackets
- `/data/sales/**/*.csv` Gets all csv files under /data/sales
- `/data/sales/20??/**/` Gets all files in the 20th century
- `/data/sales/*/*/*.csv` Gets csv files two levels under /data/sales
- `/data/sales/2004/*/12/[XY]1?.csv` Gets all csv files in 2004 in December starting with X or Y prefixed by a two-digit number

Partition Root Path: If you have partitioned folders in your file source with a `key=value` format (for example, year=2019), then you can assign the top level of that partition folder tree to a column name in your data flow data stream.

First, set a wildcard to include all paths that are the partitioned folders plus the leaf files that you wish to read.



Use the Partition Root Path setting to define what the top level of the folder structure is. When you view the contents of your data via a data preview, you'll see that the service will add the resolved partitions found in each of your folder levels.



Rating abc	Rotton Tomato abc	releaseyear 123	Month 123	myfile abc
1	54	2019	7	/partdata/releaseyear=2019/...
7	80	2019	7	/partdata/releaseyear=2019/...
6	92	2019	7	/partdata/releaseyear=2019/...
4	82	2019	7	/partdata/releaseyear=2019/...
9	92	2019	7	/partdata/releaseyear=2019/...
4	59	2019	7	/partdata/releaseyear=2019/...
3	83	2019	7	/partdata/releaseyear=2019/...
2	63	2019	7	/partdata/releaseyear=2019/...
4	63	2019	7	/partdata/releaseyear=2019/...
8	50	2019	7	/partdata/releaseyear=2019/...

List of files: This is a file set. Create a text file that includes a list of relative path files to process. Point to this text file.

Column to store file name: Store the name of the source file in a column in your data. Enter a new column name here to store the file name string.

After completion: Choose to do nothing with the source file after the data flow runs, delete the source file, or move the source file. The paths for the move are relative.

To move source files to another location post-processing, first select "Move" for file operation. Then, set the "from" directory. If you're not using any wildcards for your path, then the "from" setting will be the same folder as your source folder.

If you have a source path with wildcard, your syntax will look like this below:

```
/data/sales/20??/**/*.csv
```

You can specify "from" as

```
/data/sales
```

And "to" as

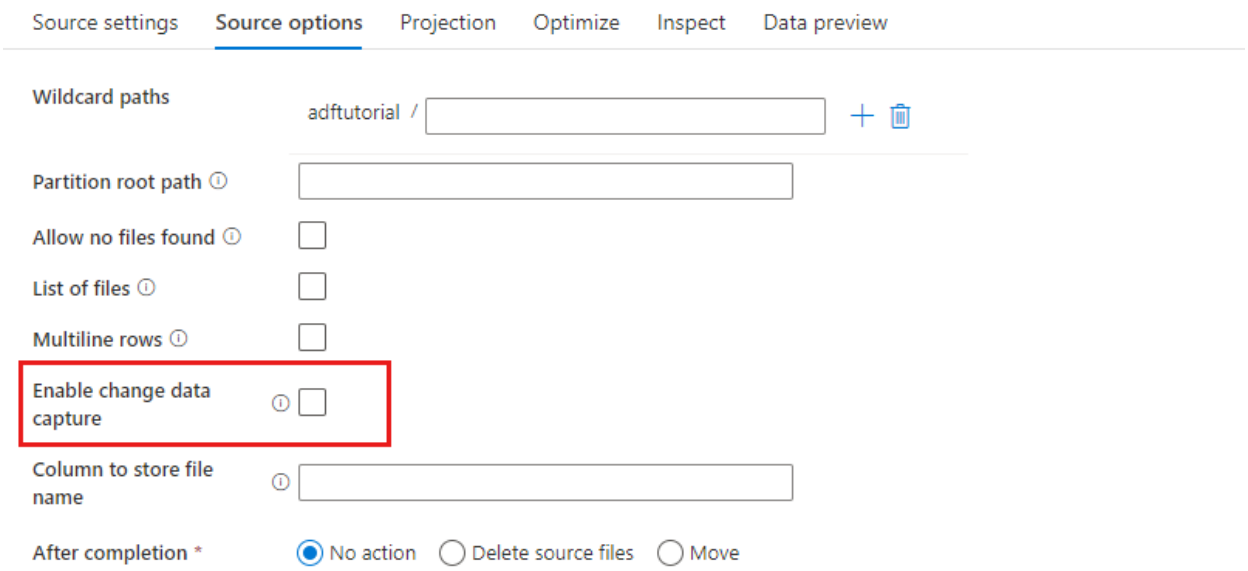
```
/backup/priorSales
```

In this case, all files that were sourced under /data/sales are moved to /backup/priorSales.

> **NOTE**
> File operations run only when you start the data flow from a pipeline run (a pipeline debug or execution run) that uses the Execute Data Flow activity in a pipeline. File operations *do not* run in Data Flow debug mode.

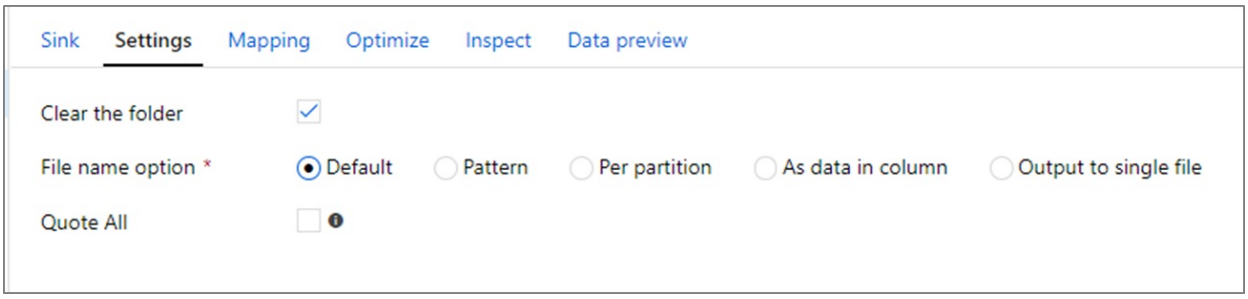
Filter by last modified: You can filter which files you process by specifying a date range of when they were last modified. All date-times are in UTC.

Enable change data capture: If true, you will get new or changed files only from the last run. Initial load of full snapshot data will always be gotten in the first run, followed by capturing new or changed files only in next runs. For more details, see Change data capture.



Sink properties

In the sink transformation, you can write to either a container or folder in Azure Data Lake Storage Gen1. the **Settings** tab lets you manage how the files get written.



Clear the folder: Determines whether or not the destination folder gets cleared before the data is written.

File name option: Determines how the destination files are named in the destination folder. The file name options are:

- **Default**: Allow Spark to name files based on PART defaults.
- **Pattern**: Enter a pattern that enumerates your output files per partition. For example, **loans[n].csv** will create loans1.csv, loans2.csv, and so on.
- **Per partition**: Enter one file name per partition.
- **As data in column**: Set the output file to the value of a column. The path is relative to the dataset container, not the destination folder. If you have a folder path in your dataset, it will be overridden.
- **Output to a single file**: Combine the partitioned output files into a single named file. The path is relative to the dataset folder. Please be aware that the merge operation can possibly fail based upon node size. This

option is not recommended for large datasets.

Quote all: Determines whether to enclose all values in quotes

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity

Delete activity properties

To learn details about the properties, check Delete activity

Legacy models

> **NOTE**
> The following models are still supported as-is for backward compatibility. You are suggested to use the new model mentioned in above sections going forward, and the authoring UI has switched to generating the new model.

Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **AzureDataLakeStoreFile**.	Yes
folderPath	Path to the folder in Data Lake Store. If not specified, it points to the root. Wildcard filter is supported. Allowed wildcards are `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your actual folder name has a wildcard or this escape char inside. For example: rootfolder/subfolder/. See more examples in Folder and file filter examples.	No

PROPERTY	DESCRIPTION	REQUIRED
fileName	Name or wildcard filter for the files under the specified "folderPath". If you don't specify a value for this property, the dataset points to all files in the folder. For filter, the wildcards allowed are `*` (matches zero or more characters) and `?` (matches zero or single character). - Example 1: `"fileName": "*.csv"` - Example 2: `"fileName": "???20180427.txt"` Use `^` to escape if your actual file name has a wildcard or this escape char inside. When fileName isn't specified for an output dataset and **preserveHierarchy** isn't specified in the activity sink, the copy activity automatically generates the file name with the following pattern: "*Data.[activity run ID GUID].[GUID if FlattenHierarchy].[format if configured].[compression if configured]*", for example, "Data.0a405f8a-93ff-4c6f-b3be-f69616f1df7a.txt.gz". If you copy from a tabular source by using a table name instead of a query, the name pattern is "*[table name].[format].[compression if configured]*", for example, "MyTable.csv".	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	Files filter based on the attribute Last Modified. The files are selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z". The overall performance of data movement is affected by enabling this setting when you want to do file filter with huge amounts of files. The properties can be NULL, which means no file attribute filter is applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal to the datetime value are selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value are selected.	No
modifiedDatetimeEnd	Files filter based on the attribute Last Modified. The files are selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z". The overall performance of data movement is affected by enabling this setting when you want to do file filter with huge amounts of files. The properties can be NULL, which means no file attribute filter is applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal to the datetime value are selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value are selected.	No

PROPERTY	DESCRIPTION	REQUIRED
format	If you want to copy files as is between file-based stores (binary copy), skip the format section in both input and output dataset definitions. If you want to parse or generate files with a specific format, the following file format types are supported: **TextFormat**, **JsonFormat**, **AvroFormat**, **OrcFormat**, and **ParquetFormat**. Set the **type** property under **format** to one of these values. For more information, see the Text format, JSON format, Avro format, Orc format, and Parquet format sections.	No (only for binary copy scenario)
compression	Specify the type and level of compression for the data. For more information, see Supported file formats and compression codecs. Supported types are **GZip**, **Deflate**, **BZip2**, and **ZipDeflate**. Supported levels are **Optimal** and **Fastest**.	No

TIP

To copy all files under a folder, specify **folderPath** only.
To copy a single file with a particular name, specify **folderPath** with a folder part and **fileName** with a file name.
To copy a subset of files under a folder, specify **folderPath** with a folder part and **fileName** with a wildcard filter.

Example:

```
{
    "name": "ADLSDataset",
    "properties": {
        "type": "AzureDataLakeStoreFile",
        "linkedServiceName":{
            "referenceName": "<ADLS linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "folderPath": "datalake/myfolder/",
            "fileName": "*",
            "modifiedDatetimeStart": "2018-12-01T05:00:00Z",
            "modifiedDatetimeEnd": "2018-12-01T06:00:00Z",
            "format": {
                "type": "TextFormat",
                "columnDelimiter": ",",
                "rowDelimiter": "\n"
            },
            "compression": {
                "type": "GZip",
                "level": "Optimal"
            }
        }
    }
}
```

Legacy copy activity source model

PROPERTY	DESCRIPTION	REQUIRED
type	The `type` property of the copy activity source must be set to **AzureDataLakeStoreSource**.	Yes
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. When `recursive` is set to true and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromADLSGen1",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<ADLS Gen1 input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AzureDataLakeStoreSource",
                "recursive": true
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Legacy copy activity sink model

PROPERTY	DESCRIPTION	REQUIRED
type	The `type` property of the copy activity sink must be set to **AzureDataLakeStoreSink**.	Yes
copyBehavior	Defines the copy behavior when the source is files from a file-based data store. Allowed values are: - **PreserveHierarchy (default)**: Preserves the file hierarchy in the target folder. The relative path of the source file to the source folder is identical to the relative path of the target file to the target folder. - **FlattenHierarchy**: All files from the source folder are in the first level of the target folder. The target files have autogenerated names. - **MergeFiles**: Merges all files from the source folder to one file. If the file name is specified, the merged file name is the specified name. Otherwise, the file name is autogenerated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToADLSGen1",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<ADLS Gen1 output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureDataLakeStoreSink",
                "copyBehavior": "PreserveHierarchy"
            }
        }
    }
]
```

Change data capture (preview)

Azure Data Factory can get new or changed files only from Azure Data Lake Storage Gen1 by enabling **Enable change data capture (Preview)** in the mapping data flow source transformation. With this connector option, you can read new or updated files only and apply transformations before loading transformed data into destination datasets of your choice.

Make sure you keep the pipeline and activity name unchanged, so that the checkpoint can always be recorded from the last run to get changes from there. If you change your pipeline name or activity name, the checkpoint will be reset, and you will start from the beginning in the next run.

When you debug the pipeline, the **Enable change data capture (Preview)** works as well. Be aware that the checkpoint will be reset when you refresh your browser during the debug run. After you are satisfied with the result from debug run, you can publish and trigger the pipeline. It will always start from the beginning regardless of the previous checkpoint recorded by debug run.

In the monitoring section, you always have the chance to rerun a pipeline. When you are doing so, the changes are always gotten from the checkpoint record in your selected pipeline run.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy and transform data in Azure Data Lake Storage Gen2 using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 33 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Azure Data Lake Storage Gen2 (ADLS Gen2) is a set of capabilities dedicated to big data analytics built into Azure Blob storage. You can use it to interface with your data by using both file system and object storage paradigms.

This article outlines how to use Copy Activity to copy data from and to Azure Data Lake Storage Gen2, and use Data Flow to transform data in Azure Data Lake Storage Gen2. To learn more, read the introductory article for Azure Data Factory or Azure Synapse Analytics.

> **TIP**
> For data lake or data warehouse migration scenario, learn more in Migrate data from your data lake or data warehouse to Azure.

Supported capabilities

This Azure Data Lake Storage Gen2 connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/sink)	① ②	✓
Mapping data flow (source/sink)	①	✓
Lookup activity	① ②	✓
GetMetadata activity	① ②	✓
Delete activity	① ②	✓

① Azure integration runtime ② Self-hosted integration runtime

For Copy activity, with this connector you can:

- Copy data from/to Azure Data Lake Storage Gen2 by using account key, service principal, or managed identities for Azure resources authentications.
- Copy files as-is or parse or generate files with supported file formats and compression codecs.
- Preserve file metadata during copy.
- Preserve ACLs when copying from Azure Data Lake Storage Gen1/Gen2.

Get started

TIP

For a walk-through of how to use the Data Lake Storage Gen2 connector, see Load data into Azure Data Lake Storage Gen2.

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

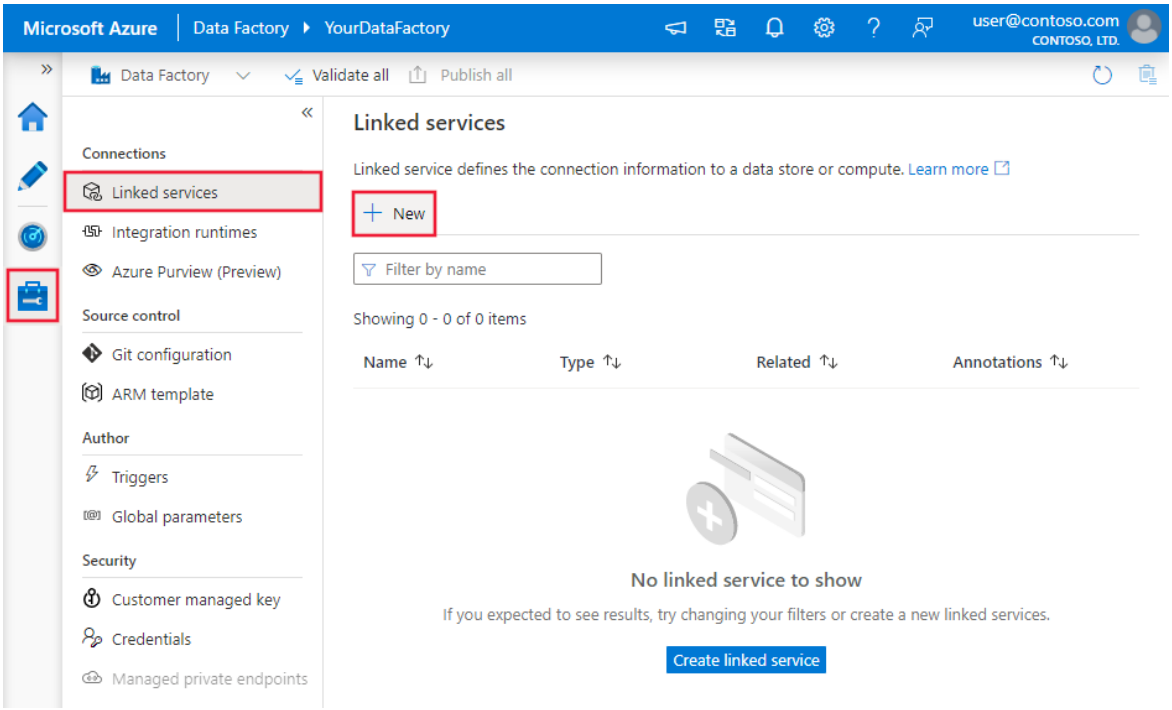
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create an Azure Data Lake Storage Gen2 linked service using UI

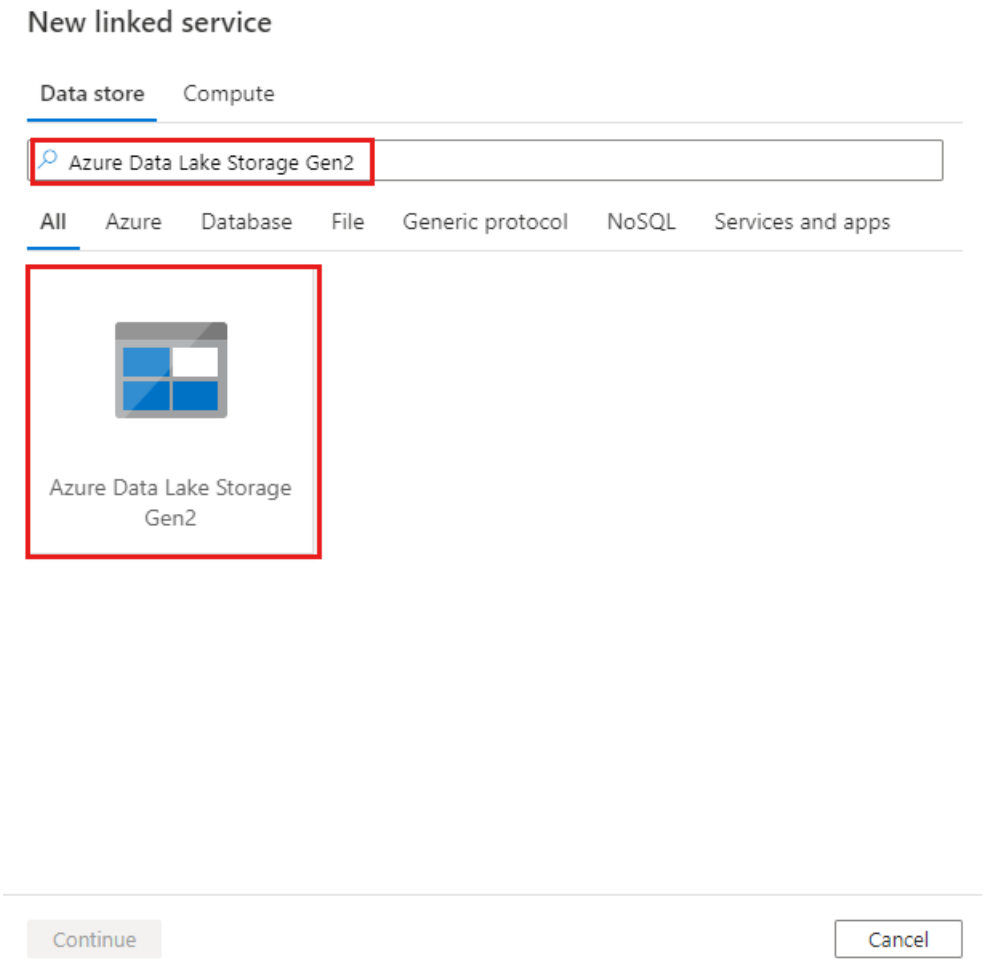
Use the following steps to create an Azure Data Lake Storage Gen2 linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

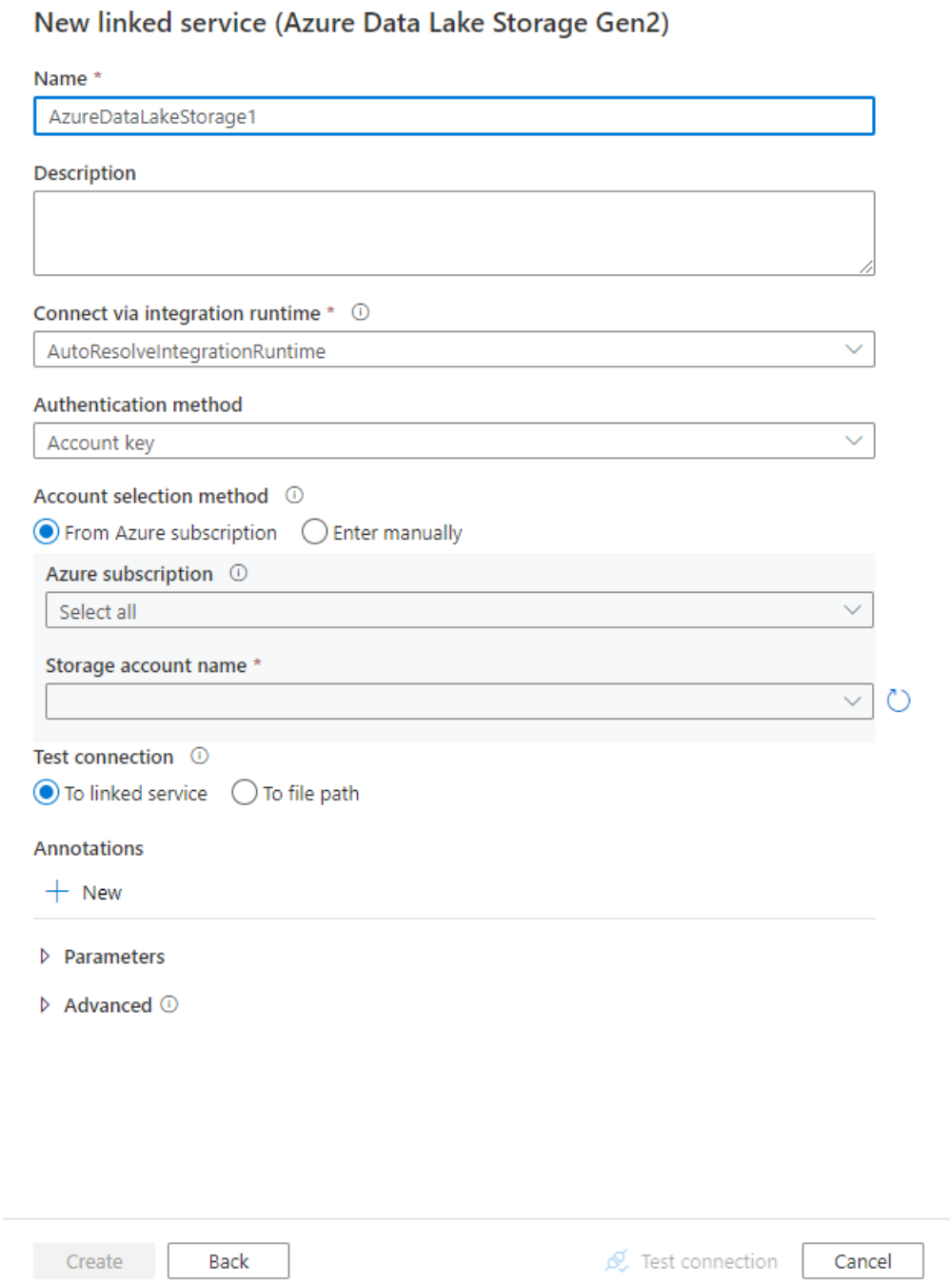
   - Azure Data Factory
   - Azure Synapse



2. Search for Azure Data Lake Storage Gen2 and select the Azure Data Lake Storage Gen2 connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide information about properties that are used to define Data Factory and Synapse pipeline entities specific to Data Lake Storage Gen2.

Linked service properties

The Azure Data Lake Storage Gen2 connector supports the following authentication types. See the corresponding sections for details:

- Account key authentication
- Service principal authentication
- System-assigned managed identity authentication
- User-assigned managed identity authentication

NOTE

- If want to use the public Azure integration runtime to connect to the Data Lake Storage Gen2 by leveraging the **Allow trusted Microsoft services to access this storage account** option enabled on Azure Storage firewall, you must use managed identity authentication. For more information about the Azure Storage firewalls settings, see Configure Azure Storage firewalls and virtual networks.
- When you use PolyBase or COPY statement to load data into Azure Synapse Analytics, if your source or staging Data Lake Storage Gen2 is configured with an Azure Virtual Network endpoint, you must use managed identity authentication as required by Azure Synapse. See the managed identity authentication section with more configuration prerequisites.

Account key authentication

To use storage account key authentication, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AzureBlobFS**.	Yes
url	Endpoint for Data Lake Storage Gen2 with the pattern of `https://<accountname>.dfs.core.windows.net`.	Yes
accountKey	Account key for Data Lake Storage Gen2. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If this property isn't specified, the default Azure integration runtime is used.	No

NOTE

Secondary ADLS file system endpoint is not supported when using account key authentication. You can use other authentication types.

Example:

```
{
    "name": "AzureDataLakeStorageGen2LinkedService",
    "properties": {
        "type": "AzureBlobFS",
        "typeProperties": {
            "url": "https://<accountname>.dfs.core.windows.net",
            "accountkey": {
                "type": "SecureString",
                "value": "<accountkey>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Service principal authentication

To use service principal authentication, follow these steps.

1. Register an application entity in Azure Active Directory (Azure AD) by following the steps in Register your application with an Azure AD tenant. Make note of the following values, which you use to define the linked service:
   - Application ID
   - Application key
   - Tenant ID
2. Grant the service principal proper permission. See examples on how permission works in Data Lake Storage Gen2 from Access control lists on files and directories
   - **As source**: In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Read** permission for the files to copy. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Reader** role.
   - **As sink**: In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Write** permission for the sink folder. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Contributor** role.

> **NOTE**
> If you use UI to author and the service principal is not set with "Storage Blob Data Reader/Contributor" role in IAM, when doing test connection or browsing/navigating folders, choose "Test connection to file path" or "Browse from specified path", and specify a path with **Read + Execute** permission to continue.

These properties are supported for the linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AzureBlobFS**.	Yes
url	Endpoint for Data Lake Storage Gen2 with the pattern of `https://<accountname>.dfs.core.windows.net`.	Yes

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalCredentialType	The credential type to use for service principal authentication. Allowed values are **ServicePrincipalKey** and **ServicePrincipalCert**.	Yes
servicePrincipalCredential	The service principal credential. When you use **ServicePrincipalKey** as the credential type, specify the application's key. Mark this field as **SecureString** to store it securely, or reference a secret stored in Azure Key Vault. When you use **ServicePrincipalCert** as the credential, reference a certificate in Azure Key Vault, and ensure the certificate content type is **PKCS #12**.	Yes
servicePrincipalKey	Specify the application's key. Mark this field as **SecureString** to store it securely, or reference a secret stored in Azure Key Vault. This property is still supported as-is for `servicePrincipalId` + `servicePrincipalKey`. As ADF adds new service principal certificate authentication, the new model for service principal authentication is `servicePrincipalId` + `servicePrincipalCredentialType` + `servicePrincipalCredential`.	No
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. Retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Yes
azureCloudType	For service principal authentication, specify the type of Azure cloud environment to which your Azure Active Directory application is registered. Allowed values are **AzurePublic**, **AzureChina**, **AzureUsGovernment**, and **AzureGermany**. By default, the data factory or Synapse pipeline's cloud environment is used.	No
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

Example: using service principal key authentication

You can also store service principal key in Azure Key Vault.

```
{
    "name": "AzureDataLakeStorageGen2LinkedService",
    "properties": {
        "type": "AzureBlobFS",
        "typeProperties": {
            "url": "https://<accountname>.dfs.core.windows.net",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalCredentialType": "ServicePrincipalKey",
            "servicePrincipalCredential": {
                "type": "SecureString",
                "value": "<service principal key>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: using service principal certificate authentication

```
{
    "name": "AzureDataLakeStorageGen2LinkedService",
    "properties": {
        "type": "AzureBlobFS",
        "typeProperties": {
            "url": "https://<accountname>.dfs.core.windows.net",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalCredentialType": "ServicePrincipalCert",
            "servicePrincipalCredential": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<AKV reference>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<certificate name in AKV>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

System-assigned managed identity authentication

A data factory or Synapse workspace can be associated with a system-assigned managed identity. You can directly use this system-assigned managed identity for Data Lake Storage Gen2 authentication, similar to using your own service principal. It allows this designated factory or workspace to access and copy data to or from your Data Lake Storage Gen2.

To use system-assigned managed identity authentication, follow these steps.

1. Retrieve the system-assigned managed identity information by copying the value of the **managed identity object ID** generated along with your data factory or Synapse workspace.

2. Grant the system-assigned managed identity proper permission. See examples on how permission works in Data Lake Storage Gen2 from Access control lists on files and directories.

   - **As source**: In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Read** permission for the files to copy. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Reader** role.
   - **As sink**: In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Write** permission for the sink folder. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Contributor** role.

These properties are supported for the linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AzureBlobFS**.	Yes
url	Endpoint for Data Lake Storage Gen2 with the pattern of `https://<accountname>.dfs.core.windows.net`.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

Example:

```
{
    "name": "AzureDataLakeStorageGen2LinkedService",
    "properties": {
        "type": "AzureBlobFS",
        "typeProperties": {
            "url": "https://<accountname>.dfs.core.windows.net",
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

User-assigned managed identity authentication

A data factory can be assigned with one or multiple user-assigned managed identities. You can use this user-assigned managed identity for Blob storage authentication, which allows to access and copy data from or to Data Lake Storage Gen2. To learn more about managed identities for Azure resources, see Managed identities for Azure resources

To use user-assigned managed identity authentication, follow these steps:

1. Create one or multiple user-assigned managed identities and grant access to Azure Data Lake Storage Gen2. See examples on how permission works in Data Lake Storage Gen2 from Access control lists on files and directories.

- **As source**: In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Read** permission for the files to copy. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Reader** role.
- **As sink**: In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Write** permission for the sink folder. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Contributor** role.

2. Assign one or multiple user-assigned managed identities to your data factory and create credentials for each user-assigned managed identity.

These properties are supported for the linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AzureBlobFS**.	Yes
url	Endpoint for Data Lake Storage Gen2 with the pattern of `https://<accountname>.dfs.core.windows.net`.	Yes
credentials	Specify the user-assigned managed identity as the credential object.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

Example:

```
{
    "name": "AzureDataLakeStorageGen2LinkedService",
    "properties": {
        "type": "AzureBlobFS",
        "typeProperties": {
            "url": "https://<accountname>.dfs.core.windows.net",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
                }
            },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

NOTE

If you use Data Factory UI to author and the managed identity is not set with "Storage Blob Data Reader/Contributor" role in IAM, when doing test connection or browsing/navigating folders, choose "Test connection to file path" or "Browse from specified path", and specify a path with **Read + Execute** permission to continue.

IMPORTANT

If you use PolyBase or COPY statement to load data from Data Lake Storage Gen2 into Azure Synapse Analytics, when you use managed identity authentication for Data Lake Storage Gen2, make sure you also follow steps 1 to 3 in this guidance. Those steps will register your server with Azure AD and assign the Storage Blob Data Contributor role to your server. Data Factory handles the rest. If you configure Blob storage with an Azure Virtual Network endpoint, you also need to have **Allow trusted Microsoft services to access this storage account** turned on under Azure Storage account **Firewalls and Virtual networks** settings menu as required by Azure Synapse.

Dataset properties

For a full list of sections and properties available for defining datasets, see Datasets.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Data Lake Storage Gen2 under `location` settings in the format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `location` in the dataset must be set to **AzureBlobFSLocation**.	Yes
fileSystem	The Data Lake Storage Gen2 file system name.	No
folderPath	The path to a folder under the given file system. If you want to use a wildcard to filter folders, skip this setting and specify it in activity source settings.	No
fileName	The file name under the given fileSystem + folderPath. If you want to use a wildcard to filter files, skip this setting and specify it in activity source settings.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<Data Lake Storage Gen2 linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AzureBlobFSLocation",
                "fileSystem": "filesystemname",
                "folderPath": "folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see Copy activity configurations and Pipelines and activities. This section provides a list of properties supported by the Data Lake Storage Gen2 source and sink.

Azure Data Lake Storage Gen2 as a source type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

You have several options to copy data from ADLS Gen2:

- Copy from the given path specified in the dataset.
- Wildcard filter against folder path or file name, see `wildcardFolderPath` and `wildcardFileName`.
- Copy the files defined in a given text file as file set, see `fileListPath`.

The following properties are supported for Data Lake Storage Gen2 under `storeSettings` settings in format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `storeSettings` must be set to **AzureBlobFSReadSettings**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
Locate the files to copy:		
OPTION 1: static path	Copy from the given file system or folder/file path specified in the dataset. If you want to copy all files from a file system/folder, additionally specify `wildcardFileName` as `*`.	
OPTION 2: wildcard - wildcardFolderPath	The folder path with wildcard characters under the given file system configured in dataset to filter source folders. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual folder name has wildcard or this escape char inside. See more examples in Folder and file filter examples.	No
OPTION 2: wildcard - wildcardFileName	The file name with wildcard characters under the given file system + folderPath/wildcardFolderPath to filter source files. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual file name has wildcard or this escape char inside. See more examples in Folder and file filter examples.	Yes
OPTION 3: a list of files - fileListPath	Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset. When using this option, do not specify file name in dataset. See more examples in File list examples.	No
Additional settings:		
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when recursive is set to true and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**. This property doesn't apply when you configure `fileListPath`.	No

PROPERTY	DESCRIPTION	REQUIRED
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No
modifiedDatetimeStart	Files filter based on the attribute: Last Modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z". The properties can be NULL, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal with the datetime value will be selected. When `modifiedDatetimeEnd` has datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value will be selected. This property doesn't apply when you configure `fileListPath`.	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromADLSGen2",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "AzureBlobFSReadSettings",
                    "recursive": true,
                    "wildcardFolderPath": "myfolder*A",
                    "wildcardFileName": "*.csv"
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Azure Data Lake Storage Gen2 as a sink type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- JSON format
- ORC format
- Parquet format

The following properties are supported for Data Lake Storage Gen2 under `storeSettings` settings in format-based copy sink:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `storeSettings` must be set to **AzureBlobFSWriteSettings**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
copyBehavior	Defines the copy behavior when the source is files from a file-based data store. Allowed values are: - **PreserveHierarchy (default)**: Preserves the file hierarchy in the target folder. The relative path of the source file to the source folder is identical to the relative path of the target file to the target folder. - **FlattenHierarchy**: All files from the source folder are in the first level of the target folder. The target files have autogenerated names. - **MergeFiles**: Merges all files from the source folder to one file. If the file name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.	No
blockSizeInMB	Specify the block size in MB used to write data to ADLS Gen2. Learn more about Block Blobs. Allowed value is **between 4 MB and 100 MB**. By default, ADF automatically determines the block size based on your source store type and data. For non-binary copy into ADLS Gen2, the default block size is 100 MB so as to fit in at most approximately 4.75-TB data. It may be not optimal when your data is not large, especially when you use Self-hosted Integration Runtime with poor network resulting in operation timeout or performance issue. You can explicitly specify a block size, while ensure blockSizeInMB*50000 is big enough to store the data, otherwise copy activity run will fail.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

PROPERTY	DESCRIPTION	REQUIRED
metadata	Set custom metadata when copy to sink. Each object under the `metadata` array represents an extra column. The `name` defines the metadata key name, and the `value` indicates the data value of that key. If preserve attributes feature is used, the specified metadata will union/overwrite with the source file metadata. Allowed data values are: - `$$LASTMODIFIED`: a reserved variable indicates to store the source files' last modified time. Apply to file-based source with binary format only. - **Expression** - **Static value**	No

Example:

```
"activities":[
    {
        "name": "CopyToADLSGen2",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Parquet output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "ParquetSink",
                "storeSettings":{
                    "type": "AzureBlobFSWriteSettings",
                    "copyBehavior": "PreserveHierarchy",
                    "metadata": [
                        {
                            "name": "testKey1",
                            "value": "value1"
                        },
                        {
                            "name": "testKey2",
                            "value": "value2"
                        },
                        {
                            "name": "lastModifiedKey",
                            "value": "$$LASTMODIFIED"
                        }
                    ]
                }
            }
        }
    }
]
```

Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`Folder*`	(Empty, use default)	false	FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`Folder*`	(Empty, use default)	true	FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv
`Folder*`	`*.csv`	false	FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	`*.csv`	true	FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This section describes the resulting behavior of using file list path in copy activity source.

Assuming you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	ADF CONFIGURATION
filesystem FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In dataset:** - File system: `filesystem` - Folder path: `FolderA` **In copy activity source:** - File list path: `filesystem/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line with the relative path to the path configured in the dataset.

Some recursive and copyBehavior examples

This section describes the resulting behavior of the copy operation for different combinations of recursive and copyBehavior values.

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET

true	preserveHierarchy	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target Folder1 is created with the same structure as the source:
			
			Folder1
			  File1
			  File2
			  Subfolder1
			  File3
			  File4
			  File5

true	flattenHierarchy	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target Folder1 is created with the following structure:
			
			Folder1
			  autogenerated name for File1
			  autogenerated name for File2
			  autogenerated name for File3
			  autogenerated name for File4
			  autogenerated name for File5

true	mergeFiles	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target Folder1 is created with the following structure:
			
			Folder1
			  File1 + File2 + File3 + File4 + File5 contents are merged into one file with an autogenerated file name.

false	preserveHierarchy	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target Folder1 is created with the following structure:
			
			Folder1
			  File1
			  File2
			
			Subfolder1 with File3, File4, and File5 isn't picked up.

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET

false	flattenHierarchy	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target Folder1 is created with the following structure:
			
			Folder1
			  autogenerated name for File1
			  autogenerated name for File2
			
			Subfolder1 with File3, File4, and File5 isn't picked up.

false	mergeFiles	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target Folder1 is created with the following structure:
			
			Folder1
			  File1 + File2 contents are merged into one file with an autogenerated file name. autogenerated name for File1
			
			Subfolder1 with File3, File4, and File5 isn't picked up.

Preserve metadata during copy

When you copy files from Amazon S3/Azure Blob/Azure Data Lake Storage Gen2 to Azure Data Lake Storage Gen2/Azure Blob, you can choose to preserve the file metadata along with data. Learn more from Preserve metadata.

Preserve ACLs from Data Lake Storage Gen1/Gen2

When you copy files from Azure Data Lake Storage Gen1/Gen2 to Gen2, you can choose to preserve the POSIX access control lists (ACLs) along with data. Learn more from Preserve ACLs from Data Lake Storage Gen1/Gen2 to Gen2.

> **TIP**
> To copy data from Azure Data Lake Storage Gen1 into Gen2 in general, see Copy data from Azure Data Lake Storage Gen1 to Gen2 for a walk-through and best practices.

Mapping data flow properties

When you're transforming data in mapping data flows, you can read and write files from Azure Data Lake Storage Gen2 in the following formats:

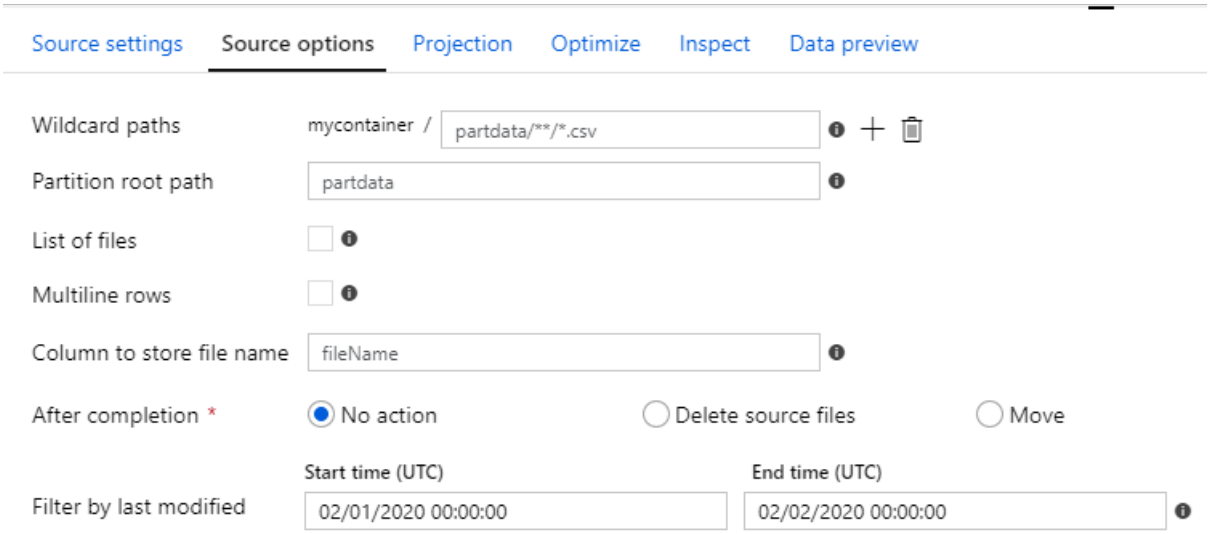
- Avro
- Common Data Model
- Delimited text
- Delta

- Excel
- JSON
- Parquet

Format specific settings are located in the documentation for that format. For more information, see Source transformation in mapping data flow and Sink transformation in mapping data flow.

Source transformation

In the source transformation, you can read from a container, folder, or individual file in Azure Data Lake Storage Gen2. The **Source options** tab lets you manage how the files get read.



Wildcard path: Using a wildcard pattern will instruct ADF to loop through each matching folder and file in a single Source transformation. This is an effective way to process multiple files within a single flow. Add multiple wildcard matching patterns with the + sign that appears when hovering over your existing wildcard pattern.

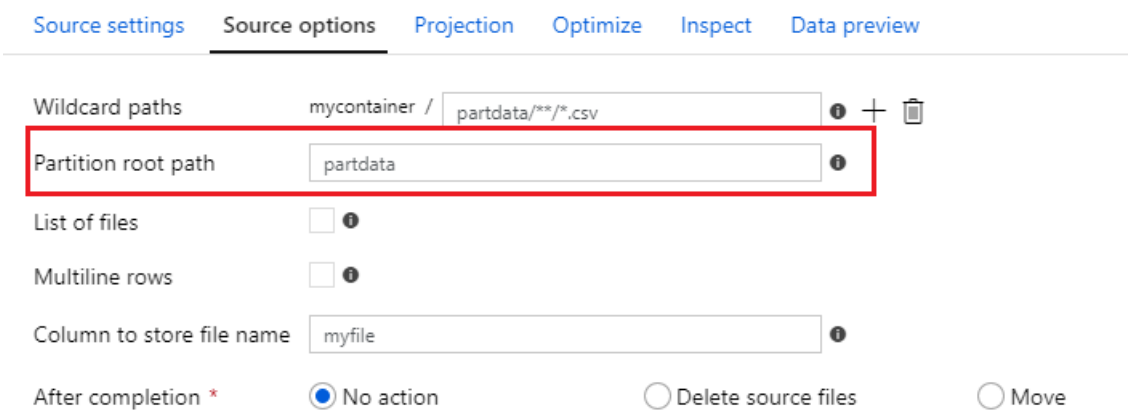
From your source container, choose a series of files that match a pattern. Only container can be specified in the dataset. Your wildcard path must therefore also include your folder path from the root folder.

Wildcard examples:

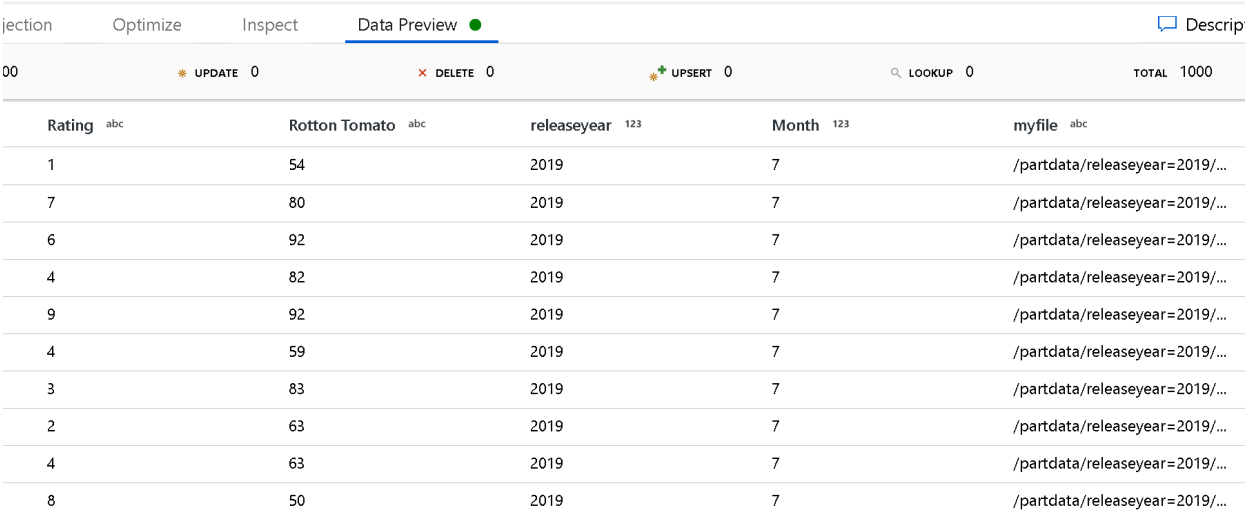
- `*` Represents any set of characters
- `**` Represents recursive directory nesting
- `?` Replaces one character
- `[]` Matches one of more characters in the brackets
- `/data/sales/**/*.csv` Gets all csv files under /data/sales
- `/data/sales/20??/**/` Gets all files in the 20th century
- `/data/sales/*/*/*.csv` Gets csv files two levels under /data/sales
- `/data/sales/2004/*/12/[XY]1?.csv` Gets all csv files in 2004 in December starting with X or Y prefixed by a two-digit number

Partition Root Path: If you have partitioned folders in your file source with a `key=value` format (for example, year=2019), then you can assign the top level of that partition folder tree to a column name in your data flow data stream.

First, set a wildcard to include all paths that are the partitioned folders plus the leaf files that you wish to read.



Use the Partition Root Path setting to define what the top level of the folder structure is. When you view the contents of your data via a data preview, you'll see that ADF will add the resolved partitions found in each of your folder levels.



Rating abc	Rotton Tomato abc	releaseyear 123	Month 123	myfile abc
1	54	2019	7	/partdata/releaseyear=2019/...
7	80	2019	7	/partdata/releaseyear=2019/...
6	92	2019	7	/partdata/releaseyear=2019/...
4	82	2019	7	/partdata/releaseyear=2019/...
9	92	2019	7	/partdata/releaseyear=2019/...
4	59	2019	7	/partdata/releaseyear=2019/...
3	83	2019	7	/partdata/releaseyear=2019/...
2	63	2019	7	/partdata/releaseyear=2019/...
4	63	2019	7	/partdata/releaseyear=2019/...
8	50	2019	7	/partdata/releaseyear=2019/...

List of files: This is a file set. Create a text file that includes a list of relative path files to process. Point to this text file.

Column to store file name: Store the name of the source file in a column in your data. Enter a new column name here to store the file name string.

After completion: Choose to do nothing with the source file after the data flow runs, delete the source file, or move the source file. The paths for the move are relative.

To move source files to another location post-processing, first select "Move" for file operation. Then, set the "from" directory. If you're not using any wildcards for your path, then the "from" setting will be the same folder as your source folder.

If you have a source path with wildcard, your syntax will look like this below:

```
/data/sales/20??/**/*.csv
```

You can specify "from" as

```
/data/sales
```

And "to" as

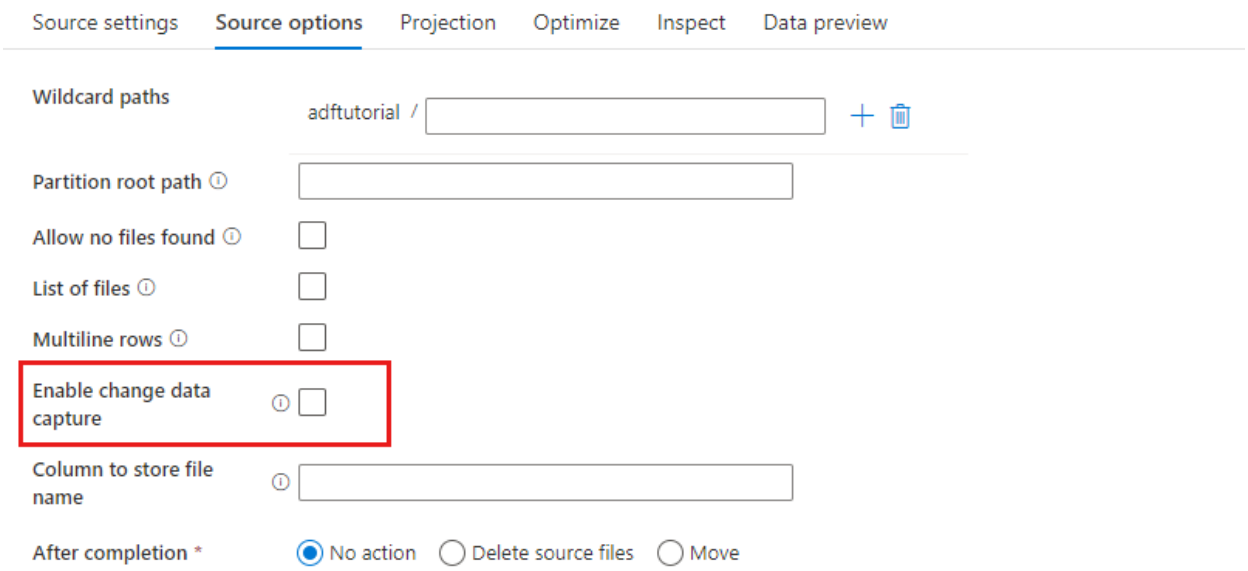
```
/backup/priorSales
```

In this case, all files that were sourced under /data/sales are moved to /backup/priorSales.

> **NOTE**
>
> File operations run only when you start the data flow from a pipeline run (a pipeline debug or execution run) that uses the Execute Data Flow activity in a pipeline. File operations *do not* run in Data Flow debug mode.

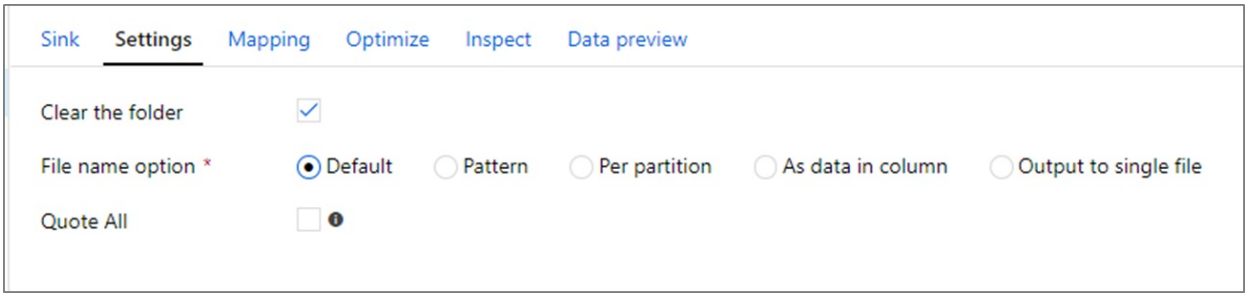
Filter by last modified: You can filter which files you process by specifying a date range of when they were last modified. All date-times are in UTC.

Enable change data capture: If true, you will get new or changed files only from the last run. Initial load of full snapshot data will always be gotten in the first run, followed by capturing new or changed files only in next runs. For more details, see Change data capture.



Sink properties

In the sink transformation, you can write to either a container or folder in Azure Data Lake Storage Gen2. the **Settings** tab lets you manage how the files get written.



Clear the folder: Determines whether or not the destination folder gets cleared before the data is written.

File name option: Determines how the destination files are named in the destination folder. The file name options are:

- **Default**: Allow Spark to name files based on PART defaults.
- **Pattern**: Enter a pattern that enumerates your output files per partition. For example, **loans[n].csv** will create loans1.csv, loans2.csv, and so on.
- **Per partition**: Enter one file name per partition.
- **As data in column**: Set the output file to the value of a column. The path is relative to the dataset container, not the destination folder. If you have a folder path in your dataset, it will be overridden.
- **Output to a single file**: Combine the partitioned output files into a single named file. The path is relative to the dataset folder. Please be aware that the merge operation can possibly fail based upon node size. This

option is not recommended for large datasets.

Quote all: Determines whether to enclose all values in quotes

`umask`

You can optionally set the `umask` for files using POSIX read, write, execute flags for owner, user and group.

Pre-processing and post-processing commands

You can optionally execute Hadoop filesystem commands before or after writing to an ADLS Gen2 sink. The following commands are supported:

- `cp`
- `mv`
- `rm`
- `mkdir`

Examples:

- `mkdir /folder1`
- `mkdir -p folder1`
- `mv /folder1/*.* /folder2/`
- `cp /folder1/file1.txt /folder2`
- `rm -r /folder1`

Parameters are also supported through expression builder, for example:

`mkdir -p {$tempPath}/commands/c1/c2` `mv {$tempPath}/commands/*.* {$tempPath}/commands/c1/c2`

By default, folders are created as user/root. Refer to the top level container with '/'.

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity

Delete activity properties

To learn details about the properties, check Delete activity

Legacy models

> **NOTE**
> The following models are still supported as-is for backward compatibility. You are suggested to use the new model mentioned in above sections going forward, and the ADF authoring UI has switched to generating the new model.

Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **AzureBlobFSFile**.	Yes
folderPath	Path to the folder in Data Lake Storage Gen2. If not specified, it points to the root. Wildcard filter is supported. Allowed wildcards are `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your actual folder name has a wildcard or this escape char is inside. Examples: filesystem/folder/. See more examples in Folder and file filter examples.	No
fileName	Name or wildcard filter for the files under the specified "folderPath". If you don't specify a value for this property, the dataset points to all files in the folder. For filter, the wildcards allowed are `*` (matches zero or more characters) and `?` (matches zero or single character). - Example 1: `"fileName": "*.csv"` - Example 2: `"fileName": "???20180427.txt"` Use `^` to escape if your actual file name has a wildcard or this escape char is inside. When fileName isn't specified for an output dataset and **preserveHierarchy** isn't specified in the activity sink, the copy activity automatically generates the file name with the following pattern: "*Data.[activity run ID GUID].[GUID if FlattenHierarchy].[format if configured].[compression if configured]*", for example, "Data.0a405f8a-93ff-4c6f-b3be-f69616f1df7a.txt.gz". If you copy from a tabular source using a table name instead of a query, the name pattern is "*[table name].[format].[compression if configured]*", for example, "MyTable.csv".	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	Files filter based on the attribute Last Modified. The files are selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z". The overall performance of data movement is affected by enabling this setting when you want to do file filter with huge amounts of files. The properties can be NULL, which means no file attribute filter is applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal to the datetime value are selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value are selected.	No
modifiedDatetimeEnd	Files filter based on the attribute Last Modified. The files are selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z". The overall performance of data movement is affected by enabling this setting when you want to do file filter with huge amounts of files. The properties can be NULL, which means no file attribute filter is applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal to the datetime value are selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value are selected.	No

PROPERTY	DESCRIPTION	REQUIRED
format	If you want to copy files as is between file-based stores (binary copy), skip the format section in both the input and output dataset definitions. If you want to parse or generate files with a specific format, the following file format types are supported: **TextFormat**, **JsonFormat**, **AvroFormat**, **OrcFormat**, and **ParquetFormat**. Set the **type** property under **format** to one of these values. For more information, see the Text format, JSON format, Avro format, ORC format, and Parquet format sections.	No (only for binary copy scenario)
compression	Specify the type and level of compression for the data. For more information, see Supported file formats and compression codecs. Supported types are `**GZip**, **Deflate**, **BZip2**, and **ZipDeflate**`. Supported levels are **Optimal** and **Fastest**.	No

TIP

To copy all files under a folder, specify **folderPath** only.
To copy a single file with a given name, specify **folderPath** with a folder part and **fileName** with a file name.
To copy a subset of files under a folder, specify **folderPath** with a folder part and **fileName** with a wildcard filter.

Example:

```
{
    "name": "ADLSGen2Dataset",
    "properties": {
        "type": "AzureBlobFSFile",
        "linkedServiceName": {
            "referenceName": "<Azure Data Lake Storage Gen2 linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "folderPath": "myfilesystem/myfolder",
            "fileName": "*",
            "modifiedDatetimeStart": "2018-12-01T05:00:00Z",
            "modifiedDatetimeEnd": "2018-12-01T06:00:00Z",
            "format": {
                "type": "TextFormat",
                "columnDelimiter": ",",
                "rowDelimiter": "\n"
            },
            "compression": {
                "type": "GZip",
                "level": "Optimal"
            }
        }
    }
}
```

Legacy copy activity source model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **AzureBlobFSSource**.	Yes
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. When recursive is set to true and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromADLSGen2",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<ADLS Gen2 input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AzureBlobFSSource",
                "recursive": true
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Legacy copy activity sink model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to **AzureBlobFSSink**.	Yes
copyBehavior	Defines the copy behavior when the source is files from a file-based data store. Allowed values are: - **PreserveHierarchy (default)**: Preserves the file hierarchy in the target folder. The relative path of the source file to the source folder is identical to the relative path of the target file to the target folder. - **FlattenHierarchy**: All files from the source folder are in the first level of the target folder. The target files have autogenerated names. - **MergeFiles**: Merges all files from the source folder to one file. If the file name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToADLSGen2",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<ADLS Gen2 output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureBlobFSSink",
                "copyBehavior": "PreserveHierarchy"
            }
        }
    }
]
```

Change data capture

Azure Data Factory can get new or changed files only from Azure Data Lake Storage Gen2 by enabling **Enable change data capture** in the mapping data flow source transformation. With this connector option, you can read new or updated files only and apply transformations before loading transformed data into destination datasets of your choice.

Make sure you keep the pipeline and activity name unchanged, so that the checkpoint can always be recorded from the last run to get changes from there. If you change your pipeline name or activity name, the checkpoint will be reset, and you will start from the beginning in the next run.

When you debug the pipeline, the **Enable change data capture** works as well. Be aware that the checkpoint will be reset when you refresh your browser during the debug run. After you are satisfied with the result from debug run, you can publish and trigger the pipeline. It will always start from the beginning regardless of the previous checkpoint recorded by debug run.

In the monitoring section, you always have the chance to rerun a pipeline. When you are doing so, the changes are always gotten from the checkpoint record in your selected pipeline run.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from Azure Database for MariaDB using Azure Data Factory or Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Azure Database for MariaDB. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Azure Database for MariaDB connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/-)	① ②	✓
Lookup activity	① ②	✓

① Azure integration runtime ② Self-hosted integration runtime

You can copy data from Azure Database for MariaDB to any supported sink data store. For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

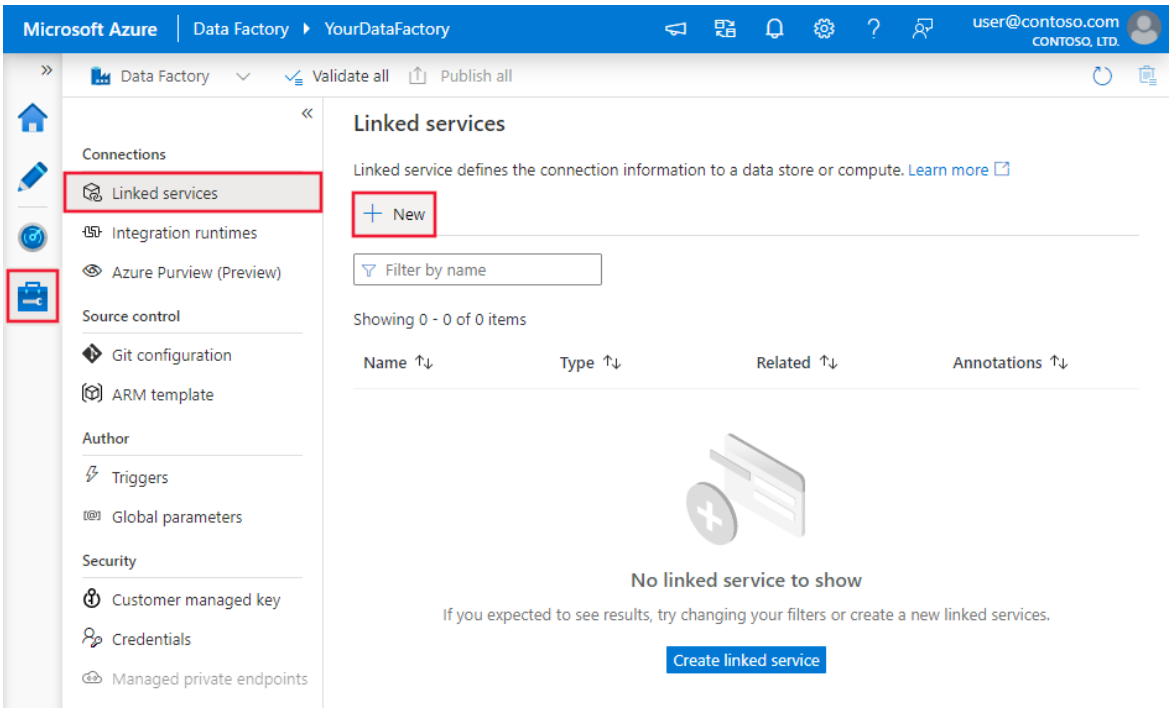
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Azure Database for MariaDB using UI

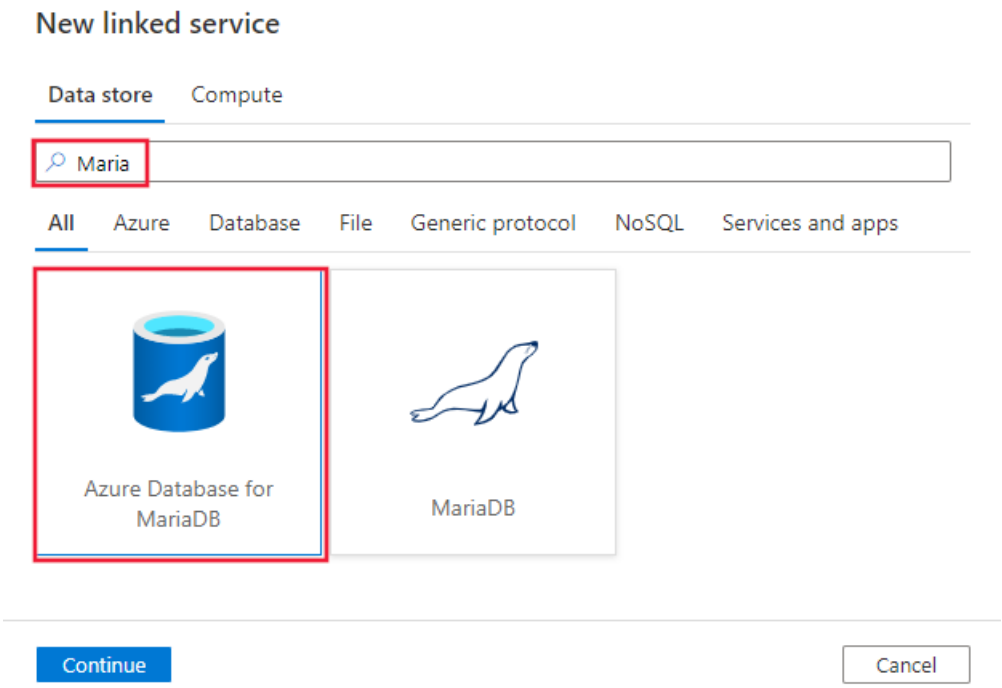
Use the following steps to create a linked service to Azure Database for MariaDB in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

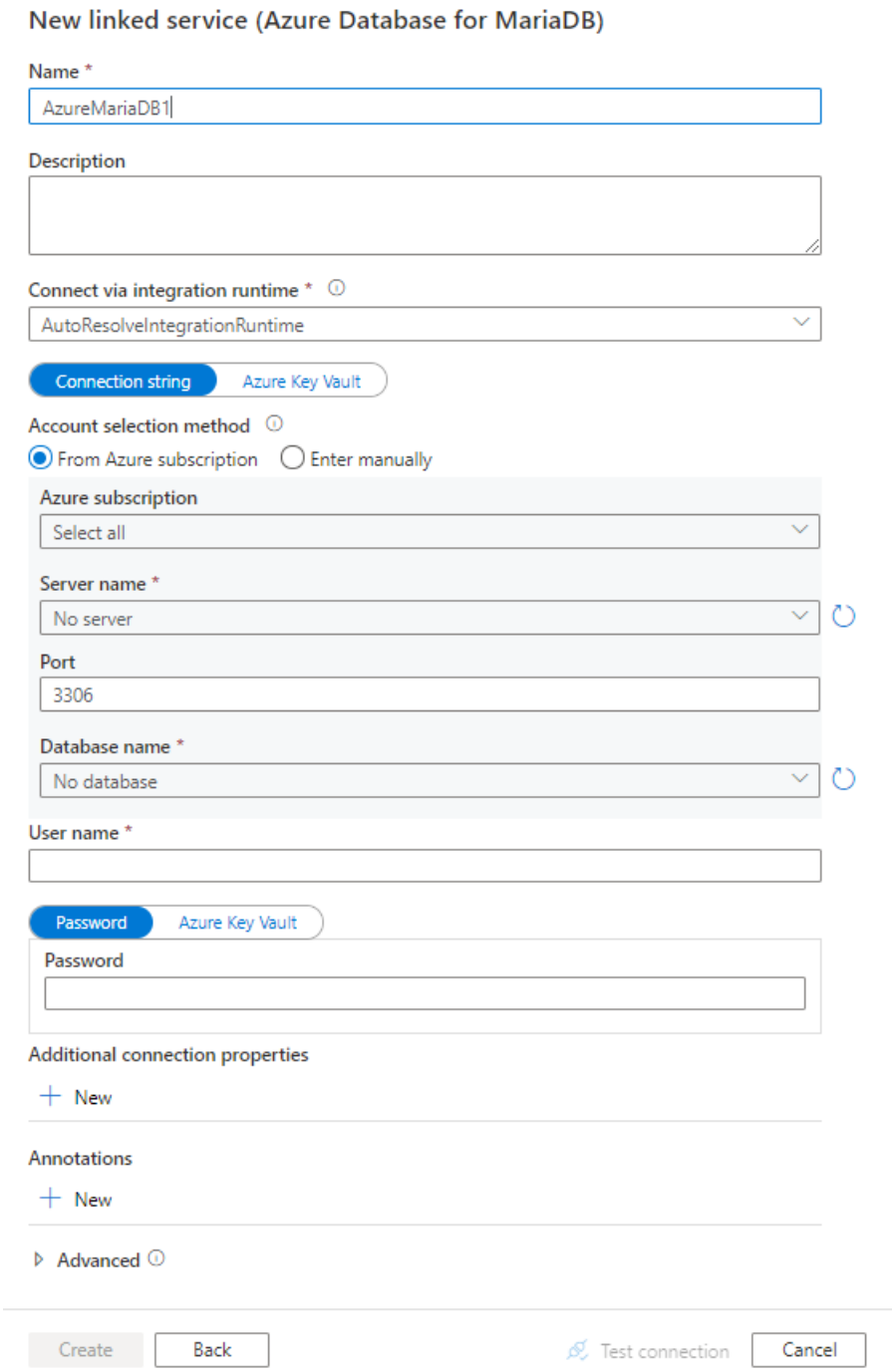
   - Azure Data Factory
   - Azure Synapse



2. Search for Maria and select the Azure Database for MariaDB connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Azure Database for MariaDB connector.

Linked service properties

The following properties are supported for Azure Database for MariaDB linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **AzureMariaDB**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	A connection string to connect to Azure Database for MariaDB. You can find it from the Azure portal -> your Azure Database for MariaDB -> Connection strings -> ADO.NET one. You can also put password in Azure Key Vault and pull the `pwd` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "AzureDatabaseForMariaDBLinkedService",
    "properties": {
        "type": "AzureMariaDB",
        "typeProperties": {
            "connectionString": "Server={your_server}.mariadb.database.azure.com; Port=3306; Database=
{your_database}; Uid={your_user}@{your_server}; Pwd={your_password}; SslMode=Preferred;"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "AzureDatabaseForMariaDBLinkedService",
    "properties": {
        "type": "AzureMariaDB",
        "typeProperties": {
            "connectionString": "Server={your_server}.mariadb.database.azure.com; Port=3306; Database=
{your_database}; Uid={your_user}@{your_server}; SslMode=Preferred;",
            "pwd": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Azure Database for MariaDB dataset.

To copy data from Azure Database for MariaDB, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **AzureMariaDBTable**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "AzureDatabaseForMariaDBDataset",
    "properties": {
        "type": "AzureMariaDBTable",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Azure Database for MariaDB linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Azure Database for MariaDB source.

Azure Database for MariaDB as source

To copy data from Azure Database for MariaDB, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **AzureMariaDBSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromAzureDatabaseForMariaDB",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Azure Database for MariaDB input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AzureMariaDBSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy and transform data in Azure Database for MySQL using Azure Data Factory or Synapse Analytics

8/19/2022 • 10 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory or Synapse Analytics pipelines to copy data from and to Azure Database for MySQL, and use Data Flow to transform data in Azure Database for MySQL. To learn more, read the introductory articles for Azure Data Factory and Synapse Analytics.

This connector is specialized for

- Azure Database for MySQL Single Server
- Azure Database for MySQL Flexible Server (Currently public access is only supported)

To copy data from generic MySQL database located on-premises or in the cloud, use MySQL connector.

Prerequisites

This quickstart requires the following resources and configuration mentioned below as a starting point:

- An existing Azure database for MySQL Single server or MySQL Flexible Server.
- Enable **Allow public access from any Azure service within Azure to this server** in networking page of the MySQL server . This will allow you to use Data factory studio.

Supported capabilities

This Azure Database for MySQL connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/sink)	① ②	✓
Mapping data flow (source/sink)	①	✓
Lookup activity	① ②	✓

① Azure integration runtime ② Self-hosted integration runtime

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell

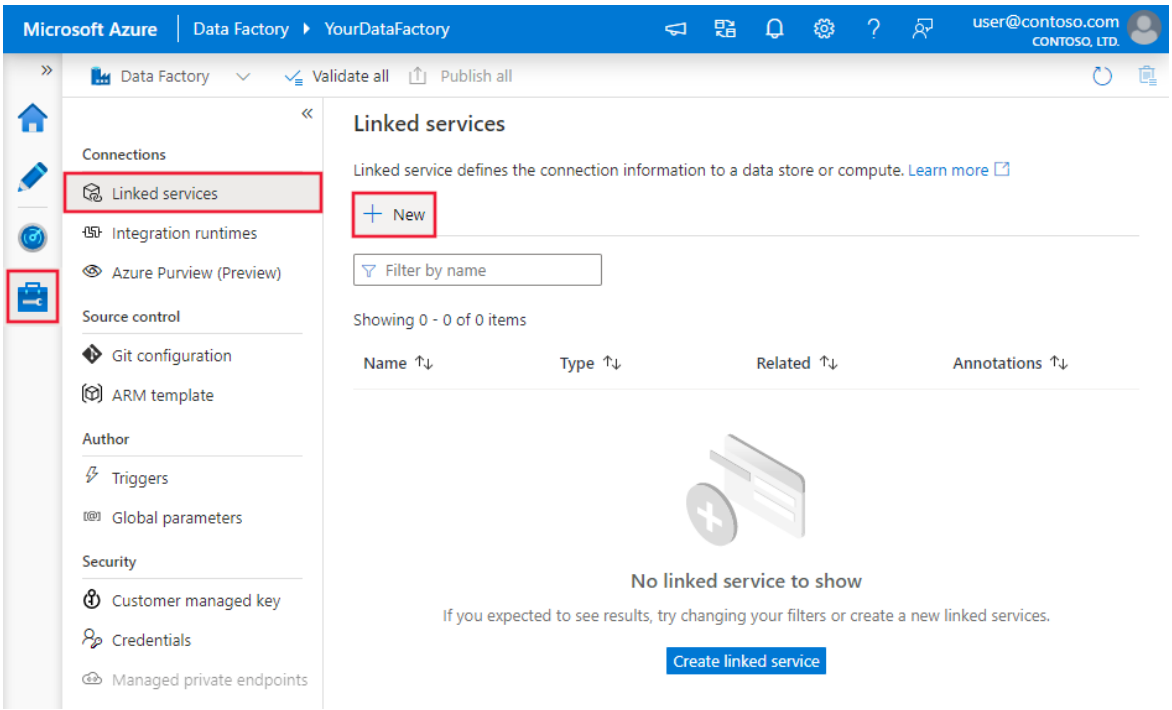
- The REST API
- The Azure Resource Manager template

Create a linked service to Azure Database for MySQL using UI

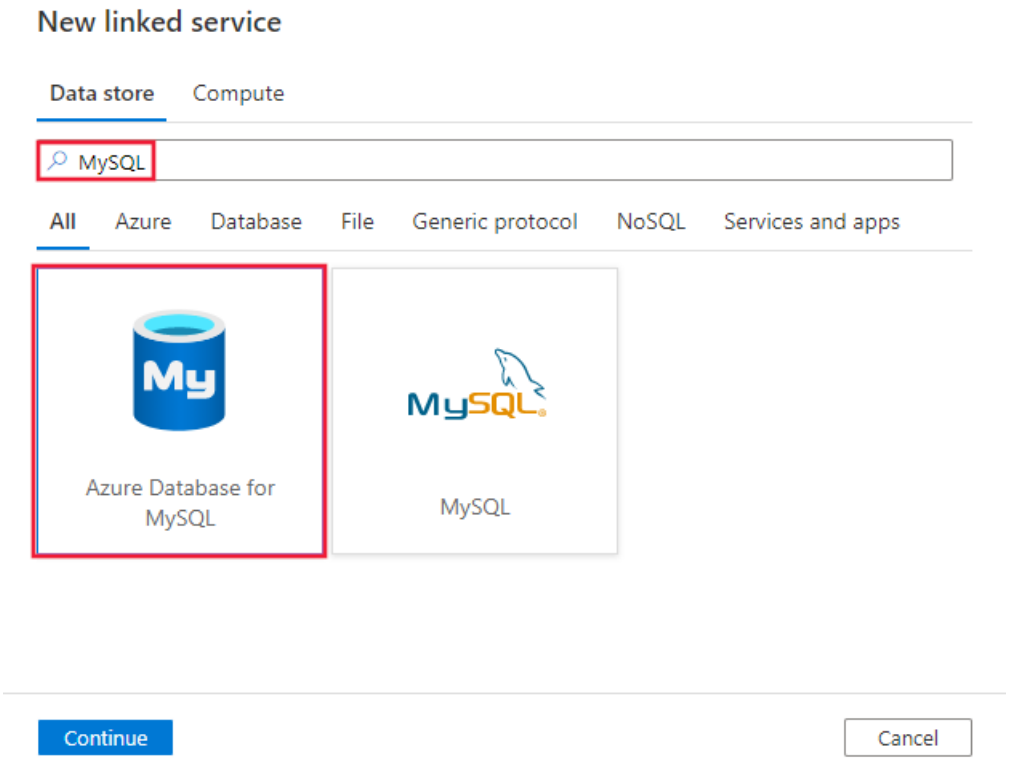
Use the following steps to create a linked service to Azure Database for MySQL in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

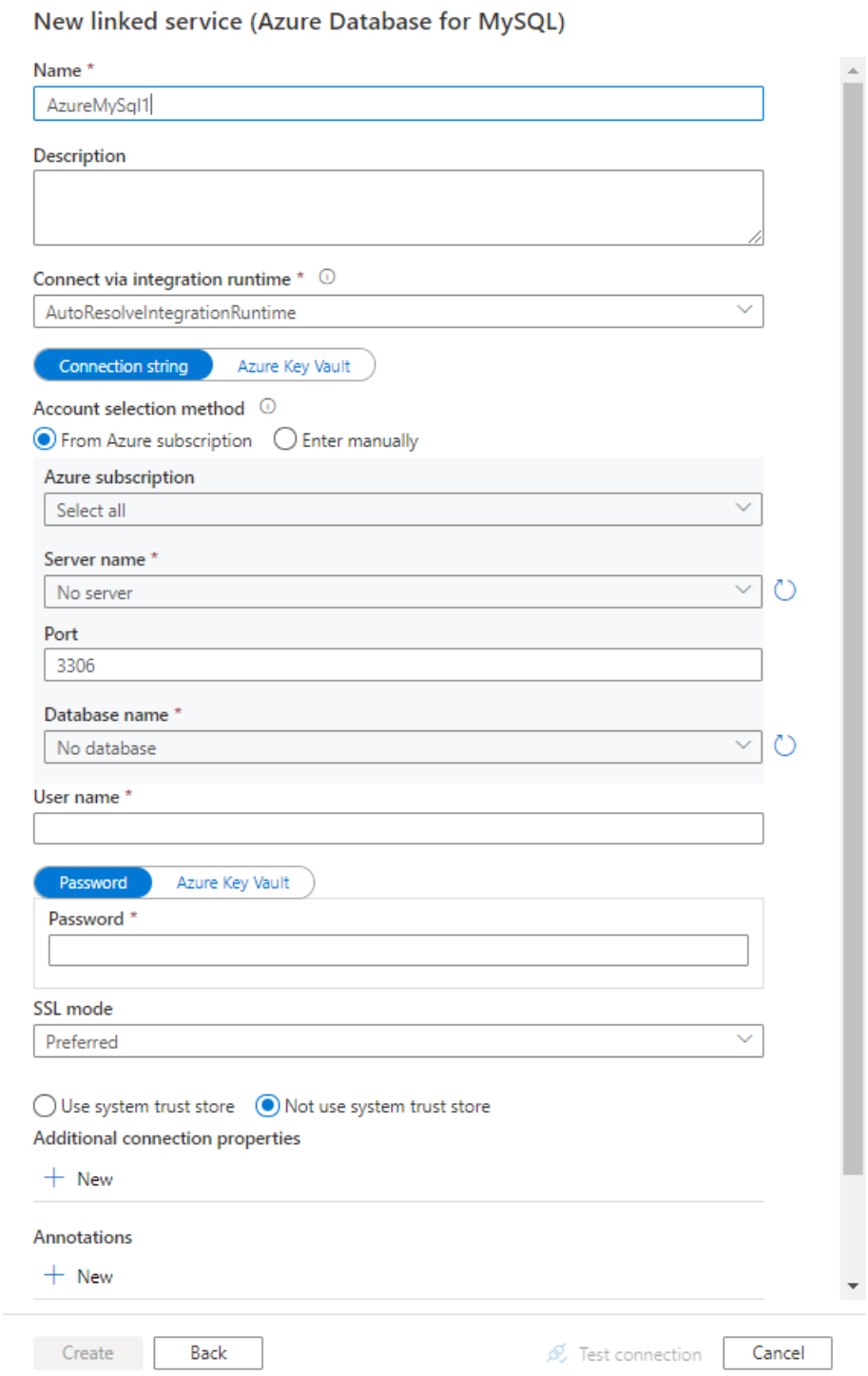
   - Azure Data Factory
   - Azure Synapse



2. Search for MySQL and select the Azure Database for MySQL connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Azure Database for MySQL connector.

Linked service properties

The following properties are supported for Azure Database for MySQL linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **AzureMySql**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	Specify information needed to connect to the Azure Database for MySQL instance. You can also put password in Azure Key Vault and pull the `password` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is located in private network). If not specified, it uses the default Azure Integration Runtime.	No

A typical connection string is `Server=<server>.mysql.database.azure.com;Port=<port>;Database=<database>;UID=<username>;PWD=<password>`. More properties you can set per your case:

PROPERTY	DESCRIPTION	OPTIONS	REQUIRED
SSLMode	This option specifies whether the driver uses TLS encryption and verification when connecting to MySQL. E.g. `SSLMode=<0/1/2/3/4>`	DISABLED (0) / PREFERRED (1) **(Default)** / REQUIRED (2) / VERIFY_CA (3) / VERIFY_IDENTITY (4)	No
UseSystemTrustStore	This option specifies whether to use a CA certificate from the system trust store, or from a specified PEM file. E.g. `UseSystemTrustStore=<0/1>;`	Enabled (1) / Disabled (0) **(Default)**	No

Example:

```
{
    "name": "AzureDatabaseForMySQLLinkedService",
    "properties": {
        "type": "AzureMySql",
        "typeProperties": {
            "connectionString": "Server=<server>.mysql.database.azure.com;Port=<port>;Database=
<database>;UID=<username>;PWD=<password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "AzureDatabaseForMySQLLinkedService",
    "properties": {
        "type": "AzureMySql",
        "typeProperties": {
            "connectionString": "Server=<server>.mysql.database.azure.com;Port=<port>;Database=
<database>;UID=<username>;",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Azure Database for MySQL dataset.

To copy data from Azure Database for MySQL, set the type property of the dataset to **AzureMySqlTable**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **AzureMySqlTable**	Yes
tableName	Name of the table in the MySQL database.	No (if "query" in activity source is specified)

Example

```
{
    "name": "AzureMySQLDataset",
    "properties": {
        "type": "AzureMySqlTable",
        "linkedServiceName": {
            "referenceName": "<Azure MySQL linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "tableName": "<table name>"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Azure Database for MySQL source and sink.

Azure Database for MySQL as source

To copy data from Azure Database for MySQL, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **AzureMySqlSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)
queryCommandTimeout	The wait time before the query request times out. Default is 120 minutes (02:00:00)	No

Example:

```
"activities":[
    {
        "name": "CopyFromAzureDatabaseForMySQL",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Azure MySQL input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AzureMySqlSource",
                "query": "<custom query e.g. SELECT * FROM MyTable>"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Azure Database for MySQL as sink

To copy data to Azure Database for MySQL, the following properties are supported in the copy activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to: **AzureMySqlSink**	Yes

PROPERTY	DESCRIPTION	REQUIRED
preCopyScript	Specify a SQL query for the copy activity to execute before writing data into Azure Database for MySQL in each run. You can use this property to clean up the preloaded data.	No
writeBatchSize	Inserts data into the Azure Database for MySQL table when the buffer size reaches writeBatchSize. Allowed value is integer representing number of rows.	No (default is 10,000)
writeBatchTimeout	Wait time for the batch insert operation to complete before it times out. Allowed values are Timespan. An example is 00:30:00 (30 minutes).	No (default is 00:00:30)

Example:

```
"activities":[
    {
        "name": "CopyToAzureDatabaseForMySQL",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Azure MySQL output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureMySqlSink",
                "preCopyScript": "<custom SQL script>",
                "writeBatchSize": 100000
            }
        }
    }
]
```

Mapping data flow properties

When transforming data in mapping data flow, you can read and write to tables from Azure Database for MySQL. For more information, see the source transformation and sink transformation in mapping data flows. You can choose to use an Azure Database for MySQL dataset or an inline dataset as source and sink type.

Source transformation

The below table lists the properties supported by Azure Database for MySQL source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Table	If you select Table as input, data flow fetches all the data from the table specified in the dataset.	No	-	*(for inline dataset only)* tableName

Query	If you select Query as input, specify a SQL query to fetch data from source, which overrides any table you specify in dataset. Using queries is a great way to reduce rows for testing or lookups.
	
	Order By clause is not supported, but you can set a full SELECT FROM statement. You can also use user-defined table functions. **select * from udfGetData()** is a UDF in SQL that returns a table that you can use in data flow.
	Query example:
	`select * from mytable where customerId > 1000 and customerId < 2000`
	or
	`select * from "MyTable"`
	.
		No	String	query

Stored procedure	If you select Stored procedure as input, specify a name of the stored procedure to read data from the source table, or select Refresh to ask the service to discover the procedure names.	Yes (if you select Stored procedure as input)	String	procedureName

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Procedure parameters	If you select Stored procedure as input, specify any input parameters for the stored procedure in the order set in the procedure, or select Import to import all procedure parameters using the form `@paraName`.	No	Array	inputs
Batch size	Specify a batch size to chunk large data into batches.	No	Integer	batchSize
Isolation Level	Choose one of the following isolation levels: - Read Committed - Read Uncommitted (default) - Repeatable Read - Serializable - None (ignore isolation level)	No	READ_COMMITTED READ_UNCOMMITTED REPEATABLE_READ SERIALIZABLE NONE	isolationLevel

Azure Database for MySQL source script example

When you use Azure Database for MySQL as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
    validateSchema: false,
    isolationLevel: 'READ_UNCOMMITTED',
    query: 'select * from mytable',
    format: 'query') ~> AzureMySQLSource
```

Sink transformation

The below table lists the properties supported by Azure Database for MySQL sink. You can edit these properties in the **Sink options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Update method	Specify what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an Alter row transformation is required to tag rows for those actions.	Yes	`true` or `false`	deletable insertable updateable upsertable

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Key columns	For updates, upserts and deletes, key column(s) must be set to determine which row to alter. The column name that you pick as the key will be used as part of the subsequent update, upsert, delete. Therefore, you must pick a column that exists in the Sink mapping.	No	Array	keys
Skip writing key columns	If you wish to not write the value to the key column, select "Skip writing key columns".	No	`true` or `false`	skipKeyWrites
Table action	Determines whether to recreate or remove all rows from the destination table prior to writing. - **None**: No action will be done to the table. - **Recreate**: The table will get dropped and recreated. Required if creating a new table dynamically. - **Truncate**: All rows from the target table will get removed.	No	`true` or `false`	recreate truncate
Batch size	Specify how many rows are being written in each batch. Larger batch sizes improve compression and memory optimization, but risk out of memory exceptions when caching data.	No	Integer	batchSize
Pre and Post SQL scripts	Specify multi-line SQL scripts that will execute before (pre-processing) and after (post-processing) data is written to your Sink database.	No	String	preSQLs postSQLs

TIP

1. It's recommended to break single batch scripts with multiple commands into multiple batches.
2. Only Data Definition Language (DDL) and Data Manipulation Language (DML) statements that return a simple update count can be run as part of a batch. Learn more from Performing batch operations

- Enable incremental extract: Use this option to tell ADF to only process rows that have changed since the last time that the pipeline executed.
- Incremental date column: When using the incremental extract feature, you must choose the date/time column that you wish to use as the watermark in your source table.
- Start reading from beginning: Setting this option with incremental extract will instruct ADF to read all rows on first execution of a pipeline with incremental extract turned on.

Azure Database for MySQL sink script example

When you use Azure Database for MySQL as sink type, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    deletable:false,
    insertable:true,
    updateable:true,
    upsertable:true,
    keys:['keyColumn'],
    format: 'table',
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> AzureMySQLSink
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Data type mapping for Azure Database for MySQL

When copying data from Azure Database for MySQL, the following mappings are used from MySQL data types to interim data types used internally within the service. See Schema and data type mappings to learn about how copy activity maps the source schema and data type to the sink.

AZURE DATABASE FOR MYSQL DATA TYPE	INTERIM SERVICE DATA TYPE
`bigint`	`Int64`
`bigint unsigned`	`Decimal`
`bit`	`Boolean`
`bit(M), M>1`	`Byte[]`
`blob`	`Byte[]`
`bool`	`Int16`
`char`	`String`

AZURE DATABASE FOR MYSQL DATA TYPE	INTERIM SERVICE DATA TYPE
`date`	`Datetime`
`datetime`	`Datetime`
`decimal`	`Decimal, String`
`double`	`Double`
`double precision`	`Double`
`enum`	`String`
`float`	`Single`
`int`	`Int32`
`int unsigned`	`Int64`
`integer`	`Int32`
`integer unsigned`	`Int64`
`long varbinary`	`Byte[]`
`long varchar`	`String`
`longblob`	`Byte[]`
`longtext`	`String`
`mediumblob`	`Byte[]`
`mediumint`	`Int32`
`mediumint unsigned`	`Int64`
`mediumtext`	`String`
`numeric`	`Decimal`
`real`	`Double`
`set`	`String`
`smallint`	`Int16`
`smallint unsigned`	`Int32`

AZURE DATABASE FOR MYSQL DATA TYPE	INTERIM SERVICE DATA TYPE
`text`	`String`
`time`	`TimeSpan`
`timestamp`	`Datetime`
`tinyblob`	`Byte[]`
`tinyint`	`Int16`
`tinyint unsigned`	`Int16`
`tinytext`	`String`
`varchar`	`String`
`year`	`Int32`

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy and transform data in Azure Database for PostgreSQL using Azure Data Factory or Synapse Analytics

8/19/2022 • 15 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from and to Azure Database for PostgreSQL, and use Data Flow to transform data in Azure Database for PostgreSQL. To learn more, read the introductory articles for Azure Data Factory and Synapse Analytics.

This connector is specialized for the Azure Database for PostgreSQL service. To copy data from a generic PostgreSQL database located on-premises or in the cloud, use the PostgreSQL connector.

Supported capabilities

This Azure Database for PostgreSQL connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/sink)	① ②	✓
Mapping data flow (source/sink)	①	✓
Lookup activity	① ②	✓

① Azure integration runtime ② Self-hosted integration runtime

The three activities work on all Azure Database for PostgreSQL deployment options:

- Single Server
- Flexible Server
- Hyperscale (Citus)

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

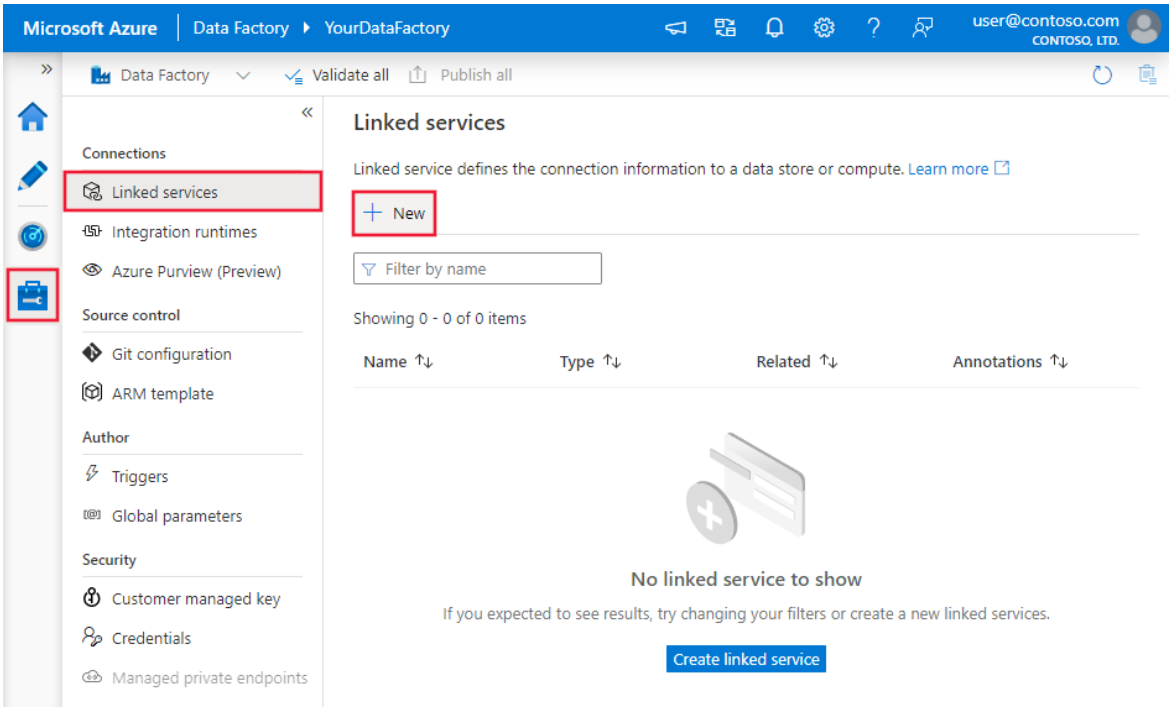
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Azure database for PostgreSQL using UI

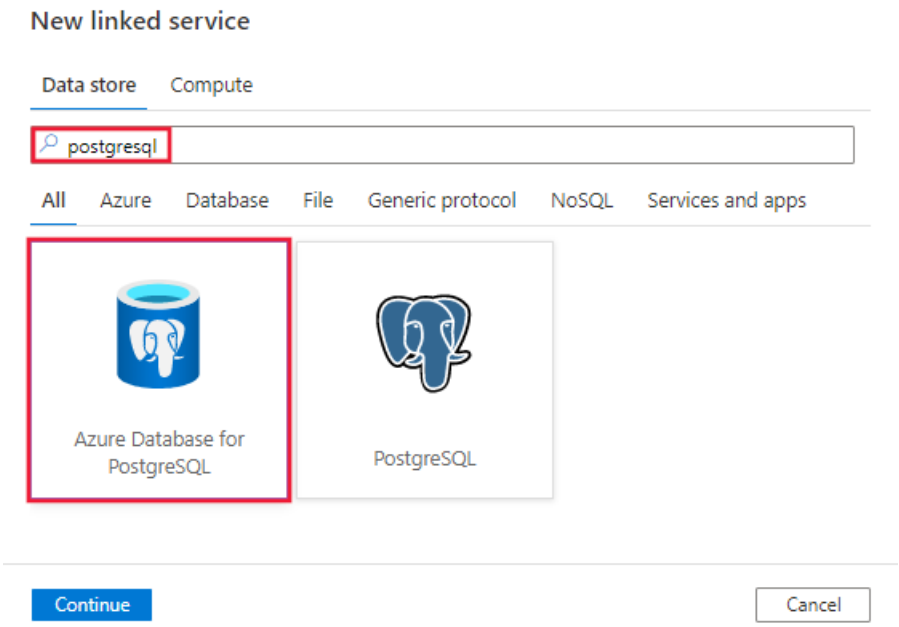
Use the following steps to create a linked service to Azure database for PostgreSQL in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

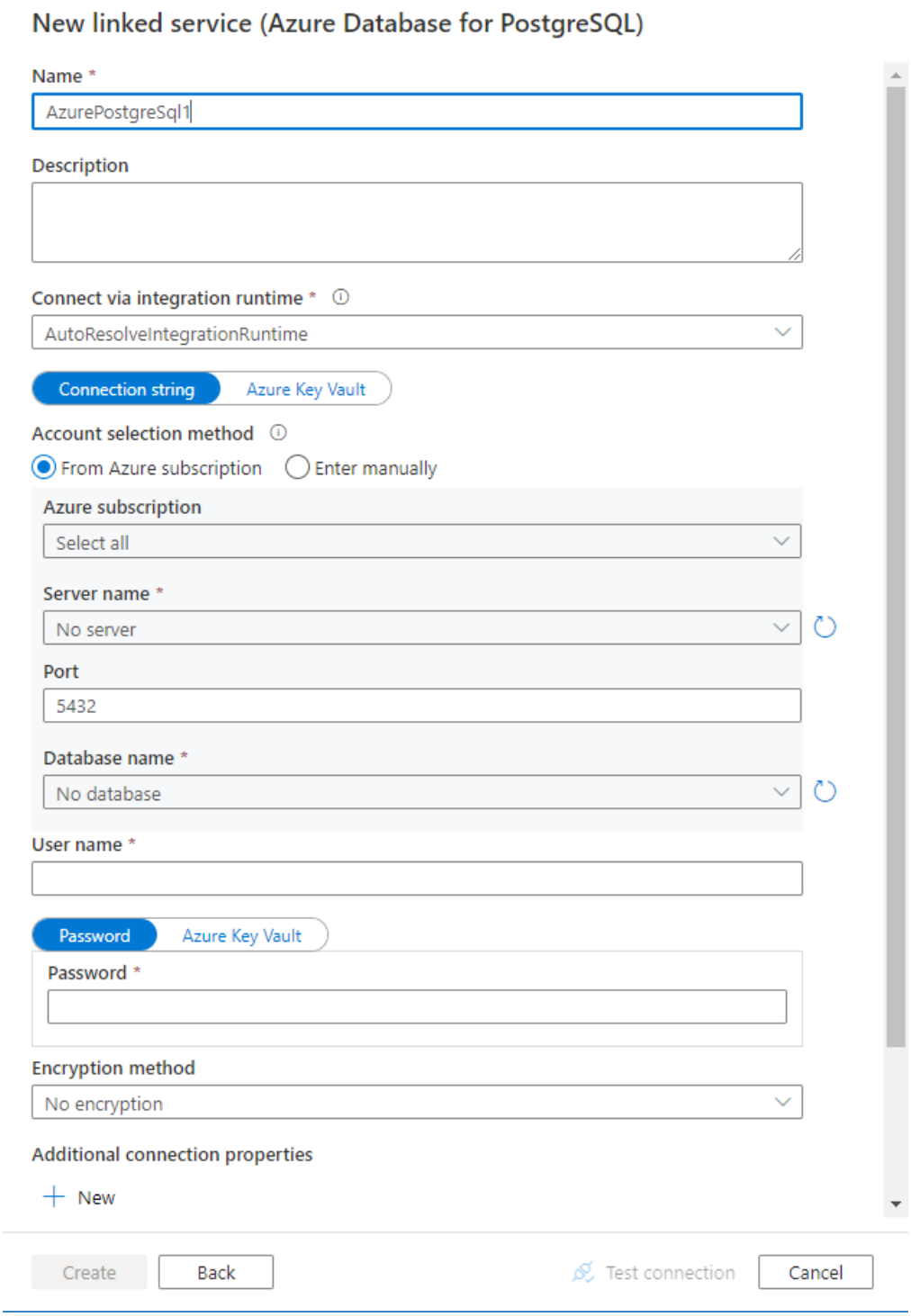
   - Azure Data Factory
   - Azure Synapse



2. Search for PostgreSQL and select the Azure database for PostgreSQL connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections offer details about properties that are used to define Data Factory entities specific to Azure Database for PostgreSQL connector.

Linked service properties

The following properties are supported for the Azure Database for PostgreSQL linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **AzurePostgreSql**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	An ODBC connection string to connect to Azure Database for PostgreSQL. You can also put a password in Azure Key Vault and pull the `password` configuration out of the connection string. See the following samples and Store credentials in Azure Key Vault for more details.	Yes
connectVia	This property represents the integration runtime to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is located in private network). If not specified, it uses the default Azure Integration Runtime.	No

A typical connection string is `Server=<server>.postgres.database.azure.com;Database=<database>;Port=<port>;UID=<username>;Password=<Password>`. Here are more properties you can set per your case:

PROPERTY	DESCRIPTION	OPTIONS	REQUIRED
EncryptionMethod (EM)	The method the driver uses to encrypt data sent between the driver and the database server. For example, `EncryptionMethod=<0/1/6>;`	0 (No Encryption) **(Default)** / 1 (SSL) / 6 (RequestSSL)	No
ValidateServerCertificate (VSC)	Determines whether the driver validates the certificate that's sent by the database server when SSL encryption is enabled (Encryption Method=1). For example, `ValidateServerCertificate=<0/1>;`	0 (Disabled) **(Default)** / 1 (Enabled)	No

Example:

```
{
    "name": "AzurePostgreSqlLinkedService",
    "properties": {
        "type": "AzurePostgreSql",
        "typeProperties": {
            "connectionString": "Server=<server>.postgres.database.azure.com;Database=<database>;Port=
<port>;UID=<username>;Password=<Password>"
        }
    }
}
```

Example:

Store password in Azure Key Vault

```
{
    "name": "AzurePostgreSqlLinkedService",
    "properties": {
        "type": "AzurePostgreSql",
        "typeProperties": {
            "connectionString": "Server=<server>.postgres.database.azure.com;Database=<database>;Port=
<port>;UID=<username>;",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see Datasets. This section provides a list of properties that Azure Database for PostgreSQL supports in datasets.

To copy data from Azure Database for PostgreSQL, set the type property of the dataset to **AzurePostgreSqlTable**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **AzurePostgreSqlTable**	Yes
tableName	Name of the table	No (if "query" in activity source is specified)

Example:

```
{
    "name": "AzurePostgreSqlDataset",
    "properties": {
        "type": "AzurePostgreSqlTable",
        "linkedServiceName": {
            "referenceName": "<AzurePostgreSql linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {}
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see Pipelines and activities. This section provides a list of properties supported by an Azure Database for PostgreSQL source.

Azure Database for PostgreSql as source

To copy data from Azure Database for PostgreSQL, set the source type in the copy activity to

AzurePostgreSqlSource. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **AzurePostgreSqlSource**	Yes
query	Use the custom SQL query to read data. For example: `SELECT * FROM mytable` or `SELECT * FROM "MyTable"`. Note in PostgreSQL, the entity name is treated as case-insensitive if not quoted.	No (if the tableName property in the dataset is specified)
partitionOptions	Specifies the data partitioning options used to load data from Azure SQL Database. Allowed values are: **None** (default), **PhysicalPartitionsOfTable**, and **DynamicRange**. When a partition option is enabled (that is, not `None`), the degree of parallelism to concurrently load data from an Azure SQL Database is controlled by the `parallelCopies` setting on the copy activity.	No
partitionSettings	Specify the group of the settings for data partitioning. Apply when the partition option isn't `None`.	No
Under `partitionSettings`:		
partitionNames	The list of physical partitions that needs to be copied. Apply when the partition option is `PhysicalPartitionsOfTable`. If you use a query to retrieve the source data, hook `?AdfTabularPartitionName` in the WHERE clause. For an example, see the Parallel copy from Azure Database for PostgreSQL section.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionColumnName	Specify the name of the source column **in integer or date/datetime type** (`int` , `smallint` , `bigint` , `date` , `timestamp without time zone` , `timestamp with time zone` or `time without time zone`) that will be used by range partitioning for parallel copy. If not specified, the primary key of the table is auto-detected and used as the partition column. Apply when the partition option is `DynamicRange` . If you use a query to retrieve the source data, hook `?AdfRangePartitionColumnName` in the WHERE clause. For an example, see the Parallel copy from Azure Database for PostgreSQL section.	No
partitionUpperBound	The maximum value of the partition column to copy data out. Apply when the partition option is `DynamicRange` . If you use a query to retrieve the source data, hook `?AdfRangePartitionUpbound` in the WHERE clause. For an example, see the Parallel copy from Azure Database for PostgreSQL section.	No
partitionLowerBound	The minimum value of the partition column to copy data out. Apply when the partition option is `DynamicRange` . If you use a query to retrieve the source data, hook `?AdfRangePartitionLowbound` in the WHERE clause. For an example, see the Parallel copy from Azure Database for PostgreSQL section.	No

Example:

```
"activities":[
    {
        "name": "CopyFromAzurePostgreSql",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<AzurePostgreSql input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AzurePostgreSqlSource",
                "query": "<custom query e.g. SELECT * FROM mytable>"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Azure Database for PostgreSQL as sink

To copy data to Azure Database for PostgreSQL, the following properties are supported in the copy activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to **AzurePostgreSQLSink**.	Yes
preCopyScript	Specify a SQL query for the copy activity to execute before you write data into Azure Database for PostgreSQL in each run. You can use this property to clean up the preloaded data.	No
writeMethod	The method used to write data into Azure Database for PostgreSQL. Allowed values are: **CopyCommand** (default, which is more performant), **BulkInsert**.	No
writeBatchSize	The number of rows loaded into Azure Database for PostgreSQL per batch. Allowed value is an integer that represents the number of rows.	No (default is 1,000,000)

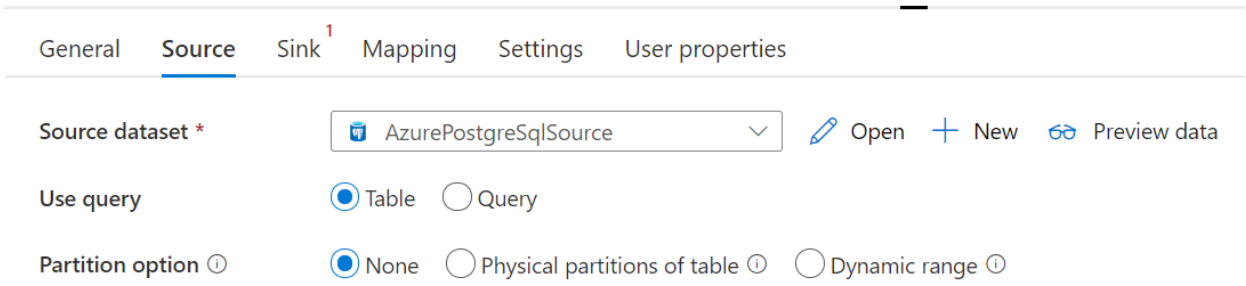
PROPERTY	DESCRIPTION	REQUIRED
writeBatchTimeout	Wait time for the batch insert operation to complete before it times out. Allowed values are Timespan strings. An example is 00:30:00 (30 minutes).	No (default is 00:30:00)

Example:

```
"activities":[
    {
        "name": "CopyToAzureDatabaseForPostgreSQL",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Azure PostgreSQL output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzurePostgreSQLSink",
                "preCopyScript": "<custom SQL script>",
                "writeMethod": "CopyCommand",
                "writeBatchSize": 1000000
            }
        }
    }
]
```

Parallel copy from Azure Database for PostgreSQL

The Azure Database for PostgreSQL connector in copy activity provides built-in data partitioning to copy data in parallel. You can find data partitioning options on the **Source** tab of the copy activity.



When you enable partitioned copy, copy activity runs parallel queries against your Azure Database for PostgreSQL source to load data by partitions. The parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your Azure Database for PostgreSQL.

You are suggested to enable parallel copy with data partitioning especially when you load large amount of data from your Azure Database for PostgreSQL. The following are suggested configurations for different scenarios. When copying data into file-based data store, it's recommended to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

SCENARIO	SUGGESTED SETTINGS
Full load from large table, with physical partitions.	**Partition option**: Physical partitions of table. During execution, the service automatically detects the physical partitions, and copies data by partitions.
Full load from large table, without physical partitions, while with an integer column for data partitioning.	**Partition options**: Dynamic range partition. **Partition column**: Specify the column used to partition data. If not specified, the primary key column is used.
Load a large amount of data by using a custom query, with physical partitions.	**Partition option**: Physical partitions of table. **Query**: `SELECT * FROM ?AdfTabularPartitionName WHERE <your_additional_where_clause>` . **Partition name**: Specify the partition name(s) to copy data from. If not specified, the service automatically detects the physical partitions on the table you specified in the PostgreSQL dataset. During execution, the service replaces `?AdfTabularPartitionName` with the actual partition name, and sends to Azure Database for PostgreSQL.
Load a large amount of data by using a custom query, without physical partitions, while with an integer column for data partitioning.	**Partition options**: Dynamic range partition. **Query**: `SELECT * FROM ?AdfTabularPartitionName WHERE ? AdfRangePartitionColumnName <= ? AdfRangePartitionUpbound AND ? AdfRangePartitionColumnName >= ? AdfRangePartitionLowbound AND <your_additional_where_clause>` . **Partition column**: Specify the column used to partition data. You can partition against the column with integer or date/datetime data type. **Partition upper bound** and **partition lower bound**: Specify if you want to filter against partition column to retrieve data only between the lower and upper range. During execution, the service replaces `?AdfRangePartitionColumnName`, `?AdfRangePartitionUpbound`, and `?AdfRangePartitionLowbound` with the actual column name and value ranges for each partition, and sends to Azure Database for PostgreSQL. For example, if your partition column "ID" is set with the lower bound as 1 and the upper bound as 80, with parallel copy set as 4, the service retrieves data by 4 partitions. Their IDs are between [1,20], [21, 40], [41, 60], and [61, 80], respectively.

Best practices to load data with partition option:

1. Choose distinctive column as partition column (like primary key or unique key) to avoid data skew.
2. If the table has built-in partition, use partition option "Physical partitions of table" to get better performance.

3. If you use Azure Integration Runtime to copy data, you can set larger "Data Integration Units (DIU)" (>4) to utilize more computing resource. Check the applicable scenarios there.
4. "Degree of copy parallelism" control the partition numbers, setting this number too large sometime hurts the performance, recommend setting this number as (DIU or number of Self-hosted IR nodes) * (2 to 4).

Example: full load from large table with physical partitions

```
"source": {
    "type": "AzurePostgreSqlSource",
    "partitionOption": "PhysicalPartitionsOfTable"
}
```

Example: query with dynamic range partition

```
"source": {
    "type": "AzurePostgreSqlSource",
    "query":"SELECT * FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition AND
<your_additional_where_clause>",
    "partitionOption": "DynamicRange",
    "partitionSettings": {
        "partitionColumnName": "<partition_column_name>",
        "partitionUpperBound": "<upper_value_of_partition_column (optional) to decide the partition stride,
not as data filter>",
        "partitionLowerBound": "<lower_value_of_partition_column (optional) to decide the partition stride,
not as data filter>"
    }
}
```

Mapping data flow properties

When transforming data in mapping data flow, you can read and write to tables from Azure Database for PostgreSQL. For more information, see the source transformation and sink transformation in mapping data flows. You can choose to use an Azure Database for PostgreSQL dataset or an inline dataset as source and sink type.

Source transformation

The below table lists the properties supported by Azure Database for PostgreSQL source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Table	If you select Table as input, data flow fetches all the data from the table specified in the dataset.	No	-	*(for inline dataset only)* tableName

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Query	If you select Query as input, specify a SQL query to fetch data from source, which overrides any table you specify in dataset. Using queries is a great way to reduce rows for testing or lookups. **Order By** clause is not supported, but you can set a full SELECT FROM statement. You can also use user-defined table functions. **select * from udfGetData()** is a UDF in SQL that returns a table that you can use in data flow. Query example: `select * from mytable where customerId > 1000 and customerId < 2000` or `select * from "MyTable"`. Note in PostgreSQL, the entity name is treated as case-insensitive if not quoted.	No	String	query
Schema name	If you select Stored procedure as input, specify a schema name of the stored procedure, or select Refresh to ask the service to discover the schema names.	No	String	schemaName
Stored procedure	If you select Stored procedure as input, specify a name of the stored procedure to read data from the source table, or select Refresh to ask the service to discover the procedure names.	Yes (if you select Stored procedure as input)	String	procedureName

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Procedure parameters	If you select Stored procedure as input, specify any input parameters for the stored procedure in the order set in the procedure, or select Import to import all procedure parameters using the form `@paraName`.	No	Array	inputs
Batch size	Specify a batch size to chunk large data into batches.	No	Integer	batchSize
Isolation Level	Choose one of the following isolation levels: - Read Committed - Read Uncommitted (default) - Repeatable Read - Serializable - None (ignore isolation level)	No	READ_COMMITTED READ_UNCOMMITTED REPEATABLE_READ SERIALIZABLE NONE	isolationLevel

Azure Database for PostgreSQL source script example

When you use Azure Database for PostgreSQL as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
    validateSchema: false,
    isolationLevel: 'READ_UNCOMMITTED',
    query: 'select * from mytable',
    format: 'query') ~> AzurePostgreSQLSource
```

Sink transformation

The below table lists the properties supported by Azure Database for PostgreSQL sink. You can edit these properties in the **Sink options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Update method	Specify what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an Alter row transformation is required to tag rows for those actions.	Yes	`true` or `false`	deletable insertable updateable upsertable

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Key columns	For updates, upserts and deletes, key column(s) must be set to determine which row to alter. The column name that you pick as the key will be used as part of the subsequent update, upsert, delete. Therefore, you must pick a column that exists in the Sink mapping.	No	Array	keys
Skip writing key columns	If you wish to not write the value to the key column, select "Skip writing key columns".	No	`true` or `false`	skipKeyWrites
Table action	Determines whether to recreate or remove all rows from the destination table prior to writing. - **None**: No action will be done to the table. - **Recreate**: The table will get dropped and recreated. Required if creating a new table dynamically. - **Truncate**: All rows from the target table will get removed.	No	`true` or `false`	recreate truncate
Batch size	Specify how many rows are being written in each batch. Larger batch sizes improve compression and memory optimization, but risk out of memory exceptions when caching data.	No	Integer	batchSize

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Select user DB schema	By default, a temporary table will be created under the sink schema as staging. You can alternatively uncheck the **Use sink schema** option and instead, specify a schema name under which Data Factory will create a staging table to load upstream data and automatically clean them up upon completion. Make sure you have create table permission in the database and alter permission on the schema.	No	String	stagingSchemaName
Pre and Post SQL scripts	Specify multi-line SQL scripts that will execute before (pre-processing) and after (post-processing) data is written to your Sink database.	No	String	preSQLs postSQLs

TIP

1. It's recommended to break single batch scripts with multiple commands into multiple batches.
2. Only Data Definition Language (DDL) and Data Manipulation Language (DML) statements that return a simple update count can be run as part of a batch. Learn more from Performing batch operations

- Enable incremental extract: Use this option to tell ADF to only process rows that have changed since the last time that the pipeline executed.
- Incremental date column: When using the incremental extract feature, you must choose the date/time column that you wish to use as the watermark in your source table.
- Start reading from beginning: Setting this option with incremental extract will instruct ADF to read all rows on first execution of a pipeline with incremental extract turned on.

Azure Database for PostgreSQL sink script example

When you use Azure Database for PostgreSQL as sink type, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    deletable:false,
    insertable:true,
    updateable:true,
    upsertable:true,
    keys:['keyColumn'],
    format: 'table',
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> AzurePostgreSQLSink
```

Lookup activity properties

For more information about the properties, see Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data to and from Azure Databricks Delta Lake using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 14 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy activity in Azure Data Factory and Azure Synapse to copy data to and from Azure Databricks Delta Lake. It builds on the Copy activity article, which presents a general overview of copy activity.

Supported capabilities

This Azure Databricks Delta Lake connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Mapping data flow (source/sink)	①
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

In general, the service supports Delta Lake with the following capabilities to meet your various needs.

- Copy activity supports Azure Databricks Delta Lake connector to copy data from any supported source data store to Azure Databricks delta lake table, and from delta lake table to any supported sink data store. It leverages your Databricks cluster to perform the data movement, see details in Prerequisites section.
- Mapping Data Flow supports generic Delta format on Azure Storage as source and sink to read and write Delta files for code-free ETL, and runs on managed Azure Integration Runtime.
- Databricks activities supports orchestrating your code-centric ETL or machine learning workload on top of delta lake.

Prerequisites

To use this Azure Databricks Delta Lake connector, you need to set up a cluster in Azure Databricks.

- To copy data to delta lake, Copy activity invokes Azure Databricks cluster to read data from an Azure Storage, which is either your original source or a staging area to where the service firstly writes the source data via built-in staged copy. Learn more from Delta lake as the sink.
- Similarly, to copy data from delta lake, Copy activity invokes Azure Databricks cluster to write data to an Azure Storage, which is either your original sink or a staging area from where the service continues to write data to final sink via built-in staged copy. Learn more from Delta lake as the source.

The Databricks cluster needs to have access to Azure Blob or Azure Data Lake Storage Gen2 account, both the storage container/file system used for source/sink/staging and the container/file system where you want to write the Delta Lake tables.

- To use **Azure Data Lake Storage Gen2**, you can configure a **service principal** on the Databricks cluster as part of the Apache Spark configuration. Follow the steps in Access directly with service principal.
- To use **Azure Blob storage**, you can configure a **storage account access key** or **SAS token** on the Databricks cluster as part of the Apache Spark configuration. Follow the steps in Access Azure Blob storage using the RDD API.

During copy activity execution, if the cluster you configured has been terminated, the service automatically starts it. If you author pipeline using authoring UI, for operations like data preview, you need to have a live cluster, the service won't start the cluster on your behalf.

Specify the cluster configuration

1. In the **Cluster Mode** drop-down, select **Standard**.
2. In the **Databricks Runtime Version** drop-down, select a Databricks runtime version.
3. Turn on Auto Optimize by adding the following properties to your Spark configuration:

   ```
   spark.databricks.delta.optimizeWrite.enabled true
   spark.databricks.delta.autoCompact.enabled true
   ```

4. Configure your cluster depending on your integration and scaling needs.

For cluster configuration details, see Configure clusters.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

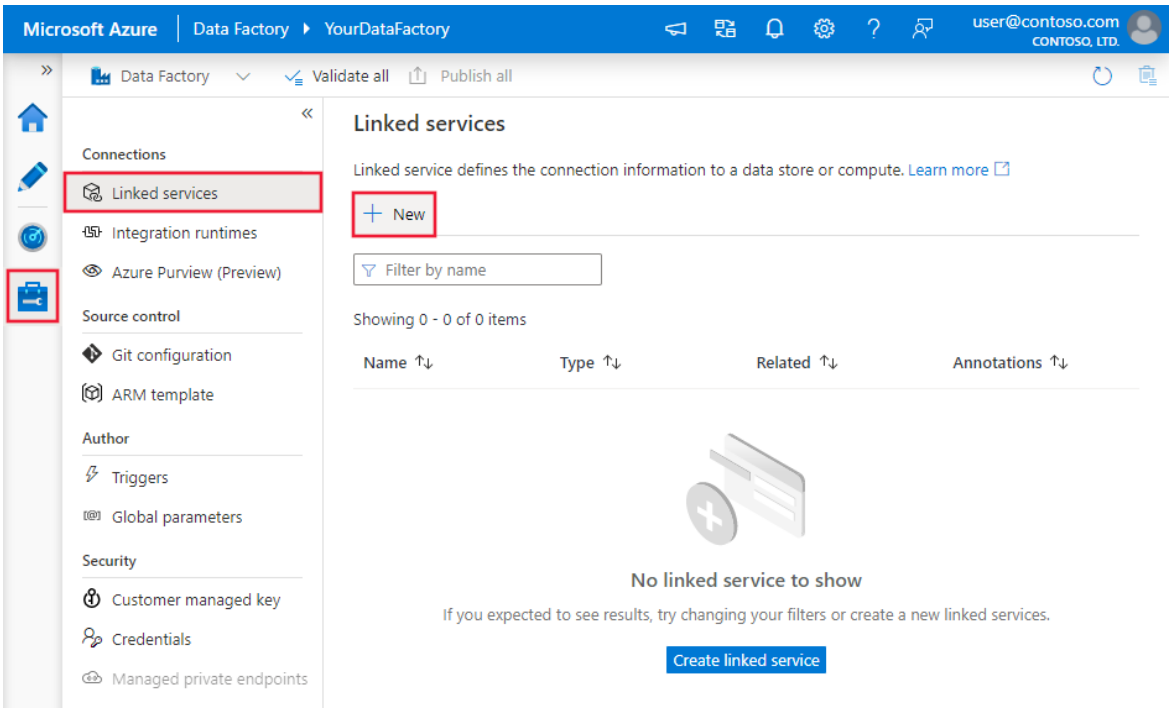
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Azure Databricks Delta Lake using UI

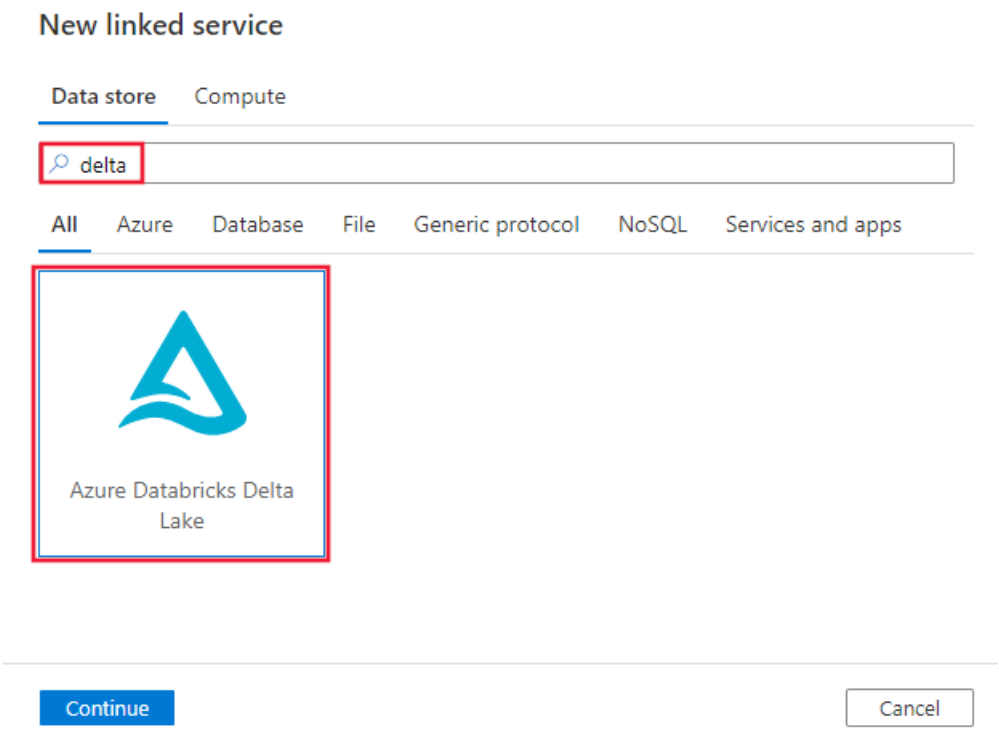
Use the following steps to create a linked service to Azure Databricks Delta Lake in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

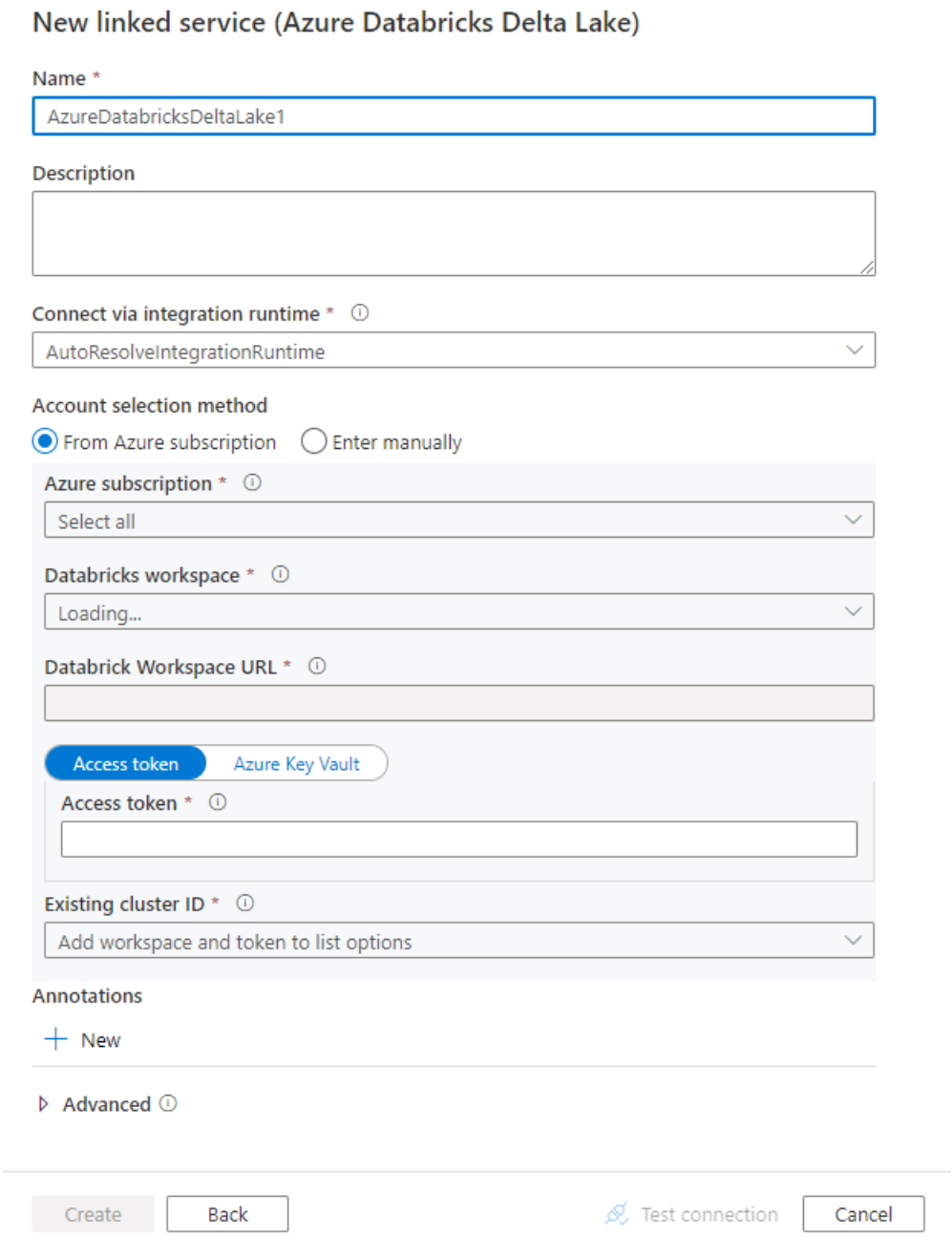
   - Azure Data Factory
   - Azure Synapse



2. Search for delta and select the Azure Databricks Delta Lake connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that define entities specific to an Azure Databricks Delta Lake connector.

Linked service properties

This Azure Databricks Delta Lake connector supports the following authentication types. See the corresponding sections for details.

- Access token
- System-assigned managed identity authentication
- User-assigned managed identity authentication

Access token

The following properties are supported for the Azure Databricks Delta Lake linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AzureDatabricksDeltaLake**.	Yes
domain	Specify the Azure Databricks workspace URL, e.g. `https://adb-xxxxxxxxx.xx.azuredatabricks.net`.	
clusterId	Specify the cluster ID of an existing cluster. It should be an already created Interactive Cluster. You can find the Cluster ID of an Interactive Cluster on Databricks workspace -> Clusters -> Interactive Cluster Name -> Configuration -> Tags. Learn more.	
accessToken	Access token is required for the service to authenticate to Azure Databricks. Access token needs to be generated from the databricks workspace. More detailed steps to find the access token can be found here.	
connectVia	The integration runtime that is used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime (if your data store is located in a private network). If not specified, it uses the default Azure integration runtime.	No

Example:

```
{
    "name": "AzureDatabricksDeltaLakeLinkedService",
    "properties": {
        "type": "AzureDatabricksDeltaLake",
        "typeProperties": {
            "domain": "https://adb-xxxxxxxxx.xx.azuredatabricks.net",
            "clusterId": "<cluster id>",
            "accessToken": {
                "type": "SecureString",
                "value": "<access token>"
           }
        }
    }
}
```

System-assigned managed identity authentication

To learn more about system-assigned managed identities for Azure resources, see system-assigned managed identity for Azure resources.

To use system-assigned managed identity authentication, follow these steps to grant permissions:

1. Retrieve the managed identity information by copying the value of the **managed identity object ID** generated along with your data factory or Synapse workspace.

2. Grant the managed identity the correct permissions in Azure Databricks. In general, you must grant at least the **Contributor** role to your system-assigned managed identity in **Access control (IAM)** of Azure Databricks.

The following properties are supported for the Azure Databricks Delta Lake linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AzureDatabricksDeltaLake**.	Yes
domain	Specify the Azure Databricks workspace URL, e.g. `https://adb-xxxxxxxxx.xx.azuredatabricks.net`.	Yes
clusterId	Specify the cluster ID of an existing cluster. It should be an already created Interactive Cluster. You can find the Cluster ID of an Interactive Cluster on Databricks workspace -> Clusters -> Interactive Cluster Name -> Configuration -> Tags. Learn more.	Yes
workspaceResourceId	Specify the workspace resource ID of your Azure Databricks.	Yes
connectVia	The integration runtime that is used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime (if your data store is located in a private network). If not specified, it uses the default Azure integration runtime.	No

Example:

```
{
    "name": "AzureDatabricksDeltaLakeLinkedService",
    "properties": {
        "type": "AzureDatabricksDeltaLake",
        "typeProperties": {
            "domain": "https://adb-xxxxxxxxx.xx.azuredatabricks.net",
            "clusterId": "<cluster id>",
            "workspaceResourceId": "<workspace resource id>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

User-assigned managed identity authentication

To learn more about user-assigned managed identities for Azure resources, see user-assigned managed identities

To use user-assigned managed identity authentication, follow these steps:

1. Create one or multiple user-assigned managed identities and grant permission in your Azure Databricks. In general, you must grant at least the **Contributor** role to your user-assigned managed identity in **Access control (IAM)** of Azure Databricks.
2. Assign one or multiple user-assigned managed identities to your data factory or Synapse workspace, and create credentials for each user-assigned managed identity.

The following properties are supported for the Azure Databricks Delta Lake linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AzureDatabricksDeltaLake**.	Yes
domain	Specify the Azure Databricks workspace URL, e.g. `https://adb-xxxxxxxxx.xx.azuredatabricks.net`.	Yes
clusterId	Specify the cluster ID of an existing cluster. It should be an already created Interactive Cluster. You can find the Cluster ID of an Interactive Cluster on Databricks workspace -> Clusters -> Interactive Cluster Name -> Configuration -> Tags. Learn more.	Yes
credentials	Specify the user-assigned managed identity as the credential object.	Yes
workspaceResourceId	Specify the workspace resource ID of your Azure Databricks.	Yes
connectVia	The integration runtime that is used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime (if your data store is located in a private network). If not specified, it uses the default Azure integration runtime.	No

Example:

```
{
    "name": "AzureDatabricksDeltaLakeLinkedService",
    "properties": {
        "type": "AzureDatabricksDeltaLake",
        "typeProperties": {
            "domain": "https://adb-xxxxxxxxx.xx.azuredatabricks.net",
            "clusterId": "<cluster id>",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            },
            "workspaceResourceId": "<workspace resource id>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article.

The following properties are supported for the Azure Databricks Delta Lake dataset.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **AzureDatabricksDeltaLakeDataset**.	Yes
database	Name of the database.	No for source, yes for sink
table	Name of the delta table.	No for source, yes for sink

Example:

```
{
    "name": "AzureDatabricksDeltaLakeDataset",
    "properties": {
        "type": "AzureDatabricksDeltaLakeDataset",
        "typeProperties": {
            "database": "<database name>",
            "table": "<delta table name>"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "linkedServiceName": {
            "referenceName": "<name of linked service>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the Azure Databricks Delta Lake source and sink.

Delta lake as source

To copy data from Azure Databricks Delta Lake, the following properties are supported in the Copy activity **source** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the Copy activity source must be set to **AzureDatabricksDeltaLakeSource**.	Yes

query	Specify the SQL query to read data. For the time travel control, follow the below pattern:
	-
	`SELECT * FROM events TIMESTAMP AS OF timestamp_expression`
	-
	`SELECT * FROM events VERSION AS OF version`
		No

exportSettings	Advanced settings used to retrieve data from delta table.	No
Under `exportSettings`*:*		
type	The type of export command, set to **AzureDatabricksDeltaLakeExportCommand**.	Yes
dateFormat	Format date type to string with a date format. Custom date formats follow the formats at datetime pattern. If not specified, it uses the default value `yyyy-MM-dd`.	No
timestampFormat	Format timestamp type to string with a timestamp format. Custom date formats follow the formats at datetime pattern. If not specified, it uses the default value `yyyy-MM-dd'T'HH:mm:ss[.SSS][XXX]`.	No

Direct copy from delta lake

If your sink data store and format meet the criteria described in this section, you can use the Copy activity to directly copy from Azure Databricks Delta table to sink. The service checks the settings and fails the Copy activity run if the following criteria is not met:

- The **sink linked service** is Azure Blob storage or Azure Data Lake Storage Gen2. The account credential should be pre-configured in Azure Databricks cluster configuration, learn more from Prerequisites.
- The **sink data format** is of **Parquet**, **delimited text**, or **Avro** with the following configurations, and points to a folder instead of file.
  - For **Parquet** format, the compression codec is **none**, **snappy**, or **gzip**.
  - For **delimited text** format:
    - `rowDelimiter` is any single character.
    - `compression` can be **none**, **bzip2**, **gzip**.

    - `encodingName` UTF-7 is not supported.
  - For **Avro** format, the compression codec is **none**, **deflate**, or **snappy**.
- In the Copy activity source, `additionalColumns` is not specified.
- If copying data to delimited text, in copy activity sink, `fileExtension` need to be ".csv".
- In the Copy activity mapping, type conversion is not enabled.

Example:

```
"activities":[
    {
        "name": "CopyFromDeltaLake",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delta lake input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AzureDatabricksDeltaLakeSource",
                "sqlReaderQuery": "SELECT * FROM events TIMESTAMP AS OF timestamp_expression"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Staged copy from delta lake

When your sink data store or format does not match the direct copy criteria, as mentioned in the last section, enable the built-in staged copy using an interim Azure storage instance. The staged copy feature also provides you better throughput. The service exports data from Azure Databricks Delta Lake into staging storage, then copies the data to sink, and finally cleans up your temporary data from the staging storage. See Staged copy for details about copying data by using staging.

To use this feature, create an Azure Blob storage linked service or Azure Data Lake Storage Gen2 linked service that refers to the storage account as the interim staging. Then specify the `enableStaging` and `stagingSettings` properties in the Copy activity.

> **NOTE**
> The staging storage account credential should be pre-configured in Azure Databricks cluster configuration, learn more from Prerequisites.

Example:

```
"activities":[
    {
        "name": "CopyFromDeltaLake",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delta lake input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AzureDatabricksDeltaLakeSource",
                "sqlReaderQuery": "SELECT * FROM events TIMESTAMP AS OF timestamp_expression"
            },
            "sink": {
                "type": "<sink type>"
            },
            "enableStaging": true,
            "stagingSettings": {
                "linkedServiceName": {
                    "referenceName": "MyStagingStorage",
                    "type": "LinkedServiceReference"
                },
                "path": "mystagingpath"
            }
        }
    }
]
```

Delta lake as sink

To copy data to Azure Databricks Delta Lake, the following properties are supported in the Copy activity **sink** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the Copy activity sink, set to **AzureDatabricksDeltaLakeSink**.	Yes
preCopyScript	Specify a SQL query for the Copy activity to run before writing data into Databricks delta table in each run. Example : `VACUUM eventsTable DRY RUN` You can use this property to clean up the preloaded data, or add a truncate table or Vacuum statement.	No
importSettings	Advanced settings used to write data into delta table.	No
Under `importSettings`*:*		

PROPERTY	DESCRIPTION	REQUIRED
type	The type of import command, set to **AzureDatabricksDeltaLakeImport Command**.	Yes
dateFormat	Format string to date type with a date format. Custom date formats follow the formats at datetime pattern. If not specified, it uses the default value `yyyy-MM-dd`.	No
timestampFormat	Format string to timestamp type with a timestamp format. Custom date formats follow the formats at datetime pattern. If not specified, it uses the default value `yyyy-MM-dd'T'HH:mm:ss[.SSS][XXX]`.	No

Direct copy to delta lake

If your source data store and format meet the criteria described in this section, you can use the Copy activity to directly copy from source to Azure Databricks Delta Lake. The service checks the settings and fails the Copy activity run if the following criteria is not met:

- The **source linked service** is Azure Blob storage or Azure Data Lake Storage Gen2. The account credential should be pre-configured in Azure Databricks cluster configuration, learn more from Prerequisites.
- The **source data format** is of **Parquet**, **delimited text**, or **Avro** with the following configurations, and points to a folder instead of file.
  - For **Parquet** format, the compression codec is **none**, **snappy**, or **gzip**.
  - For **delimited text** format:
    - `rowDelimiter` is default, or any single character.
    - `compression` can be **none**, **bzip2**, **gzip**.
    - `encodingName` UTF-7 is not supported.
  - For **Avro** format, the compression codec is **none**, **deflate**, or **snappy**.
- In the Copy activity source:
  - `wildcardFileName` only contains wildcard `*` but not `?`, and `wildcardFolderName` is not specified.
  - `prefix`, `modifiedDateTimeStart`, `modifiedDateTimeEnd`, and `enablePartitionDiscovery` are not specified.
  - `additionalColumns` is not specified.
- In the Copy activity mapping, type conversion is not enabled.

Example:

```
"activities":[
    {
        "name": "CopyToDeltaLake",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Delta lake output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureDatabricksDeltaLakeSink",
                "sqlReadrQuery": "VACUUM eventsTable DRY RUN"
            }
        }
    }
]
```

Staged copy to delta lake

When your source data store or format does not match the direct copy criteria, as mentioned in the last section, enable the built-in staged copy using an interim Azure storage instance. The staged copy feature also provides you better throughput. The service automatically converts the data to meet the data format requirements into staging storage, then load data into delta lake from there. Finally, it cleans up your temporary data from the storage. See Staged copy for details about copying data using staging.

To use this feature, create an Azure Blob storage linked service or Azure Data Lake Storage Gen2 linked service that refers to the storage account as the interim staging. Then specify the `enableStaging` and `stagingSettings` properties in the Copy activity.

> **NOTE**
>
> The staging storage account credential should be pre-configured in Azure Databricks cluster configuration, learn more from Prerequisites.

Example:

```
"activities":[
    {
        "name": "CopyToDeltaLake",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Delta lake output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureDatabricksDeltaLakeSink"
            },
            "enableStaging": true,
            "stagingSettings": {
                "linkedServiceName": {
                    "referenceName": "MyStagingBlob",
                    "type": "LinkedServiceReference"
                },
                "path": "mystagingpath"
            }
        }
    }
]
```

Monitoring

The same copy activity monitoring experience is provided as for other connectors. In addition, because loading data from/to delta lake is running on your Azure Databricks cluster, you can further view detailed cluster logs and monitor performance.

Lookup activity properties

For more information about the properties, see Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by Copy activity, see supported data stores and formats.

Copy data from or to Azure Files by using Azure Data Factory

8/19/2022 • 20 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to copy data to and from Azure Files. To learn about Azure Data Factory, read the introductory article.

Supported capabilities

This Azure Files connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/sink)	① ②	✓ Exclude storage account V1
Lookup activity	① ②	✓ Exclude storage account V1
GetMetadata activity	① ②	✓ Exclude storage account V1
Delete activity	① ②	✓ Exclude storage account V1

① Azure integration runtime ② Self-hosted integration runtime

You can copy data from Azure Files to any supported sink data store, or copy data from any supported source data store to Azure Files. For a list of data stores that Copy Activity supports as sources and sinks, see Supported data stores and formats.

Specifically, this Azure Files connector supports:

- Copying files by using account key or service shared access signature (SAS) authentications.
- Copying files as-is or parsing/generating files with the supported file formats and compression codecs.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

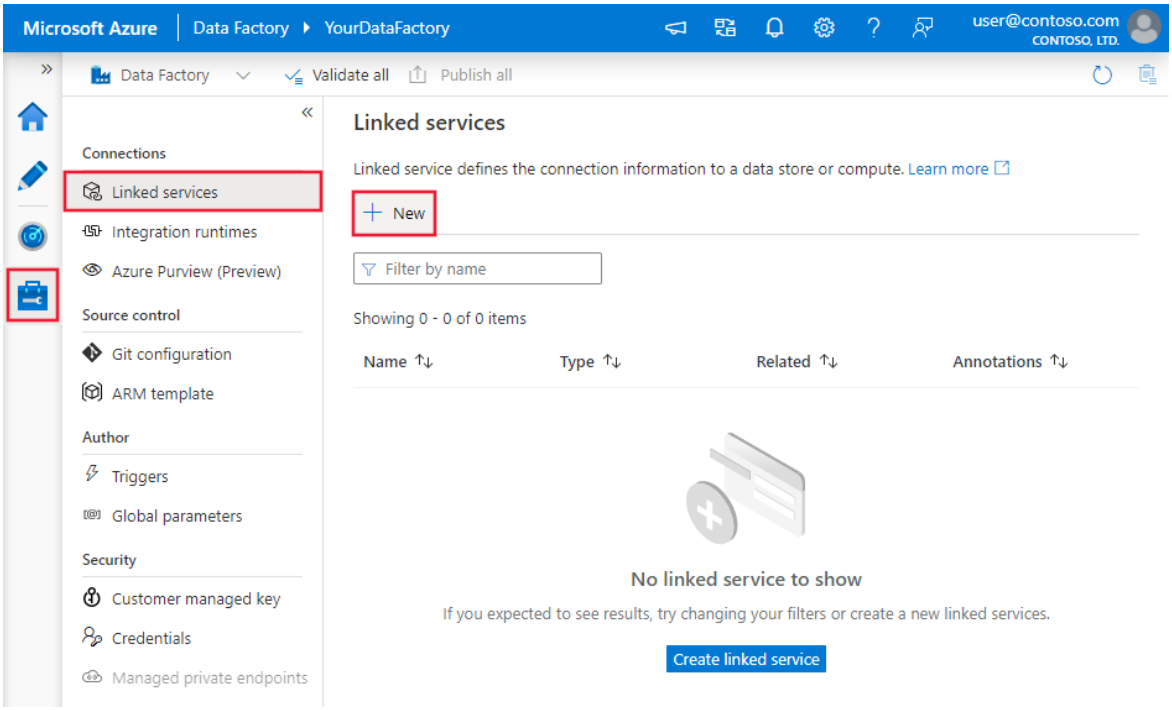
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Azure Files using UI

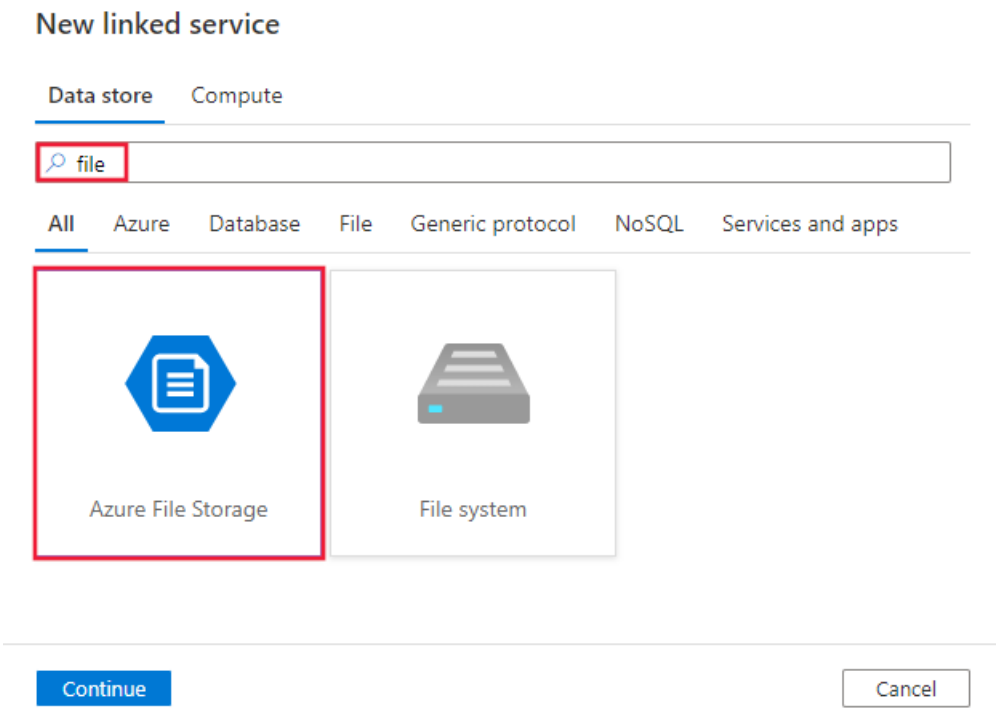
Use the following steps to create a linked service to Azure Files in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

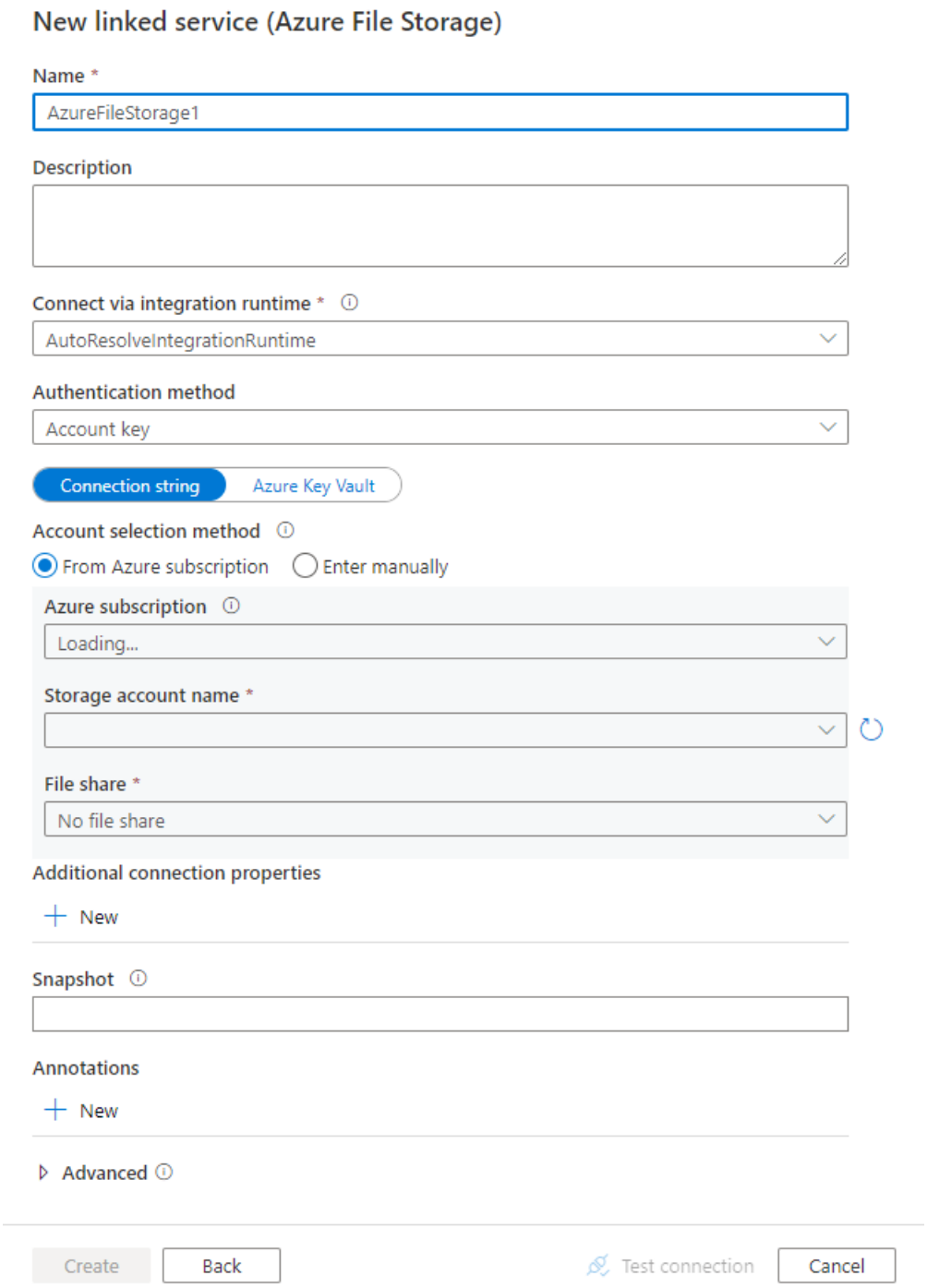
   - Azure Data Factory
   - Azure Synapse



2. Search for file and select the connector for Azure Files labeled *Azure File Storage*.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to Azure Files.

Linked service properties

The Azure Files connector supports the following authentication types. See the corresponding sections for details.

- Account key authentication
- Shared access signature authentication

> **NOTE**
>
> If you were using Azure Files linked service with legacy model, where on ADF authoring UI shown as "Basic authentication", it is still supported as-is, while you are suggested to use the new model going forward. The legacy model transfers data from/to storage over Server Message Block (SMB), while the new model utilizes the storage SDK which has better throughput. To upgrade, you can edit your linked service to switch the authentication method to "Account key" or "SAS URI"; no change needed on dataset or copy activity.

Account key authentication

Data Factory supports the following properties for Azure Files account key authentication:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **AzureFileStorage**.	Yes
connectionString	Specify the information needed to connect to Azure Files. You can also put the account key in Azure Key Vault and pull the `accountKey` configuration out of the connection string. For more information, see the following samples and the Store credentials in Azure Key Vault article.	Yes
fileShare	Specify the file share.	Yes
snapshot	Specify the date of the file share snapshot if you want to copy from a snapshot.	No
connectVia	The Integration Runtime to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is located in private network). If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "AzureFileStorageLinkedService",
    "properties": {
        "type": "AzureFileStorage",
        "typeProperties": {
            "connectionString": "DefaultEndpointsProtocol=https;AccountName=<accountName>;AccountKey=
<accountKey>;EndpointSuffix=core.windows.net;",
            "fileShare": "<file share name>"
        },
        "connectVia": {
          "referenceName": "<name of Integration Runtime>",
          "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store the account key in Azure Key Vault

```
{
    "name": "AzureFileStorageLinkedService",
    "properties": {
        "type": "AzureFileStorage",
        "typeProperties": {
            "connectionString": "DefaultEndpointsProtocol=https;AccountName=<accountname>;",
            "fileShare": "<file share name>",
            "accountKey": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Shared access signature authentication

A shared access signature provides delegated access to resources in your storage account. You can use a shared access signature to grant a client limited permissions to objects in your storage account for a specified time. For more information about shared access signatures, see Shared access signatures: Understand the shared access signature model.

The service supports the following properties for using shared access signature authentication:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **AzureFileStorage**.	Yes
sasUri	Specify the shared access signature URI to the resources. Mark this field as **SecureString** to store it securely. You can also put the SAS token in Azure Key Vault to use auto-rotation and remove the token portion. For more information, see the following samples and Store credentials in Azure Key Vault.	Yes
fileShare	Specify the file share.	Yes
snapshot	Specify the date of the file share snapshot if you want to copy from a snapshot.	No

PROPERTY	DESCRIPTION	REQUIRED
connectVia	The Integration Runtime to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is located in private network). If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "AzureFileStorageLinkedService",
    "properties": {
        "type": "AzureFileStorage",
        "typeProperties": {
            "sasUri": {
                "type": "SecureString",
                "value": "<SAS URI of the resource e.g. https://<accountname>.file.core.windows.net/?sv=
<storage version>&st=<start time>&se=<expire time>&sr=<resource>&sp=<permissions>&sip=<ip range>&spr=
<protocol>&sig=<signature>>"
            },
            "fileShare": "<file share name>",
            "snapshot": "<snapshot version>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store the SAS token in Azure Key Vault

```
{
    "name": "AzureFileStorageLinkedService",
    "properties": {
        "type": "AzureFileStorage",
        "typeProperties": {
            "sasUri": {
                "type": "SecureString",
                "value": "<SAS URI of the Azure Storage resource without token e.g.
https://<accountname>.file.core.windows.net/>"
            },
            "sasToken": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName with value of SAS token e.g. ?sv=<storage version>&st=<start
time>&se=<expire time>&sr=<resource>&sp=<permissions>&sip=<ip range>&spr=<protocol>&sig=<signature>>"
            },
            "fileShare": "<file share name>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Legacy model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **AzureFileStorage**.	Yes

host	Specifies the Azure Files endpoint as:
	-Using UI: specify
	`\\<storage name>.file.core.windows.net\<file service name>`
	- Using JSON:
	`"host": "\\\\<storage name>.file.core.windows.net\\<file service name>"`
	.
		Yes

userid	Specify the user to access the Azure Files as:
	-Using UI: specify
	`AZURE\<storage name>`
	-Using JSON:
	`"userid": "AZURE\\<storage name>"`
	.
		Yes

password	Specify the storage access key. Mark this field as a SecureString to store it securely in Data Factory, or reference a secret stored in Azure Key Vault.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is located in private network). If not specified, it uses the default Azure Integration Runtime.	No for source, Yes for sink

Example:

```
{
    "name": "AzureFileStorageLinkedService",
    "properties": {
        "type": "AzureFileStorage",
        "typeProperties": {
            "host": "\\\\<storage name>.file.core.windows.net\\<file service name>",
            "userid": "AZURE\\<storage name>",
            "password": {
                "type": "SecureString",
                "value": "<storage access key>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Azure Files under `location` settings in format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `location` in dataset must be set to **AzureFileStorageLocation**.	Yes
folderPath	The path to folder. If you want to use wildcard to filter folder, skip this setting and specify in activity source settings.	No
fileName	The file name under the given folderPath. If you want to use wildcard to filter files, skip this setting and specify in activity source settings.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<Azure File Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AzureFileStorageLocation",
                "folderPath": "root/folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Azure Files source and sink.

Azure Files as source

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Azure Files under `storeSettings` settings in format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `storeSettings` must be set to **AzureFileStorageReadSettings**.	Yes
Locate the files to copy:		
OPTION 1: static path	Copy from the given folder/file path specified in the dataset. If you want to copy all files from a folder, additionally specify `wildcardFileName` as `*`.	
OPTION 2: file prefix - prefix	Prefix for the file name under the given file share configured in a dataset to filter source files. Files with name starting with `fileshare_in_linked_service/this_prefix` are selected. It utilizes the service-side filter for Azure Files, which provides better performance than a wildcard filter. This feature is not supported when using a legacy linked service model.	No
OPTION 3: wildcard - wildcardFolderPath	The folder path with wildcard characters to filter source folders. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual folder name has wildcard or this escape char inside. See more examples in Folder and file filter examples.	No

PROPERTY	DESCRIPTION	REQUIRED
OPTION 3: wildcard - wildcardFileName	The file name with wildcard characters under the given folderPath/wildcardFolderPath to filter source files. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual file name has wildcard or this escape char inside. See more examples in Folder and file filter examples.	Yes
OPTION 4: a list of files - fileListPath	Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset. When using this option, do not specify file name in dataset. See more examples in File list examples.	No
Additional settings:		
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. When recursive is set to true and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**. This property doesn't apply when you configure `fileListPath`.	No
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	Files filter based on the attribute: Last Modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z". The properties can be NULL, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal with the datetime value will be selected. When `modifiedDatetimeEnd` has datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value will be selected. This property doesn't apply when you configure `fileListPath`.	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromAzureFileStorage",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "AzureFileStorageReadSettings",
                    "recursive": true,
                    "wildcardFolderPath": "myfolder*A",
                    "wildcardFileName": "*.csv"
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Azure Files as sink

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- JSON format
- ORC format
- Parquet format

The following properties are supported for Azure Files under `storeSettings` settings in format-based copy sink:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `storeSettings` must be set to **AzureFileStorageWriteSettings**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
copyBehavior	Defines the copy behavior when the source is files from a file-based data store. Allowed values are: - **PreserveHierarchy (default)**: Preserves the file hierarchy in the target folder. The relative path of source file to source folder is identical to the relative path of target file to target folder. - **FlattenHierarchy**: All files from the source folder are in the first level of the target folder. The target files have autogenerated names. - **MergeFiles**: Merges all files from the source folder to one file. If the file name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToAzureFileStorage",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Parquet output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "ParquetSink",
                "storeSettings":{
                    "type": "AzureFileStorageWriteSettings",
                    "copyBehavior": "PreserveHierarchy"
                }
            }
        }
    }
]
```

Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`Folder*`	(empty, use default)	false	FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	(empty, use default)	true	FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv
`Folder*`	`*.csv`	false	FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	`*.csv`	true	FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This section describes the resulting behavior of using file list path in copy activity source.

Assuming you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	CONFIGURATION
root FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In dataset:** - Folder path: `root/FolderA` **In copy activity source:** - File list path: `root/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line with the relative path to the path configured in the dataset.

recursive and copyBehavior examples

This section describes the resulting behavior of the Copy operation for different combinations of recursive and copyBehavior values.

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET
true	preserveHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder Folder1 is created with the same structure as the source: Folder1 File1 File2 Subfolder1 File3 File4 File5.
true	flattenHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the following structure: Folder1 autogenerated name for File1 autogenerated name for File2 autogenerated name for File3 autogenerated name for File4 autogenerated name for File5
true	mergeFiles	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the following structure: Folder1 File1 + File2 + File3 + File4 + File 5 contents are merged into one file with autogenerated file name

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET

false	preserveHierarchy	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target folder Folder1 is created with the following structure
			
			Folder1
			  File1
			  File2
			
			Subfolder1 with File3, File4, and File5 are not picked up.

false	flattenHierarchy	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target folder Folder1 is created with the following structure
			
			Folder1
			  autogenerated name for File1
			  autogenerated name for File2
			
			Subfolder1 with File3, File4, and File5 are not picked up.

false	mergeFiles	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target folder Folder1 is created with the following structure
			
			Folder1
			  File1 + File2 contents are merged into one file with autogenerated file name. autogenerated name for File1
			
			Subfolder1 with File3, File4, and File5 are not picked up.

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity

Delete activity properties

To learn details about the properties, check Delete activity

Legacy models

> **NOTE**
>
> The following models are still supported as-is for backward compatibility. You are suggested to use the new model mentioned in above sections going forward, and the authoring UI has switched to generating the new model.

Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **FileShare**	Yes
folderPath	Path to the folder. Wildcard filter is supported, allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual folder name has wildcard or this escape char inside. Examples: rootfolder/subfolder/, see more examples in Folder and file filter examples.	Yes
fileName	**Name or wildcard filter** for the file(s) under the specified "folderPath". If you don't specify a value for this property, the dataset points to all files in the folder. For filter, allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). - Example 1: `"fileName": "*.csv"` - Example 2: `"fileName": "???20180427.txt"` Use `^` to escape if your actual file name has wildcard or this escape char inside. When fileName isn't specified for an output dataset and **preserveHierarchy** isn't specified in the activity sink, the copy activity automatically generates the file name with the following pattern: "*Data.[activity run ID GUID].[GUID if FlattenHierarchy].[format if configured].[compression if configured]*", for example "Data.0a405f8a-93ff-4c6f-b3be-f69616f1df7a.txt.gz"; if you copy from tabular source using table name instead of query, the name pattern is "*[table name].[format].[compression if configured]*", for example "MyTable.csv".	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	Files filter based on the attribute: Last Modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z". Be aware the overall performance of data movement will be impacted by enabling this setting when you want to do file filter from huge amounts of files. The properties can be NULL that means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal with the datetime value will be selected. When `modifiedDatetimeEnd` has datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value will be selected.	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeEnd	Files filter based on the attribute: Last Modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z". Be aware the overall performance of data movement will be impacted by enabling this setting when you want to do file filter from huge amounts of files. The properties can be NULL that means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal with the datetime value will be selected. When `modifiedDatetimeEnd` has datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value will be selected.	No
format	If you want to **copy files as-is** between file-based stores (binary copy), skip the format section in both input and output dataset definitions. If you want to parse or generate files with a specific format, the following file format types are supported: **TextFormat**, **JsonFormat**, **AvroFormat**, **OrcFormat**, **ParquetFormat**. Set the **type** property under format to one of these values. For more information, see Text Format, Json Format, Avro Format, Orc Format, and Parquet Format sections.	No (only for binary copy scenario)
compression	Specify the type and level of compression for the data. For more information, see Supported file formats and compression codecs. Supported types are: **GZip**, **Deflate**, **BZip2**, and **ZipDeflate**. Supported levels are: **Optimal** and **Fastest**.	No

TIP

To copy all files under a folder, specify **folderPath** only.
To copy a single file with a given name, specify **folderPath** with folder part and **fileName** with file name.
To copy a subset of files under a folder, specify **folderPath** with folder part and **fileName** with wildcard filter.

NOTE

If you were using "fileFilter" property for file filter, it is still supported as-is, while you are suggested to use the new filter capability added to "fileName" going forward.

Example:

```
{
    "name": "AzureFileStorageDataset",
    "properties": {
        "type": "FileShare",
        "linkedServiceName":{
            "referenceName": "<Azure File Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "folderPath": "folder/subfolder/",
            "fileName": "*",
            "modifiedDatetimeStart": "2018-12-01T05:00:00Z",
            "modifiedDatetimeEnd": "2018-12-01T06:00:00Z",
            "format": {
                "type": "TextFormat",
                "columnDelimiter": ",",
                "rowDelimiter": "\n"
            },
            "compression": {
                "type": "GZip",
                "level": "Optimal"
            }
        }
    }
}
```

Legacy copy activity source model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **FileSystemSource**	Yes
recursive	Indicates whether the data is read recursively from the sub folders or only from the specified folder. Note when recursive is set to true and sink is file-based store, empty folder/sub-folder will not be copied/created at sink. Allowed values are: **true** (default), **false**	No

PROPERTY	DESCRIPTION	REQUIRED
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromAzureFileStorage",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Azure File Storage input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "FileSystemSource",
                "recursive": true
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Legacy copy activity sink model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to: **FileSystemSink**	Yes

PROPERTY	DESCRIPTION	REQUIRED
copyBehavior	Defines the copy behavior when the source is files from file-based data store. Allowed values are: - **PreserveHierarchy (default)**: preserves the file hierarchy in the target folder. The relative path of source file to source folder is identical to the relative path of target file to target folder. - **FlattenHierarchy**: all files from the source folder are in the first level of target folder. The target files have autogenerated name. - **MergeFiles**: merges all files from the source folder to one file. If the File Name is specified, the merged file name would be the specified name; otherwise, would be autogenerated file name.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToAzureFileStorage",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Azure File Storage output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "FileSystemSink",
                "copyBehavior": "PreserveHierarchy"
            }
        }
    }
]
```

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy and transform data in Azure SQL Database by using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 32 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory or Azure Synapse pipelines to copy data from and to Azure SQL Database, and use Data Flow to transform data in Azure SQL Database. To learn more, read the introductory article for Azure Data Factory or Azure Synapse Analytics.

Supported capabilities

This Azure SQL Database connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/sink)	① ②	✓
Mapping data flow (source/sink)	①	✓
Lookup activity	① ②	✓
GetMetadata activity	① ②	✓
Script activity	① ②	✓
Stored procedure activity	① ②	✓

① Azure integration runtime ② Self-hosted integration runtime

For Copy activity, this Azure SQL Database connector supports these functions:

- Copying data by using SQL authentication and Azure Active Directory (Azure AD) Application token authentication with a service principal or managed identities for Azure resources.
- As a source, retrieving data by using a SQL query or a stored procedure. You can also choose to parallel copy from an Azure SQL Database source, see the Parallel copy from SQL database section for details.
- As a sink, automatically creating destination table if not exists based on the source schema; appending data to a table or invoking a stored procedure with custom logic during the copy.

If you use Azure SQL Database serverless tier, note when the server is paused, activity run fails instead of waiting for the auto resume to be ready. You can add activity retry or chain additional activities to make sure the server is live upon the actual execution.

IMPORTANT

If you copy data by using the Azure integration runtime, configure a server-level firewall rule so that Azure services can access the server. If you copy data by using a self-hosted integration runtime, configure the firewall to allow the appropriate IP range. This range includes the machine's IP that's used to connect to Azure SQL Database.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

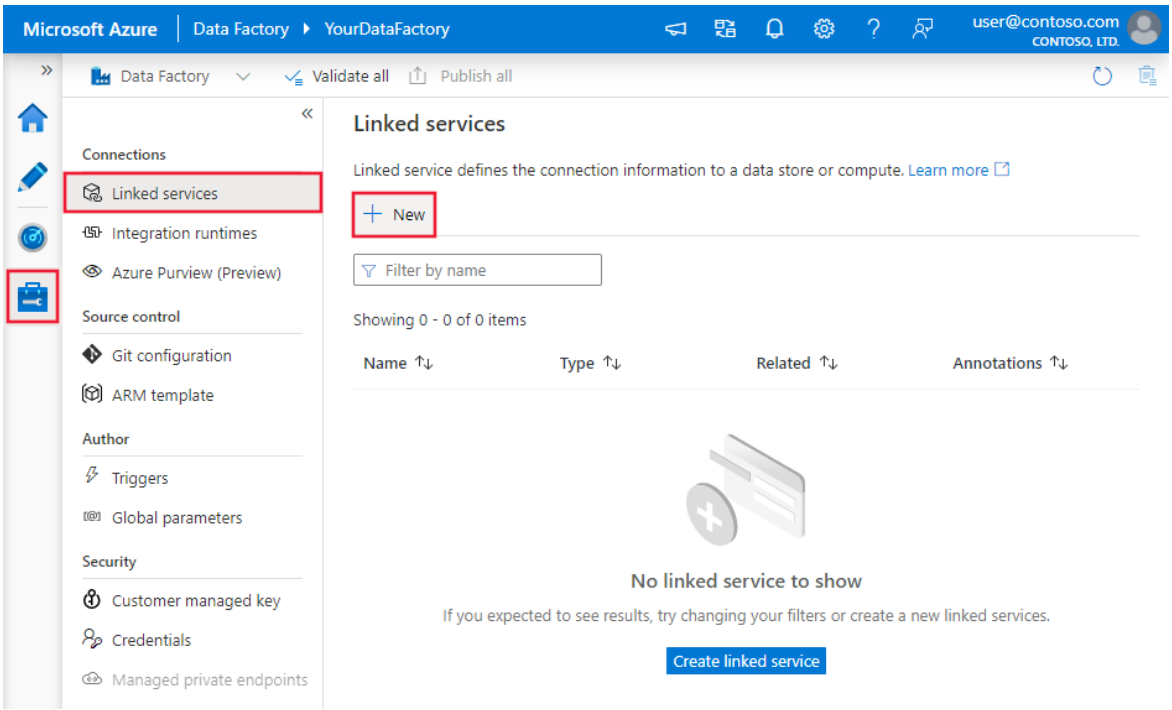
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create an Azure SQL Database linked service using UI

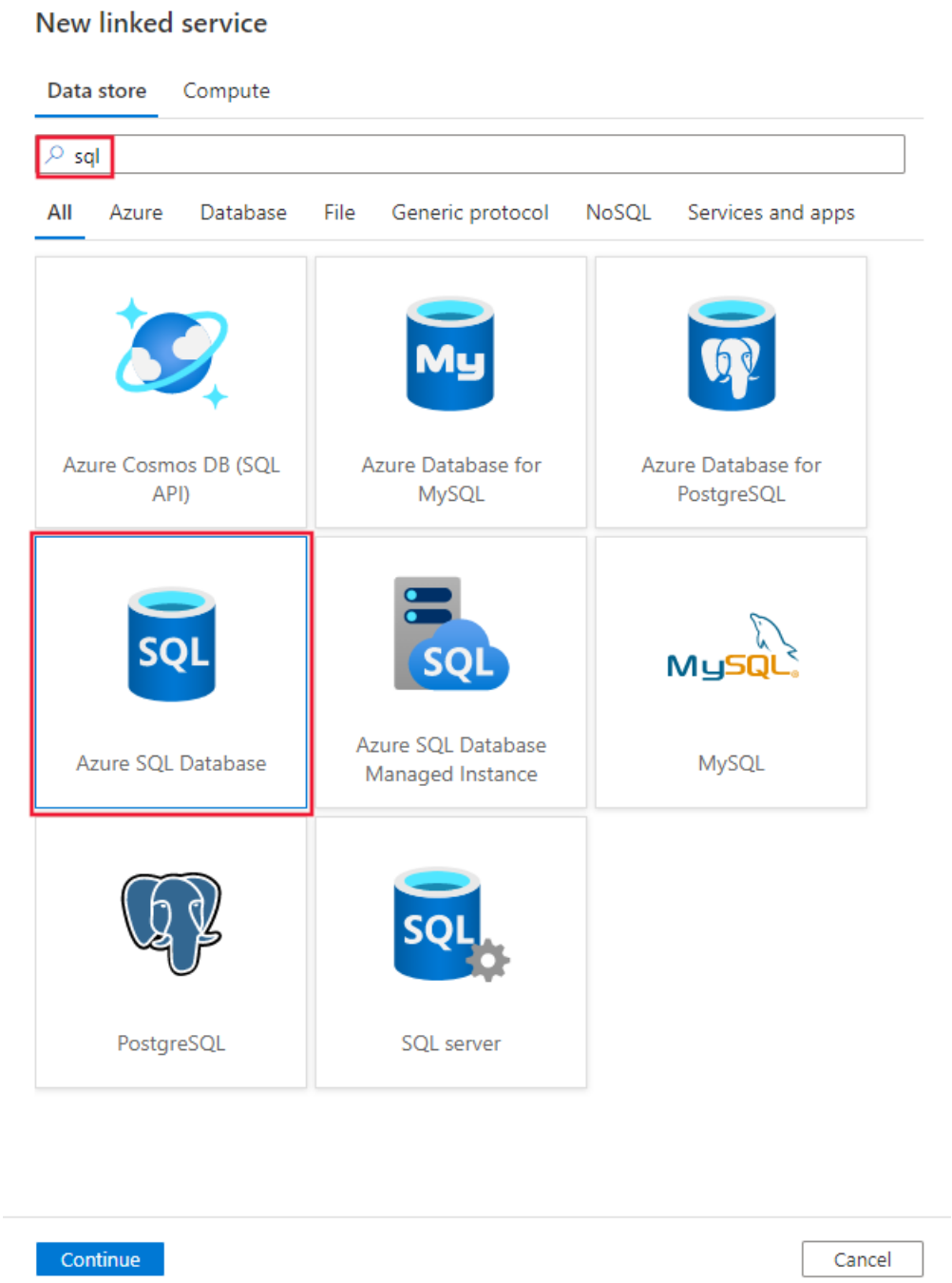
Use the following steps to create an Azure SQL Database linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

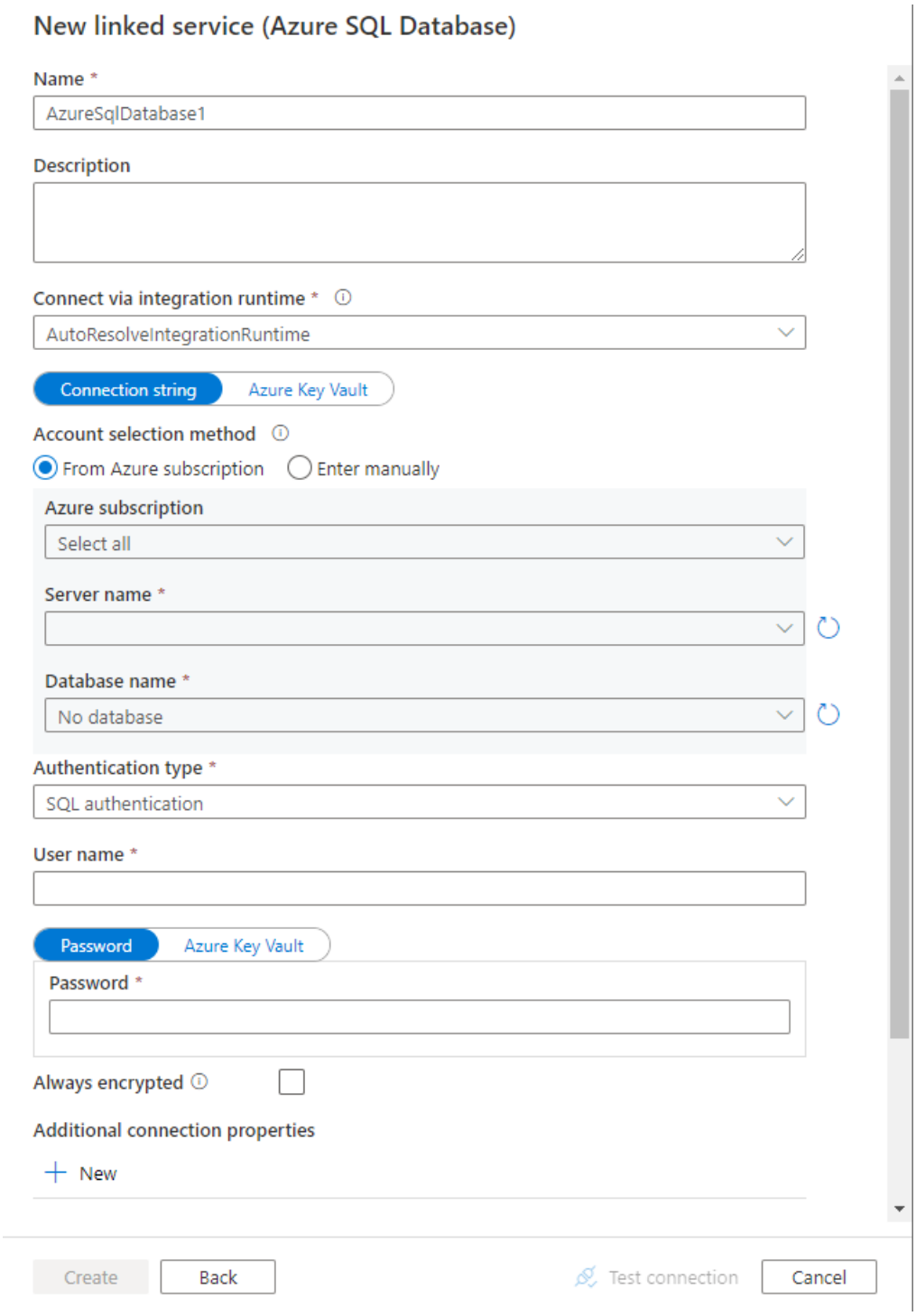
   - Azure Data Factory
   - Azure Synapse



2. Search for SQL and select the Azure SQL Database connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Azure Data Factory or Synapse pipeline entities specific to an Azure SQL Database connector.

Linked service properties

These generic properties are supported for an Azure SQL Database linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **AzureSqlDatabase**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	Specify information needed to connect to the Azure SQL Database instance for the **connectionString** property. You also can put a password or service principal key in Azure Key Vault. If it's SQL authentication, pull the `password` configuration out of the connection string. For more information, see the JSON example following the table and Store credentials in Azure Key Vault.	Yes
azureCloudType	For service principal authentication, specify the type of Azure cloud environment to which your Azure AD application is registered. Allowed values are **AzurePublic**, **AzureChina**, **AzureUsGovernment**, and **AzureGermany**. By default, the data factory or Synapse pipeline's cloud environment is used.	No
alwaysEncryptedSettings	Specify **alwaysencryptedsettings** information that's needed to enable Always Encrypted to protect sensitive data stored in SQL server by using either managed identity or service principal. For more information, see the JSON example following the table and Using Always Encrypted section. If not specified, the default always encrypted setting is disabled.	No
connectVia	This integration runtime is used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is located in a private network. If not specified, the default Azure integration runtime is used.	No

For different authentication types, refer to the following sections on specific properties, prerequisites and JSON samples, respectively:

- SQL authentication
- Service principal authentication
- System-assigned managed identity authentication
- User-assigned managed identity authentication

TIP

If you hit an error with the error code "UserErrorFailedToConnectToSqlServer" and a message like "The session limit for the database is XXX and has been reached," add `Pooling=false` to your connection string and try again. `Pooling=false` is also recommended for **SHIR(Self Hosted Integration Runtime)** type linked service setup. Pooling and other connection parameters can be added as new parameter names and values in **Additional connection properties** section of linked service creation form.

SQL authentication

To use SQL authentication authentication type, specify the generic properties that are described in the preceding section.

Example: using SQL authentication

```
{
    "name": "AzureSqlDbLinkedService",
    "properties": {
        "type": "AzureSqlDatabase",
        "typeProperties": {
            "connectionString": "Data Source=tcp:<servername>.database.windows.net,1433;Initial Catalog=
<databasename>;User ID=<username>@<servername>;Password=
<password>;Trusted_Connection=False;Encrypt=True;Connection Timeout=30"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: password in Azure Key Vault

```
{
    "name": "AzureSqlDbLinkedService",
    "properties": {
        "type": "AzureSqlDatabase",
        "typeProperties": {
            "connectionString": "Data Source=tcp:<servername>.database.windows.net,1433;Initial Catalog=
<databasename>;User ID=<username>@<servername>;Trusted_Connection=False;Encrypt=True;Connection Timeout=30",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Use Always Encrypted

```
{
    "name": "AzureSqlDbLinkedService",
    "properties": {
        "type": "AzureSqlDatabase",
        "typeProperties": {
            "connectionString": "Data Source=tcp:<servername>.database.windows.net,1433;Initial Catalog=
<databasename>;User ID=<username>@<servername>;Password=
<password>;Trusted_Connection=False;Encrypt=True;Connection Timeout=30"
        },
        "alwaysEncryptedSettings": {
            "alwaysEncryptedAkvAuthType": "ServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Service principal authentication

To use service principal authentication, in addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key. Mark this field as **SecureString** to store it securely or reference a secret stored in Azure Key Vault.	Yes
tenant	Specify the tenant information, like the domain name or tenant ID, under which your application resides. Retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Yes

You also need to follow the steps below:

1. Create an Azure Active Directory application from the Azure portal. Make note of the application name and the following values that define the linked service:
   - Application ID
   - Application key
   - Tenant ID
2. Provision an Azure Active Directory administrator for your server on the Azure portal if you haven't already done so. The Azure AD administrator must be an Azure AD user or Azure AD group, but it can't be a service principal. This step is done so that, in the next step, you can use an Azure AD identity to create a contained database user for the service principal.
3. Create contained database users for the service principal. Connect to the database from or to which you want to copy data by using tools like SQL Server Management Studio, with an Azure AD identity that has at least ALTER ANY USER permission. Run the following T-SQL:

```
CREATE USER [your application name] FROM EXTERNAL PROVIDER;
```

4. Grant the service principal needed permissions as you normally do for SQL users or others. Run the following code. For more options, see this document.

```
ALTER ROLE [role name] ADD MEMBER [your application name];
```

5. Configure an Azure SQL Database linked service in an Azure Data Factory or Synapse workspace.

Linked service example that uses service principal authentication

```
{
    "name": "AzureSqlDbLinkedService",
    "properties": {
        "type": "AzureSqlDatabase",
        "typeProperties": {
            "connectionString": "Data Source=tcp:<servername>.database.windows.net,1433;Initial Catalog=
<databasename>;Connection Timeout=30",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

System-assigned managed identity authentication

A data factory or Synapse workspace can be associated with a system-assigned managed identity for Azure resources that represents the service when authenticating to other resources in Azure. You can use this managed identity for Azure SQL Database authentication. The designated factory or Synapse workspace can access and copy data from or to your database by using this identity.

To use system-assigned managed identity authentication, specify the generic properties that are described in the preceding section, and follow these steps.

1. Provision an Azure Active Directory administrator for your server on the Azure portal if you haven't already done so. The Azure AD administrator can be an Azure AD user or an Azure AD group. If you grant the group with managed identity an admin role, skip steps 3 and 4. The administrator has full access to the database.

2. Create contained database users for the managed identity. Connect to the database from or to which you want to copy data by using tools like SQL Server Management Studio, with an Azure AD identity that has at least ALTER ANY USER permission. Run the following T-SQL:

```
CREATE USER [your_resource_name] FROM EXTERNAL PROVIDER;
```

3. Grant the managed identity needed permissions as you normally do for SQL users and others. Run the following code. For more options, see this document.

```
ALTER ROLE [role name] ADD MEMBER [your_resource_name];
```

4. Configure an Azure SQL Database linked service.

Example

```
{
    "name": "AzureSqlDbLinkedService",
    "properties": {
        "type": "AzureSqlDatabase",
        "typeProperties": {
            "connectionString": "Data Source=tcp:<servername>.database.windows.net,1433;Initial Catalog=
<databasename>;Connection Timeout=30"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

User-assigned managed identity authentication

A data factory or Synapse workspace can be associated with a user-assigned managed identities that represents the service when authenticating to other resources in Azure. You can use this managed identity for Azure SQL Database authentication. The designated factory or Synapse workspace can access and copy data from or to your database by using this identity.

To use user-assigned managed identity authentication, in addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
credentials	Specify the user-assigned managed identity as the credential object.	Yes

You also need to follow the steps below:

1. Provision an Azure Active Directory administrator for your server on the Azure portal if you haven't already done so. The Azure AD administrator can be an Azure AD user or an Azure AD group. If you grant the group with user-assigned managed identity an admin role, skip steps 3. The administrator has full access to the database.

2. Create contained database users for the user-assigned managed identity. Connect to the database from or to which you want to copy data by using tools like SQL Server Management Studio, with an Azure AD identity that has at least ALTER ANY USER permission. Run the following T-SQL:

   ```
   CREATE USER [your_resource_name] FROM EXTERNAL PROVIDER;
   ```

3. Create one or multiple user-assigned managed identities and grant the user-assigned managed identity needed permissions as you normally do for SQL users and others. Run the following code. For more options, see this document.

   ```
   ALTER ROLE [role name] ADD MEMBER [your_resource_name];
   ```

4. Assign one or multiple user-assigned managed identities to your data factory and create credentials for

each user-assigned managed identity.

5. Configure an Azure SQL Database linked service.

Example:

```
{
    "name": "AzureSqlDbLinkedService",
    "properties": {
        "type": "AzureSqlDatabase",
        "typeProperties": {
            "connectionString": "Data Source=tcp:<servername>.database.windows.net,1433;Initial Catalog=
<databasename>;Connection Timeout=30",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available to define datasets, see Datasets.

The following properties are supported for Azure SQL Database dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the dataset must be set to **AzureSqlTable**.	Yes
schema	Name of the schema.	No for source, Yes for sink
table	Name of the table/view.	No for source, Yes for sink
tableName	Name of the table/view with schema. This property is supported for backward compatibility. For new workload, use `schema` and `table`.	No for source, Yes for sink

Dataset properties example

```
{
    "name": "AzureSQLDbDataset",
    "properties":
    {
        "type": "AzureSqlTable",
        "linkedServiceName": {
            "referenceName": "<Azure SQL Database linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "typeProperties": {
            "schema": "<schema_name>",
            "table": "<table_name>"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see Pipelines. This section provides a list of properties supported by the Azure SQL Database source and sink.

Azure SQL Database as the source

> **TIP**
> To load data from Azure SQL Database efficiently by using data partitioning, learn more from Parallel copy from SQL database.

To copy data from Azure SQL Database, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the copy activity source must be set to **AzureSqlSource**. "SqlSource" type is still supported for backward compatibility.	Yes
sqlReaderQuery	This property uses the custom SQL query to read data. An example is `select * from MyTable`.	No
sqlReaderStoredProcedureName	The name of the stored procedure that reads data from the source table. The last SQL statement must be a SELECT statement in the stored procedure.	No
storedProcedureParameters	Parameters for the stored procedure. Allowed values are name or value pairs. The names and casing of parameters must match the names and casing of the stored procedure parameters.	No

PROPERTY	DESCRIPTION	REQUIRED
isolationLevel	Specifies the transaction locking behavior for the SQL source. The allowed values are: **ReadCommitted**, **ReadUncommitted**, **RepeatableRead**, **Serializable**, **Snapshot**. If not specified, the database's default isolation level is used. Refer to this doc for more details.	No
partitionOptions	Specifies the data partitioning options used to load data from Azure SQL Database. Allowed values are: **None** (default), **PhysicalPartitionsOfTable**, and **DynamicRange**. When a partition option is enabled (that is, not `None`), the degree of parallelism to concurrently load data from an Azure SQL Database is controlled by the `parallelCopies` setting on the copy activity.	No
partitionSettings	Specify the group of the settings for data partitioning. Apply when the partition option isn't `None`.	No
Under `partitionSettings`***:***		
partitionColumnName	Specify the name of the source column **in integer or date/datetime type** (`int`, `smallint`, `bigint`, `date`, `smalldatetime`, `datetime`, `datetime2`, or `datetimeoffset`) that will be used by range partitioning for parallel copy. If not specified, the index or the primary key of the table is autodetected and used as the partition column. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfDynamicRangePartitionCondition` in the WHERE clause. For an example, see the Parallel copy from SQL database section.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionUpperBound	The maximum value of the partition column for partition range splitting. This value is used to decide the partition stride, not for filtering the rows in table. All rows in the table or query result will be partitioned and copied. If not specified, copy activity auto detect the value. Apply when the partition option is `DynamicRange`. For an example, see the Parallel copy from SQL database section.	No
partitionLowerBound	The minimum value of the partition column for partition range splitting. This value is used to decide the partition stride, not for filtering the rows in table. All rows in the table or query result will be partitioned and copied. If not specified, copy activity auto detect the value. Apply when the partition option is `DynamicRange`. For an example, see the Parallel copy from SQL database section.	No

Note the following points:

- If **sqlReaderQuery** is specified for **AzureSqlSource**, the copy activity runs this query against the Azure SQL Database source to get the data. You also can specify a stored procedure by specifying **sqlReaderStoredProcedureName** and **storedProcedureParameters** if the stored procedure takes parameters.
- When using stored procedure in source to retrieve data, note if your stored procedure is designed as returning different schema when different parameter value is passed in, you may encounter failure or see unexpected result when importing schema from UI or when copying data to SQL database with auto table creation.

SQL query example

```
"activities":[
    {
        "name": "CopyFromAzureSQLDatabase",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Azure SQL Database input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AzureSqlSource",
                "sqlReaderQuery": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Stored procedure example

```
"activities":[
    {
        "name": "CopyFromAzureSQLDatabase",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Azure SQL Database input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AzureSqlSource",
                "sqlReaderStoredProcedureName": "CopyTestSrcStoredProcedureWithParameters",
                "storedProcedureParameters": {
                    "stringData": { "value": "str3" },
                    "identifier": { "value": "$$Text.Format('{0:yyyy}', <datetime parameter>)", "type":
"Int"}
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Stored procedure definition

```
CREATE PROCEDURE CopyTestSrcStoredProcedureWithParameters
(
    @stringData varchar(20),
    @identifier int
)
AS
SET NOCOUNT ON;
BEGIN
     select *
     from dbo.UnitTestSrcTable
     where dbo.UnitTestSrcTable.stringData != stringData
    and dbo.UnitTestSrcTable.identifier != identifier
END
GO
```

Azure SQL Database as the sink

> **TIP**
> Learn more about the supported write behaviors, configurations, and best practices from Best practice for loading data into Azure SQL Database.

To copy data to Azure SQL Database, the following properties are supported in the copy activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the copy activity sink must be set to **AzureSqlSink**. "SqlSink" type is still supported for backward compatibility.	Yes
preCopyScript	Specify a SQL query for the copy activity to run before writing data into Azure SQL Database. It's invoked only once per copy run. Use this property to clean up the preloaded data.	No
tableOption	Specifies whether to automatically create the sink table if not exists based on the source schema. Auto table creation is not supported when sink specifies stored procedure. Allowed values are: `none` (default), `autoCreate`.	No
sqlWriterStoredProcedureName	The name of the stored procedure that defines how to apply source data into a target table. This stored procedure is *invoked per batch*. For operations that run only once and have nothing to do with source data, for example, delete or truncate, use the `preCopyScript` property. See example from Invoke a stored procedure from a SQL sink.	No
storedProcedureTableTypeParameterName	The parameter name of the table type specified in the stored procedure.	No

PROPERTY	DESCRIPTION	REQUIRED
sqlWriterTableType	The table type name to be used in the stored procedure. The copy activity makes the data being moved available in a temp table with this table type. Stored procedure code can then merge the data that's being copied with existing data.	No
storedProcedureParameters	Parameters for the stored procedure. Allowed values are name and value pairs. Names and casing of parameters must match the names and casing of the stored procedure parameters.	No
writeBatchSize	Number of rows to insert into the SQL table *per batch*. The allowed value is **integer** (number of rows). By default, the service dynamically determines the appropriate batch size based on the row size.	No
writeBatchTimeout	The wait time for the batch insert operation to finish before it times out. The allowed value is **timespan**. An example is "00:30:00" (30 minutes).	No
disableMetricsCollection	The service collects metrics such as Azure SQL Database DTUs for copy performance optimization and recommendations, which introduces additional master DB access. If you are concerned with this behavior, specify `true` to turn it off.	No (default is `false`)
maxConcurrentConnections	The upper limit of concurrent connecti ons established to the data store durin g the activity run. Specify a value only when you want to limit concurrent con nections.	No
WriteBehavior	Specify the write behavior for copy activity to load data into Azure SQL Database. The allowed value is **Insert** and **Upsert**. By default, the service uses insert to load data.	No
upsertSettings	Specify the group of the settings for write behavior. Apply when the WriteBehavior option is `Upsert`.	No
Under `upsertSettings`:		

PROPERTY	DESCRIPTION	REQUIRED
useTempDB	Specify whether to use the a global temporary table or physical table as the interim table for upsert. By default, the service uses global temporary table as the interim table. value is `true`.	No
interimSchemaName	Specify the interim schema for creating interim table if physical table is used. Note: user need to have the permission for creating and deleting table. By default, interim table will share the same schema as sink table. Apply when the useTempDB option is `False`.	No
keys	Specify the column names for unique row identification. Either a single key or a series of keys can be used. If not specified, the primary key is used.	No

Example 1: Append data

```
"activities":[
    {
        "name": "CopyToAzureSQLDatabase",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Azure SQL Database output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureSqlSink",
                "tableOption": "autoCreate",
                "writeBatchSize": 100000
            }
        }
    }
]
```

Example 2: Invoke a stored procedure during copy

Learn more details from Invoke a stored procedure from a SQL sink.

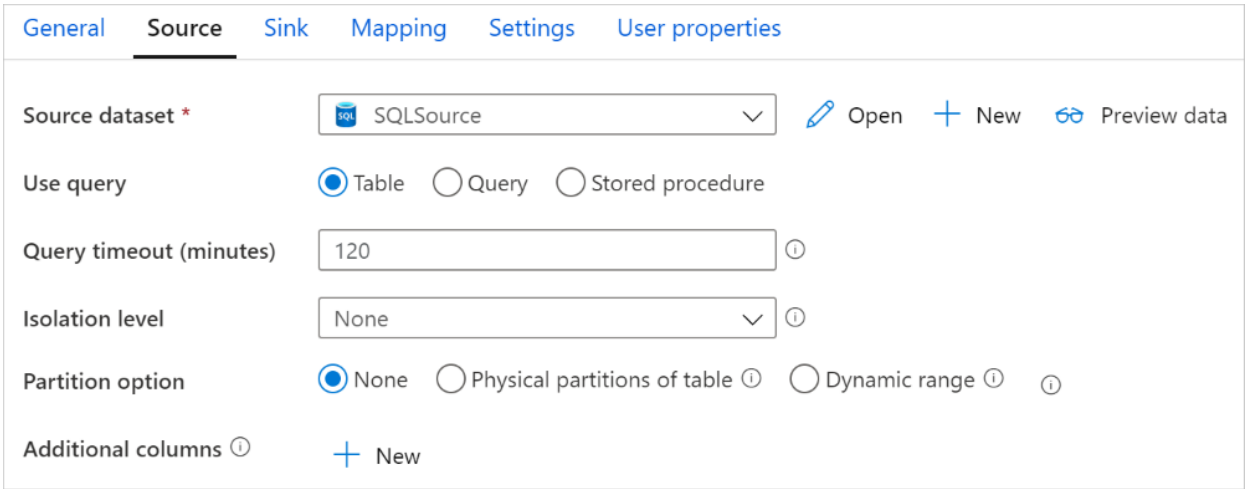
```
"activities":[
    {
        "name": "CopyToAzureSQLDatabase",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Azure SQL Database output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureSqlSink",
                "sqlWriterStoredProcedureName": "CopyTestStoredProcedureWithParameters",
                "storedProcedureTableTypeParameterName": "MyTable",
                "sqlWriterTableType": "MyTableType",
                "storedProcedureParameters": {
                    "identifier": { "value": "1", "type": "Int" },
                    "stringData": { "value": "str1" }
                }
            }
        }
    }
]
```

Example 3: Upsert data

```
"activities":[
    {
        "name": "CopyToAzureSQLDatabase",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Azure SQL Database output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureSqlSink",
                "tableOption": "autoCreate",
                "writeBehavior": "upsert",
                "upsertSettings": {
                    "useTempDB": true,
                    "keys": [
                        "<column name>"
                    ]
                },
            }
        }
    }
]
```

Parallel copy from SQL database

The Azure SQL Database connector in copy activity provides built-in data partitioning to copy data in parallel. You can find data partitioning options on the **Source** tab of the copy activity.



When you enable partitioned copy, copy activity runs parallel queries against your Azure SQL Database source to load data by partitions. The parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your Azure SQL Database.

You are suggested to enable parallel copy with data partitioning especially when you load large amount of data

from your Azure SQL Database. The following are suggested configurations for different scenarios. When copying data into file-based data store, it's recommended to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

SCENARIO	SUGGESTED SETTINGS
Full load from large table, with physical partitions.	**Partition option**: Physical partitions of table. During execution, the service automatically detects the physical partitions, and copies data by partitions. To check if your table has physical partition or not, you can refer to this query.
Full load from large table, without physical partitions, while with an integer or datetime column for data partitioning.	**Partition options**: Dynamic range partition. **Partition column** (optional): Specify the column used to partition data. If not specified, the index or primary key column is used. **Partition upper bound** and **partition lower bound** (optional): Specify if you want to determine the partition stride. This is not for filtering the rows in table, all rows in the table will be partitioned and copied. If not specified, copy activity auto detect the values. For example, if your partition column "ID" has values range from 1 to 100, and you set the lower bound as 20 and the upper bound as 80, with parallel copy as 4, the service retrieves data by 4 partitions - IDs in range <=20, [21, 50], [51, 80], and >=81, respectively.

SCENARIO	SUGGESTED SETTINGS

Load a large amount of data by using a custom query, without physical partitions, while with an integer or date/datetime column for data partitioning.	**Partition options**: Dynamic range partition.
	Query:
	`SELECT * FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	.
	Partition column: Specify the column used to partition data.
	Partition upper bound and **partition lower bound** (optional): Specify if you want to determine the partition stride. This is not for filtering the rows in table, all rows in the query result will be partitioned and copied. If not specified, copy activity auto detect the value.
	
	During execution, the service replaces `?AdfRangePartitionColumnName` with the actual column name and value ranges for each partition, and sends to Azure SQL Database.
	For example, if your partition column "ID" has values range from 1 to 100, and you set the lower bound as 20 and the upper bound as 80, with parallel copy as 4, the service retrieves data by 4 partitions- IDs in range <=20, [21, 50], [51, 80], and >=81, respectively.
	
	Here are more sample queries for different scenarios:
	1. Query the whole table:
	`SELECT * FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition`
	2. Query from a table with column selection and additional where-clause filters:
	`SELECT <column_list> FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	3. Query with subqueries:
	`SELECT <column_list> FROM (<your_sub_query>) AS T WHERE ?AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	4. Query with partition in subquery:
	`SELECT <column_list> FROM (SELECT <your_sub_query_column_list> FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition) AS T`

Best practices to load data with partition option:

1. Choose distinctive column as partition column (like primary key or unique key) to avoid data skew.
2. If the table has built-in partition, use partition option "Physical partitions of table" to get better performance.
3. If you use Azure Integration Runtime to copy data, you can set larger "Data Integration Units (DIU)" (>4) to utilize more computing resource. Check the applicable scenarios there.
4. "Degree of copy parallelism" control the partition numbers, setting this number too large sometime hurts the performance, recommend setting this number as (DIU or number of Self-hosted IR nodes) * (2 to 4).

Example: full load from large table with physical partitions

```
"source": {
    "type": "AzureSqlSource",
    "partitionOption": "PhysicalPartitionsOfTable"
}
```

Example: query with dynamic range partition

```
"source": {
    "type": "AzureSqlSource",
    "query":"SELECT * FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition AND
<your_additional_where_clause>",
    "partitionOption": "DynamicRange",
    "partitionSettings": {
        "partitionColumnName": "<partition_column_name>",
        "partitionUpperBound": "<upper_value_of_partition_column (optional) to decide the partition stride,
not as data filter>",
        "partitionLowerBound": "<lower_value_of_partition_column (optional) to decide the partition stride,
not as data filter>"
    }
}
```

Sample query to check physical partition

```
SELECT DISTINCT s.name AS SchemaName, t.name AS TableName, pf.name AS PartitionFunctionName, c.name AS
ColumnName, iif(pf.name is null, 'no', 'yes') AS HasPartition
FROM sys.tables AS t
LEFT JOIN sys.objects AS o ON t.object_id = o.object_id
LEFT JOIN sys.schemas AS s ON o.schema_id = s.schema_id
LEFT JOIN sys.indexes AS i ON t.object_id = i.object_id
LEFT JOIN sys.index_columns AS ic ON ic.partition_ordinal > 0 AND ic.index_id = i.index_id AND ic.object_id
= t.object_id
LEFT JOIN sys.columns AS c ON c.object_id = ic.object_id AND c.column_id = ic.column_id
LEFT JOIN sys.partition_schemes ps ON i.data_space_id = ps.data_space_id
LEFT JOIN sys.partition_functions pf ON pf.function_id = ps.function_id
WHERE s.name='[your schema]' AND t.name = '[your table name]'
```

If the table has physical partition, you would see "HasPartition" as "yes" like the following.

SchemaName	TableName	PartitionFunctionName	ColumnName	HasPartition
dbo	PerfWithPartition	RangePF1	Id	yes

Best practice for loading data into Azure SQL Database

When you copy data into Azure SQL Database, you might require different write behavior:

- Append: My source data has only new records.
- Upsert: My source data has both inserts and updates.
- Overwrite: I want to reload an entire dimension table each time.
- Write with custom logic: I need extra processing before the final insertion into the destination table.

Refer to the respective sections about how to configure in the service and best practices.

Append data

Appending data is the default behavior of this Azure SQL Database sink connector. the service does a bulk insert to write to your table efficiently. You can configure the source and sink accordingly in the copy activity.

Upsert data

Copy activity now supports natively loading data into a database temporary table and then update the data in sink table if key exists and otherwise insert new data. To learn more about upsert settings in copy activities, see Azure SQL Database as the sink.

Overwrite the entire table

You can configure the **preCopyScript** property in the copy activity sink. In this case, for each copy activity that runs, the service runs the script first. Then it runs the copy to insert the data. For example, to overwrite the entire table with the latest data, specify a script to first delete all the records before you bulk load the new data from

the source.

Write data with custom logic

The steps to write data with custom logic are similar to those described in the Upsert data section. When you need to apply extra processing before the final insertion of source data into the destination table, you can load to a staging table then invoke stored procedure activity, or invoke a stored procedure in copy activity sink to apply data, or use Mapping Data Flow.

Invoke a stored procedure from a SQL sink

When you copy data into Azure SQL Database, you also can configure and invoke a user-specified stored procedure with additional parameters on each batch of the source table. The stored procedure feature takes advantage of table-valued parameters.

You can use a stored procedure when built-in copy mechanisms don't serve the purpose. An example is when you want to apply extra processing before the final insertion of source data into the destination table. Some extra processing examples are when you want to merge columns, look up additional values, and insert into more than one table.

The following sample shows how to use a stored procedure to do an upsert into a table in Azure SQL Database. Assume that the input data and the sink **Marketing** table each have three columns: **ProfileID**, **State**, and **Category**. Do the upsert based on the **ProfileID** column, and only apply it for a specific category called "ProductA".

1. In your database, define the table type with the same name as **sqlWriterTableType**. The schema of the table type is the same as the schema returned by your input data.

   ```
   CREATE TYPE [dbo].[MarketingType] AS TABLE(
       [ProfileID] [varchar](256) NOT NULL,
       [State] [varchar](256) NOT NULL,
       [Category] [varchar](256) NOT NULL
   )
   ```

2. In your database, define the stored procedure with the same name as **sqlWriterStoredProcedureName**. It handles input data from your specified source and merges into the output table. The parameter name of the table type in the stored procedure is the same as **tableName** defined in the dataset.

   ```
   CREATE PROCEDURE spOverwriteMarketing @Marketing [dbo].[MarketingType] READONLY, @category
   varchar(256)
   AS
   BEGIN
   MERGE [dbo].[Marketing] AS target
   USING @Marketing AS source
   ON (target.ProfileID = source.ProfileID and target.Category = @category)
   WHEN MATCHED THEN
       UPDATE SET State = source.State
   WHEN NOT MATCHED THEN
       INSERT (ProfileID, State, Category)
       VALUES (source.ProfileID, source.State, source.Category);
   END
   ```

3. In your Azure Data Factory or Synapse pipeline, define the **SQL sink** section in the copy activity as follows:

```
"sink": {
    "type": "AzureSqlSink",
    "sqlWriterStoredProcedureName": "spOverwriteMarketing",
    "storedProcedureTableTypeParameterName": "Marketing",
    "sqlWriterTableType": "MarketingType",
    "storedProcedureParameters": {
        "category": {
            "value": "ProductA"
        }
    }
}
```

Mapping data flow properties

When transforming data in mapping data flow, you can read and write to tables from Azure SQL Database. For more information, see the source transformation and sink transformation in mapping data flows.

Source transformation

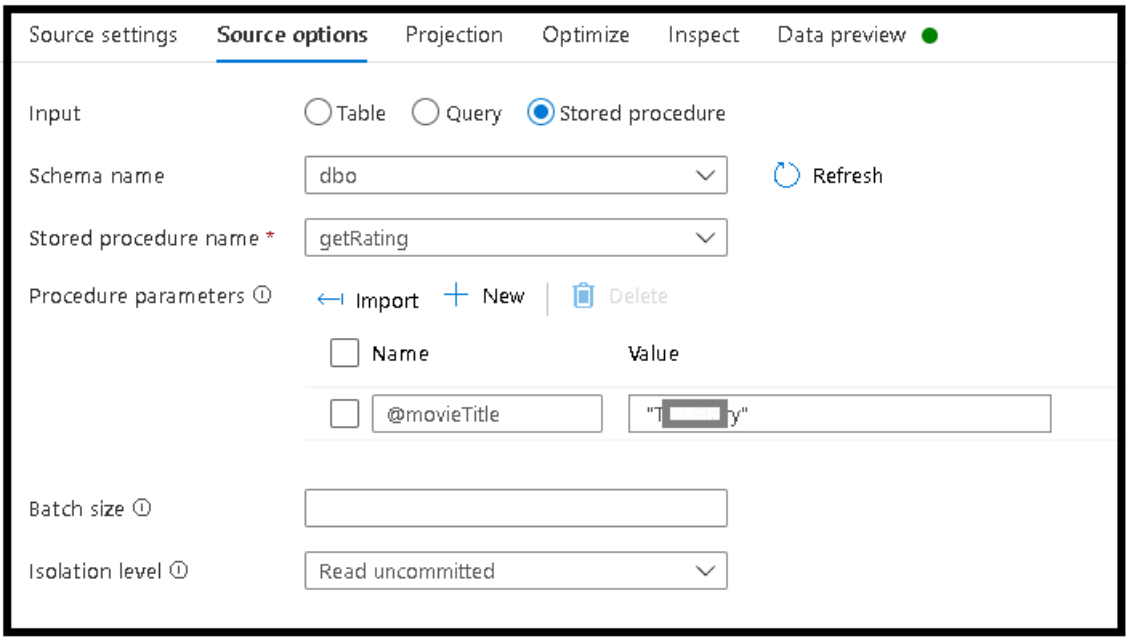
Settings specific to Azure SQL Database are available in the **Source Options** tab of the source transformation.

Input: Select whether you point your source at a table (equivalent of `Select * from <table-name>`) or enter a custom SQL query.

Query: If you select Query in the input field, enter a SQL query for your source. This setting overrides any table that you've chosen in the dataset. **Order By** clauses aren't supported here, but you can set a full SELECT FROM statement. You can also use user-defined table functions. **select * from udfGetData()** is a UDF in SQL that returns a table. This query will produce a source table that you can use in your data flow. Using queries is also a great way to reduce rows for testing or for lookups.

> **TIP**
>
> The common table expression (CTE) in SQL is not supported in the mapping data flow **Query** mode, because the prerequisite of using this mode is that queries can be used in the SQL query FROM clause but CTEs cannot do this. To use CTEs, you need to create a stored procedure using the following query:
>
> ```
> CREATE PROC CTESP @query nvarchar(max)
> AS
> BEGIN
> EXECUTE sp_executesql @query;
> END
> ```
>
> Then use the **Stored procedure** mode in the source transformation of the mapping data flow and set the `@query` like example `with CTE as (select 'test' as a) select * from CTE`. Then you can use CTEs as expected.

Stored procedure: Choose this option if you wish to generate a projection and source data from a stored procedure that is executed from your source database. You can type in the schema, procedure name, and parameters, or click on Refresh to ask the service to discover the schemas and procedure names. Then you can click on Import to import all procedure parameters using the form `@paraName`.

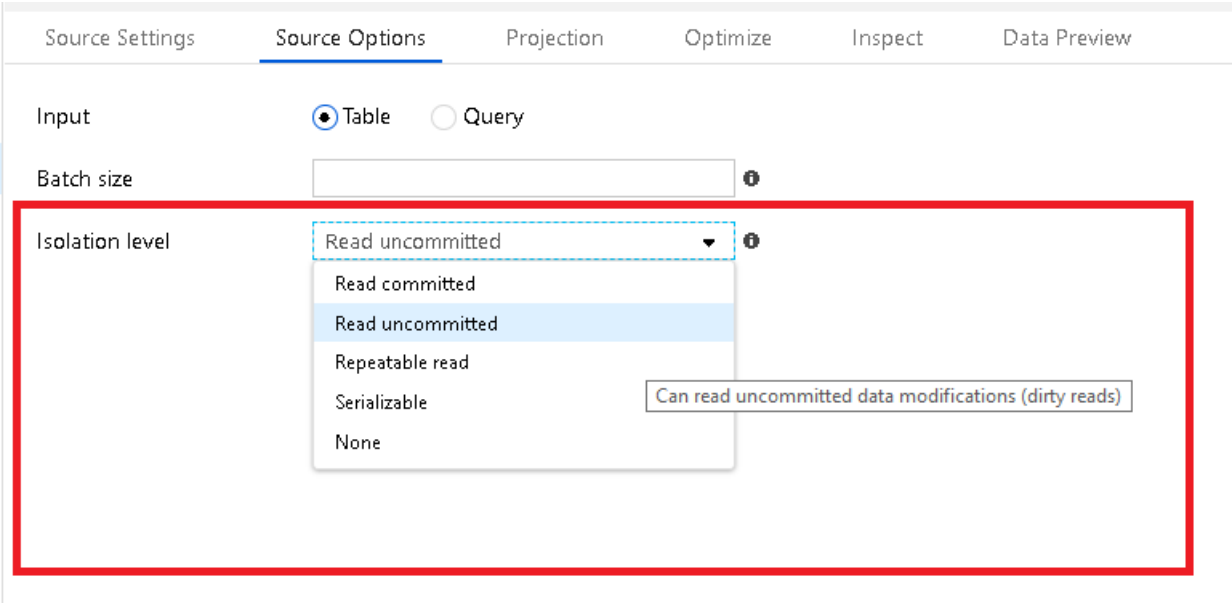


- SQL Example: `Select * from MyTable where customerId > 1000 and customerId < 2000`
- Parameterized SQL Example: `"select * from {$tablename} where orderyear > {$year}"`

Batch size: Enter a batch size to chunk large data into reads.

Isolation Level: The default for SQL sources in mapping data flow is read uncommitted. You can change the isolation level here to one of these values:

- Read Committed
- Read Uncommitted
- Repeatable Read
- Serializable
- None (ignore isolation level)



Enable incremental extract: Use this option to tell ADF to only process rows that have changed since the last time that the pipeline executed.

Incremental date column: When using the incremental extract feature, you must choose the date/time column that you wish to use as the watermark in your source table.

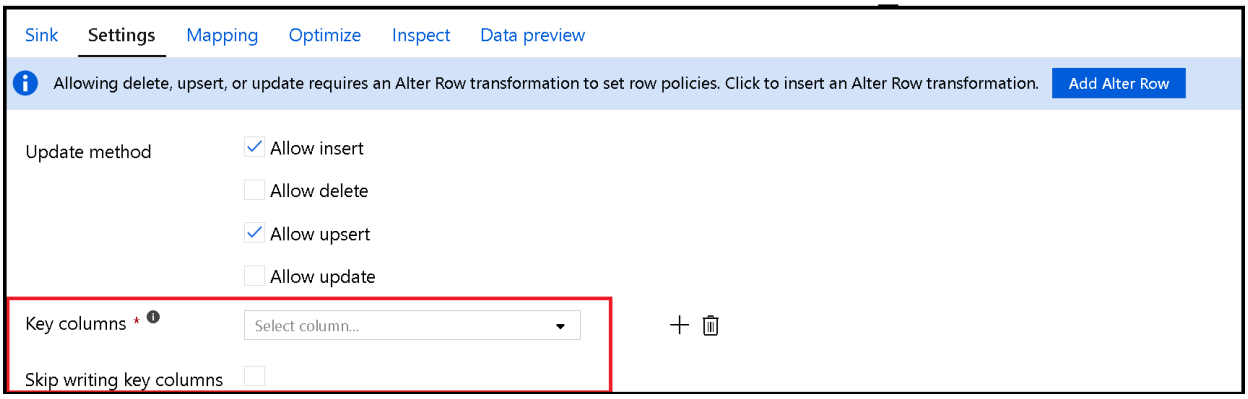
Start reading from beginning: Setting this option with incremental extract will instruct ADF to read all rows

on first execution of a pipeline with incremental extract turned on.

Sink transformation

Settings specific to Azure SQL Database are available in the **Settings** tab of the sink transformation.

Update method: Determines what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an alter-row transformation is required to tag rows for those actions. For updates, upserts and deletes, a key column or columns must be set to determine which row to alter.



The column name that you pick as the key here will be used by the service as part of the subsequent update, upsert, delete. Therefore, you must pick a column that exists in the Sink mapping. If you wish to not write the value to this key column, then click "Skip writing key columns".

You can parameterize the key column used here for updating your target Azure SQL Database table. If you have multiple columns for a composite key, the click on "Custom Expression" and you will be able to add dynamic content using the data flow expression language, which can include an array of strings with column names for a composite key.

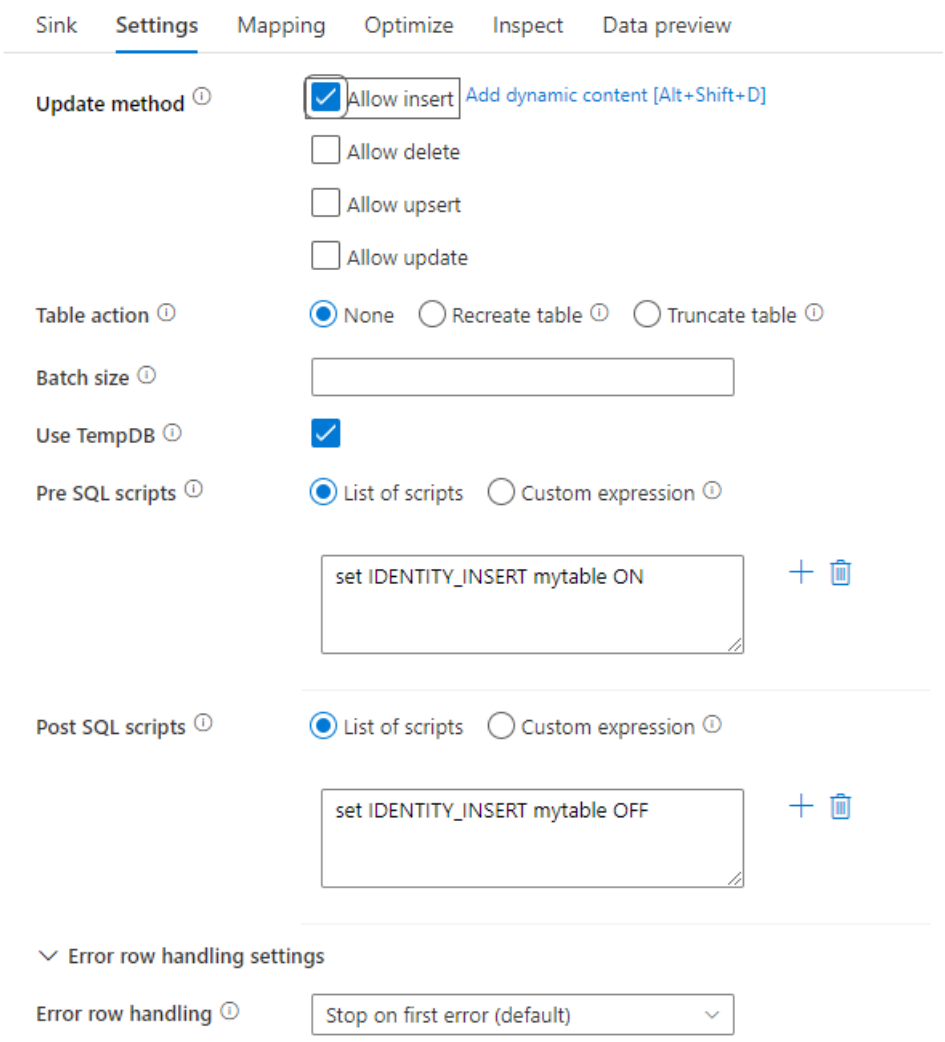
Table action: Determines whether to recreate or remove all rows from the destination table prior to writing.

- None: No action will be done to the table.
- Recreate: The table will get dropped and recreated. Required if creating a new table dynamically.
- Truncate: All rows from the target table will get removed.

Batch size: Controls how many rows are being written in each bucket. Larger batch sizes improve compression and memory optimization, but risk out of memory exceptions when caching data.

Use TempDB: By default, the service will use a global temporary table to store data as part of the loading process. You can alternatively uncheck the "Use TempDB" option and instead, ask the service to store the temporary holding table in a user database that is located in the database that is being used for this Sink.

Pre and Post SQL scripts: Enter multi-line SQL scripts that will execute before (pre-processing) and after (post-processing) data is written to your Sink database



TIP

1. It's recommended to break single batch scripts with multiple commands into multiple batches.
2. Only Data Definition Language (DDL) and Data Manipulation Language (DML) statements that return a simple update count can be run as part of a batch. Learn more from Performing batch operations

Error row handling

When writing to Azure SQL DB, certain rows of data may fail due to constraints set by the destination. Some common errors include:

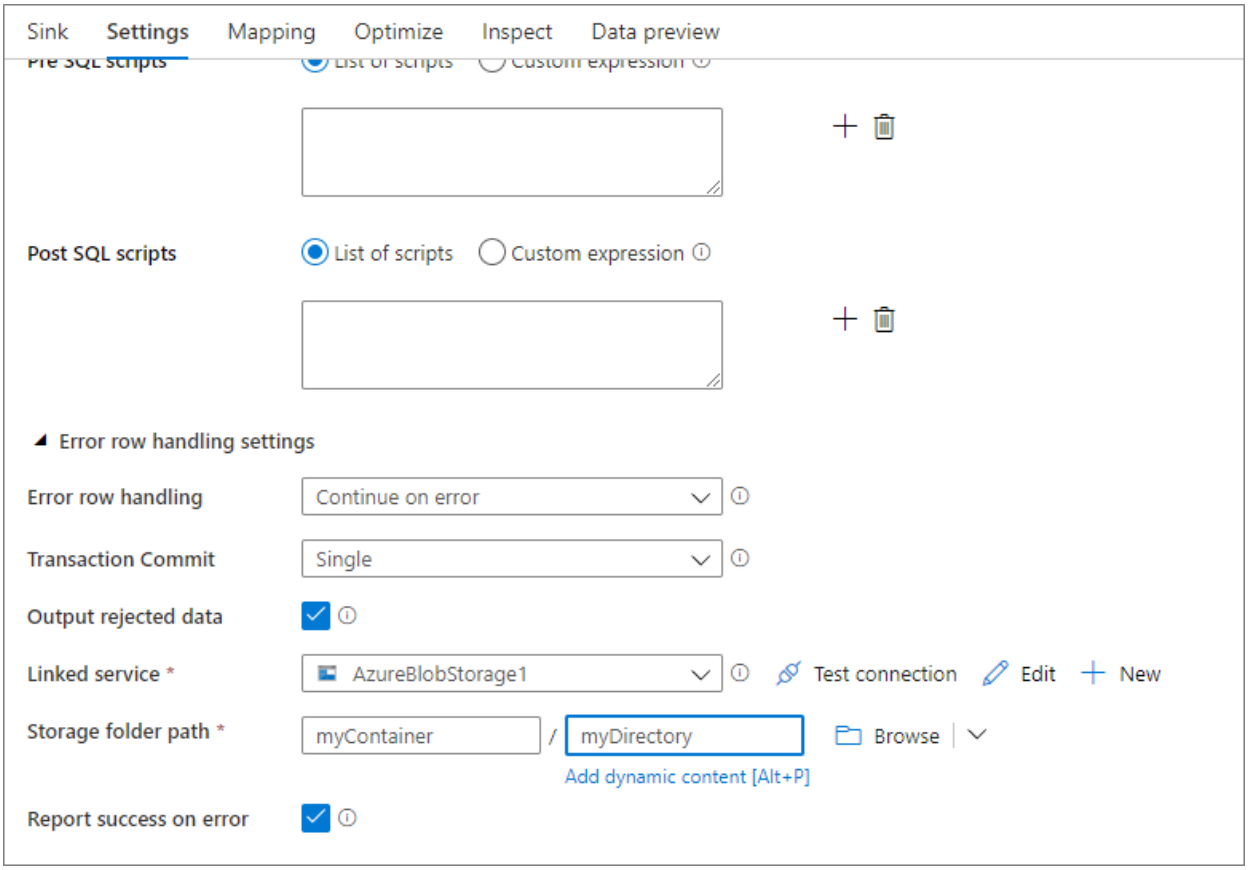
- String or binary data would be truncated in table
- Cannot insert the value NULL into column
- The INSERT statement conflicted with the CHECK constraint

By default, a data flow run will fail on the first error it gets. You can choose to **Continue on error** that allows your data flow to complete even if individual rows have errors. The service provides different options for you to handle these error rows.

Transaction Commit: Choose whether your data gets written in a single transaction or in batches. Single transaction will provide worse performance but no data written will be visible to others until the transaction completes.

Output rejected data: If enabled, you can output the error rows into a csv file in Azure Blob Storage or an Azure Data Lake Storage Gen2 account of your choosing. This will write the error rows with three additional columns: the SQL operation like INSERT or UPDATE, the data flow error code, and the error message on the row.

Report success on error: If enabled, the data flow will be marked as a success even if error rows are found.



Data type mapping for Azure SQL Database

When data is copied from or to Azure SQL Database, the following mappings are used from Azure SQL Database data types to Azure Data Factory interim data types. The same mappings are used by the Synapse pipeline feature, which implements Azure Data Factory directly. To learn how the copy activity maps the source schema and data type to the sink, see Schema and data type mappings.

AZURE SQL DATABASE DATA TYPE	DATA FACTORY INTERIM DATA TYPE
bigint	Int64
binary	Byte[]
bit	Boolean
char	String, Char[]
date	DateTime
Datetime	DateTime
datetime2	DateTime
Datetimeoffset	DateTimeOffset
Decimal	Decimal
FILESTREAM attribute (varbinary(max))	Byte[]

AZURE SQL DATABASE DATA TYPE	DATA FACTORY INTERIM DATA TYPE
Float	Double
image	Byte[]
int	Int32
money	Decimal
nchar	String, Char[]
ntext	String, Char[]
numeric	Decimal
nvarchar	String, Char[]
real	Single
rowversion	Byte[]
smalldatetime	DateTime
smallint	Int16
smallmoney	Decimal
sql_variant	Object
text	String, Char[]
time	TimeSpan
timestamp	Byte[]
tinyint	Byte
uniqueidentifier	Guid
varbinary	Byte[]
varchar	String, Char[]
xml	String

NOTE

For data types that map to the Decimal interim type, currently Copy activity supports precision up to 28. If you have data with precision larger than 28, consider converting to a string in SQL query.

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity

Using Always Encrypted

When you copy data from/to Azure SQL Database with Always Encrypted, follow below steps:

1. Store the Column Master Key (CMK) in an Azure Key Vault. Learn more on how to configure Always Encrypted by using Azure Key Vault
2. Make sure to great access to the key vault where the Column Master Key (CMK) is stored. Refer to this article for required permissions.
3. Create linked service to connect to your SQL database and enable 'Always Encrypted' function by using either managed identity or service principal.

> **NOTE**
>
> Azure SQL Database Always Encrypted supports below scenarios:
>
> 1. Either source or sink data stores is using managed identity or service principal as key provider authentication type.
> 2. Both source and sink data stores are using managed identity as key provider authentication type.
> 3. Both source and sink data stores are using the same service principal as key provider authentication type.

> **NOTE**
>
> Currently, Azure SQL Database **Always Encrypted** is only supported for source transformation in mapping data flows.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores and formats.

Copy and transform data in Azure SQL Managed Instance using Azure Data Factory or Synapse Analytics

8/19/2022 • 30 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity to copy data from and to Azure SQL Managed Instance, and use Data Flow to transform data in Azure SQL Managed Instance. To learn more, read the introductory articles for Azure Data Factory and Synapse Analytics.

Supported capabilities

This Azure SQL Managed Instance connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/sink)	① ②	✓ Public preview
Mapping data flow (source/sink)	①	✓ Public preview
Lookup activity	① ②	✓ Public preview
GetMetadata activity	① ②	✓ Public preview
Script activity	① ②	✓ Public preview
Stored procedure activity	① ②	✓ Public preview

① Azure integration runtime ② Self-hosted integration runtime

For Copy activity, this Azure SQL Database connector supports these functions:

- Copying data by using SQL authentication and Azure Active Directory (Azure AD) Application token authentication with a service principal or managed identities for Azure resources.
- As a source, retrieving data by using a SQL query or a stored procedure. You can also choose to parallel copy from SQL MI source, see the Parallel copy from SQL MI section for details.
- As a sink, automatically creating destination table if not exists based on the source schema; appending data to a table or invoking a stored procedure with custom logic during copy.

Prerequisites

To access the SQL Managed Instance public endpoint, you can use a managed Azure integration runtime. Make sure that you enable the public endpoint and also allow public endpoint traffic on the network security group so that the service can connect to your database. For more information, see this guidance.

To access the SQL Managed Instance private endpoint, set up a self-hosted integration runtime that can access the database. If you provision the self-hosted integration runtime in the same virtual network as your managed instance, make sure that your integration runtime machine is in a different subnet than your managed instance.

If you provision your self-hosted integration runtime in a different virtual network than your managed instance, you can use either a virtual network peering or a virtual network to virtual network connection. For more information, see Connect your application to SQL Managed Instance.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

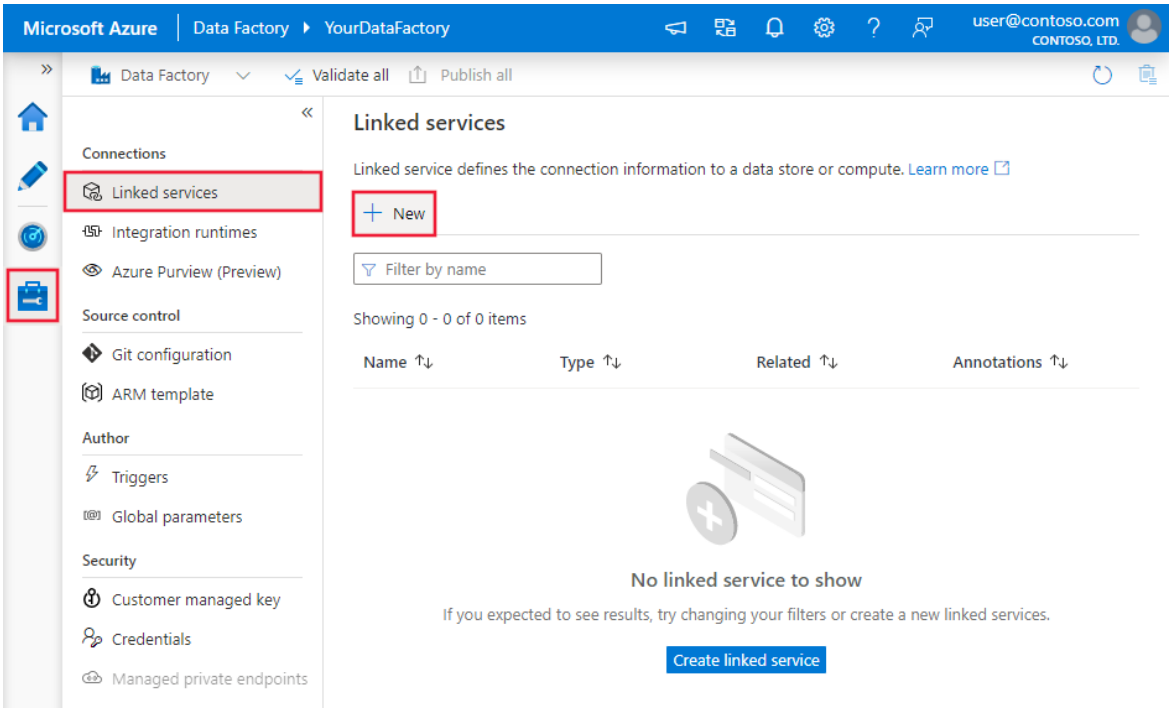
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to an Azure SQL Managed instance using UI

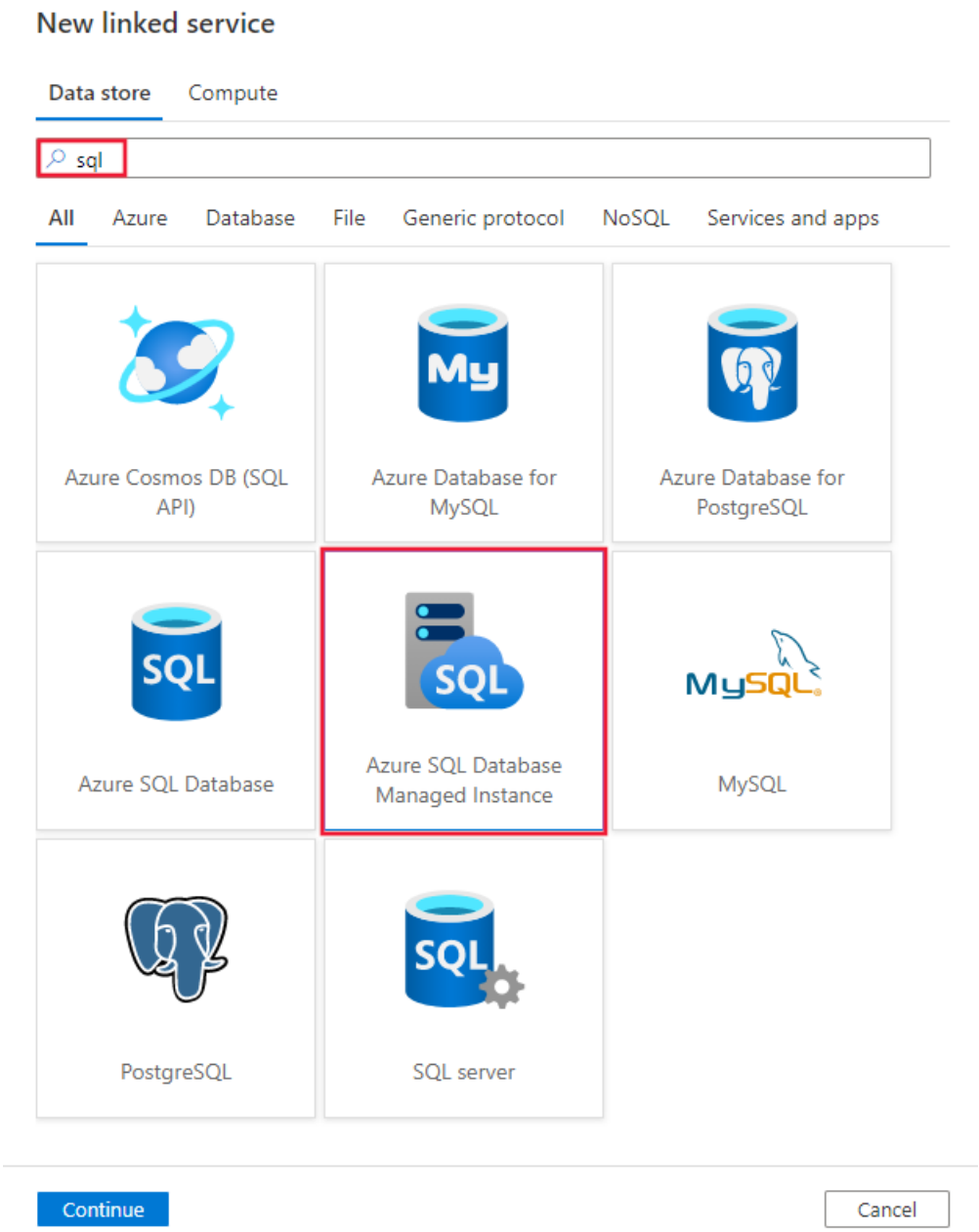
Use the following steps to create a linked service to an SQL Managed instance in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

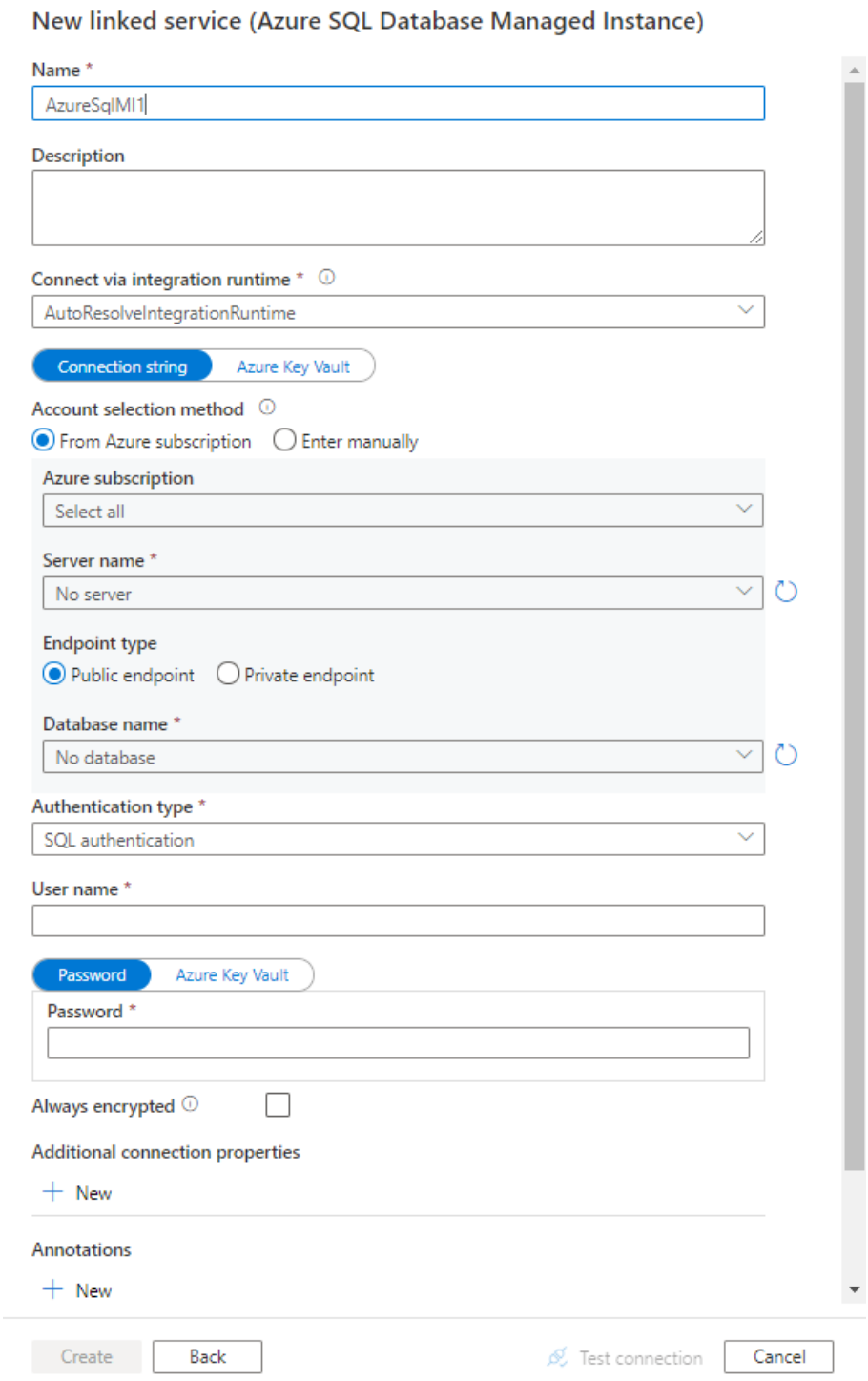
   - Azure Data Factory
   - Azure Synapse



2. Search for SQL and select the Azure SQL Server Managed Instance connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Azure Data Factory entities specific to the SQL Managed Instance connector.

Linked service properties

These generic properties are supported for an SQL Managed Instance linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AzureSqlMI**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	This property specifies the **connectionString** information that's needed to connect to SQL Managed Instance by using SQL authentication. For more information, see the following examples. The default port is 1433. If you're using SQL Managed Instance with a public endpoint, explicitly specify port 3342. You also can put a password in Azure Key Vault. If it's SQL authentication, pull the `password` configuration out of the connection string. For more information, see the JSON example following the table and Store credentials in Azure Key Vault.	Yes
azureCloudType	For service principal authentication, specify the type of Azure cloud environment to which your Azure AD application is registered. Allowed values are **AzurePublic**, **AzureChina**, **AzureUsGovernment**, and **AzureGermany**. By default, the service's cloud environment is used.	No
alwaysEncryptedSettings	Specify **alwaysencryptedsettings** information that's needed to enable Always Encrypted to protect sensitive data stored in SQL server by using either managed identity or service principal. For more information, see the JSON example following the table and Using Always Encrypted section. If not specified, the default always encrypted setting is disabled.	No
connectVia	This integration runtime is used to connect to the data store. You can use a self-hosted integration runtime or an Azure integration runtime if your managed instance has a public endpoint and allows the service to access it. If not specified, the default Azure integration runtime is used.	Yes

For different authentication types, refer to the following sections on specific properties, prerequisites and JSON samples, respectively:

- SQL authentication
- Service principal authentication
- System-assigned managed identity authentication
- User-assigned managed identity authentication

SQL authentication

To use SQL authentication authentication type, specify the generic properties that are described in the preceding section.

Example 1: use SQL authentication

```
{
    "name": "AzureSqlMILinkedService",
    "properties": {
        "type": "AzureSqlMI",
        "typeProperties": {
            "connectionString": "Data Source=<hostname,port>;Initial Catalog=<databasename>;Integrated
Security=False;User ID=<username>;Password=<password>;"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 2: use SQL authentication with a password in Azure Key Vault

```
{
    "name": "AzureSqlMILinkedService",
    "properties": {
        "type": "AzureSqlMI",
        "typeProperties": {
            "connectionString": "Data Source=<hostname,port>;Initial Catalog=<databasename>;Integrated
Security=False;User ID=<username>;",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 3: use SQL authentication with Always Encrypted

```
{
    "name": "AzureSqlMILinkedService",
    "properties": {
        "type": "AzureSqlMI",
        "typeProperties": {
            "connectionString": "Data Source=<hostname,port>;Initial Catalog=<databasename>;Integrated
Security=False;User ID=<username>;Password=<password>;"
        },
        "alwaysEncryptedSettings": {
            "alwaysEncryptedAkvAuthType": "ServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Service principal authentication

To use service principal authentication, in addition to the generic properties that are described in the preceding section, specify the following properties

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key. Mark this field as **SecureString** to store it securely or reference a secret stored in Azure Key Vault.	Yes
tenant	Specify the tenant information, like the domain name or tenant ID, under which your application resides. Retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Yes

You also need to follow the steps below:

1. Follow the steps to Provision an Azure Active Directory administrator for your Managed Instance.
2. Create an Azure Active Directory application from the Azure portal. Make note of the application name and the following values that define the linked service:
   - Application ID
   - Application key
   - Tenant ID
3. Create logins for the service principal. In SQL Server Management Studio (SSMS), connect to your managed instance using a SQL Server account that is a **sysadmin**. In **master** database, run the following T-SQL:

   ```
   CREATE LOGIN [your application name] FROM EXTERNAL PROVIDER
   ```

4. Create contained database users for the service principal. Connect to the database from or to which you want to copy data, run the following T-SQL:

```
CREATE USER [your application name] FROM EXTERNAL PROVIDER
```

5. Grant the service principal needed permissions as you normally do for SQL users and others. Run the following code. For more options, see this document.

```
ALTER ROLE [role name e.g. db_owner] ADD MEMBER [your application name]
```

6. Configure a SQL Managed Instance linked service.

Example: use service principal authentication

```
{
    "name": "AzureSqlDbLinkedService",
    "properties": {
        "type": "AzureSqlMI",
        "typeProperties": {
            "connectionString": "Data Source=<hostname,port>;Initial Catalog=<databasename>;",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

System-assigned managed identity authentication

A data factory or Synapse workspace can be associated with a system-assigned managed identity for Azure resources that represents the service for authentication to other Azure services. You can use this managed identity for SQL Managed Instance authentication. The designated service can access and copy data from or to your database by using this identity.

To use system-assigned managed identity authentication, specify the generic properties that are described in the preceding section, and follow these steps.

1. Follow the steps to Provision an Azure Active Directory administrator for your Managed Instance.

2. Create logins for the system-assigned managed identity. In SQL Server Management Studio (SSMS), connect to your managed instance using a SQL Server account that is a **sysadmin**. In **master** database, run the following T-SQL:

```
CREATE LOGIN [your_factory_or_workspace_ name] FROM EXTERNAL PROVIDER
```

3. Create contained database users for the system-assigned managed identity. Connect to the database from or to which you want to copy data, run the following T-SQL:

```
CREATE USER [your_factory_or_workspace_name] FROM EXTERNAL PROVIDER
```

4. Grant the system-assigned managed identity needed permissions as you normally do for SQL users and others. Run the following code. For more options, see this document.

```
ALTER ROLE [role name e.g. db_owner] ADD MEMBER [your_factory_or_workspace_name]
```

5. Configure a SQL Managed Instance linked service.

Example: uses system-assigned managed identity authentication

```
{
    "name": "AzureSqlDbLinkedService",
    "properties": {
        "type": "AzureSqlMI",
        "typeProperties": {
            "connectionString": "Data Source=<hostname,port>;Initial Catalog=<databasename>;"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

User-assigned managed identity authentication

A data factory or Synapse workspace can be associated with a user-assigned managed identities that represents the service for authentication to other Azure services. You can use this managed identity for SQL Managed Instance authentication. The designated service can access and copy data from or to your database by using this identity.

To use user-assigned managed identity authentication, in addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
credentials	Specify the user-assigned managed identity as the credential object.	Yes

You also need to follow the steps below:

1. Follow the steps to Provision an Azure Active Directory administrator for your Managed Instance.

2. Create logins for the user-assigned managed identity. In SQL Server Management Studio (SSMS), connect to your managed instance using a SQL Server account that is a **sysadmin**. In **master** database, run the following T-SQL:

```
CREATE LOGIN [your_factory_or_workspace_ name] FROM EXTERNAL PROVIDER
```

3. Create contained database users for the user-assigned managed identity. Connect to the database from or to which you want to copy data, run the following T-SQL:

```
CREATE USER [your_factory_or_workspace_name] FROM EXTERNAL PROVIDER
```

4. Create one or multiple user-assigned managed identities and grant the user-assigned managed identity needed permissions as you normally do for SQL users and others. Run the following code. For more options, see this document.

```
ALTER ROLE [role name e.g. db_owner] ADD MEMBER [your_factory_or_workspace_name]
```

5. Assign one or multiple user-assigned managed identities to your data factory and create credentials for each user-assigned managed identity.

6. Configure a SQL Managed Instance linked service.

Example: uses user-assigned managed identity authentication

```
{
    "name": "AzureSqlDbLinkedService",
    "properties": {
        "type": "AzureSqlMI",
        "typeProperties": {
            "connectionString": "Data Source=<hostname,port>;Initial Catalog=<databasename>;",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for use to define datasets, see the datasets article. This section provides a list of properties supported by the SQL Managed Instance dataset.

To copy data to and from SQL Managed Instance, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **AzureSqlMITable**.	Yes
schema	Name of the schema.	No for source, Yes for sink
table	Name of the table/view.	No for source, Yes for sink
tableName	Name of the table/view with schema. This property is supported for backward compatibility. For new workload, use `schema` and `table`.	No for source, Yes for sink

Example

```
{
    "name": "AzureSqlMIDataset",
    "properties":
    {
        "type": "AzureSqlMITable",
        "linkedServiceName": {
            "referenceName": "<SQL Managed Instance linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "typeProperties": {
            "schema": "<schema_name>",
            "table": "<table_name>"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for use to define activities, see the Pipelines article. This section provides a list of properties supported by the SQL Managed Instance source and sink.

SQL Managed Instance as a source

> **TIP**
> To load data from SQL MI efficiently by using data partitioning, learn more from Parallel copy from SQL MI.

To copy data from SQL Managed Instance, the following properties are supported in the copy activity source section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **SqlMISource**.	Yes
sqlReaderQuery	This property uses the custom SQL query to read data. An example is `select * from MyTable`.	No
sqlReaderStoredProcedureName	This property is the name of the stored procedure that reads data from the source table. The last SQL statement must be a SELECT statement in the stored procedure.	No
storedProcedureParameters	These parameters are for the stored procedure. Allowed values are name or value pairs. The names and casing of the parameters must match the names and casing of the stored procedure parameters.	No

PROPERTY	DESCRIPTION	REQUIRED
isolationLevel	Specifies the transaction locking behavior for the SQL source. The allowed values are: **ReadCommitted**, **ReadUncommitted**, **RepeatableRead**, **Serializable**, **Snapshot**. If not specified, the database's default isolation level is used. Refer to this doc for more details.	No
partitionOptions	Specifies the data partitioning options used to load data from SQL MI. Allowed values are: **None** (default), **PhysicalPartitionsOfTable**, and **DynamicRange**. When a partition option is enabled (that is, not `None`), the degree of parallelism to concurrently load data from SQL MI is controlled by the `parallelCopies` setting on the copy activity.	No
partitionSettings	Specify the group of the settings for data partitioning. Apply when the partition option isn't `None`.	No
Under `partitionSettings`***:***		
partitionColumnName	Specify the name of the source column **in integer or date/datetime type** (`int`, `smallint`, `bigint`, `date`, `smalldatetime`, `datetime`, `datetime2`, or `datetimeoffset`) that will be used by range partitioning for parallel copy. If not specified, the index or the primary key of the table is auto-detected and used as the partition column. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfDynamicRangePartitionCondition` in the WHERE clause. For an example, see the Parallel copy from SQL database section.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionUpperBound	The maximum value of the partition column for partition range splitting. This value is used to decide the partition stride, not for filtering the rows in table. All rows in the table or query result will be partitioned and copied. If not specified, copy activity auto detect the value. Apply when the partition option is `DynamicRange`. For an example, see the Parallel copy from SQL database section.	No
partitionLowerBound	The minimum value of the partition column for partition range splitting. This value is used to decide the partition stride, not for filtering the rows in table. All rows in the table or query result will be partitioned and copied. If not specified, copy activity auto detect the value. Apply when the partition option is `DynamicRange`. For an example, see the Parallel copy from SQL database section.	No

Note the following points:

- If **sqlReaderQuery** is specified for **SqlMISource**, the copy activity runs this query against the SQL Managed Instance source to get the data. You also can specify a stored procedure by specifying **sqlReaderStoredProcedureName** and **storedProcedureParameters** if the stored procedure takes parameters.
- When using stored procedure in source to retrieve data, note if your stored procedure is designed as returning different schema when different parameter value is passed in, you may encounter failure or see unexpected result when importing schema from UI or when copying data to SQL database with auto table creation.

Example: Use a SQL query

```
"activities":[
    {
        "name": "CopyFromAzureSqlMI",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SQL Managed Instance input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SqlMISource",
                "sqlReaderQuery": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Example: Use a stored procedure

```
"activities":[
    {
        "name": "CopyFromAzureSqlMI",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SQL Managed Instance input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SqlMISource",
                "sqlReaderStoredProcedureName": "CopyTestSrcStoredProcedureWithParameters",
                "storedProcedureParameters": {
                    "stringData": { "value": "str3" },
                    "identifier": { "value": "$$Text.Format('{0:yyyy}', <datetime parameter>)", "type":
"Int"}
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

The stored procedure definition

```
CREATE PROCEDURE CopyTestSrcStoredProcedureWithParameters
(
    @stringData varchar(20),
    @identifier int
)
AS
SET NOCOUNT ON;
BEGIN
    select *
    from dbo.UnitTestSrcTable
    where dbo.UnitTestSrcTable.stringData != stringData
    and dbo.UnitTestSrcTable.identifier != identifier
END
GO
```

SQL Managed Instance as a sink

> **TIP**
> Learn more about the supported write behaviors, configurations, and best practices from Best practice for loading data into SQL Managed Instance.

To copy data to SQL Managed Instance, the following properties are supported in the copy activity sink section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to **SqlMISink**.	Yes
preCopyScript	This property specifies a SQL query for the copy activity to run before writing data into SQL Managed Instance. It's invoked only once per copy run. You can use this property to clean up preloaded data.	No
tableOption	Specifies whether to automatically create the sink table if not exists based on the source schema. Auto table creation is not supported when sink specifies stored procedure. Allowed values are: `none` (default), `autoCreate`.	No
sqlWriterStoredProcedureName	The name of the stored procedure that defines how to apply source data into a target table. This stored procedure is *invoked per batch*. For operations that run only once and have nothing to do with source data, for example, delete or truncate, use the `preCopyScript` property. See example from Invoke a stored procedure from a SQL sink.	No
storedProcedureTableTypeParameterName	The parameter name of the table type specified in the stored procedure.	No

PROPERTY	DESCRIPTION	REQUIRED
sqlWriterTableType	The table type name to be used in the stored procedure. The copy activity makes the data being moved available in a temp table with this table type. Stored procedure code can then merge the data that's being copied with existing data.	No
storedProcedureParameters	Parameters for the stored procedure. Allowed values are name and value pairs. Names and casing of parameters must match the names and casing of the stored procedure parameters.	No
writeBatchSize	Number of rows to insert into the SQL table *per batch*. Allowed values are integers for the number of rows. By default, the service dynamically determines the appropriate batch size based on the row size.	No
writeBatchTimeout	This property specifies the wait time for the batch insert operation to complete before it times out. Allowed values are for the timespan. An example is "00:30:00," which is 30 minutes.	No
maxConcurrentConnections	The upper limit of concurrent connecti ons established to the data store durin g the activity run. Specify a value only when you want to limit concurrent con nections.	No
WriteBehavior	Specify the write behavior for copy activity to load data into Azure SQL MI. The allowed value is **Insert** and **Upsert**. By default, the service uses insert to load data.	No
upsertSettings	Specify the group of the settings for write behavior. Apply when the WriteBehavior option is `Upert`.	No
Under `upsertSettings`:		
useTempDB	Specify whether to use the a global temporary table or physical table as the interim table for upsert. By default, the service uses global temporary table as the interim table. value is `true`.	No

PROPERTY	DESCRIPTION	REQUIRED
interimSchemaName	Specify the interim schema for creating interim table if physical table is used. Note: user need to have the permission for creating and deleting table. By default, interim table will share the same schema as sink table. Apply when the useTempDB option is `False`.	No
keys	Specify the column names for unique row identification. Either a single key or a series of keys can be used. If not specified, the primary key is used.	No

Example 1: Append data

```
"activities":[
    {
        "name": "CopyToAzureSqlMI",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<SQL Managed Instance output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "SqlMISink",
                "tableOption": "autoCreate",
                "writeBatchSize": 100000
            }
        }
    }
]
```

Example 2: Invoke a stored procedure during copy

Learn more details from Invoke a stored procedure from a SQL MI sink.

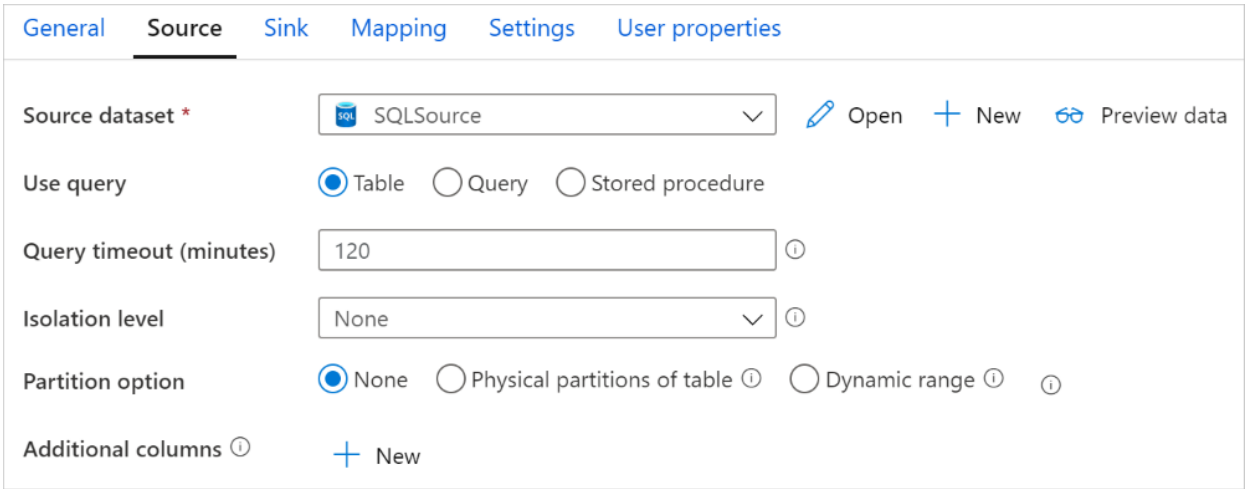
```
"activities":[
    {
        "name": "CopyToAzureSqlMI",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<SQL Managed Instance output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "SqlMISink",
                "sqlWriterStoredProcedureName": "CopyTestStoredProcedureWithParameters",
                "storedProcedureTableTypeParameterName": "MyTable",
                "sqlWriterTableType": "MyTableType",
                "storedProcedureParameters": {
                    "identifier": { "value": "1", "type": "Int" },
                    "stringData": { "value": "str1" }
                }
            }
        }
    }
]
```

Example 3: Upsert data

```
"activities":[
    {
        "name": "CopyToAzureSqlMI",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<SQL Managed Instance output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "SqlMISink",
                "tableOption": "autoCreate",
                "writeBehavior": "upsert",
                "upsertSettings": {
                    "useTempDB": true,
                    "keys": [
                        "<column name>"
                    ]
                },
            }
        }
    }
]
```

Parallel copy from SQL MI

The Azure SQL Managed Instance connector in copy activity provides built-in data partitioning to copy data in parallel. You can find data partitioning options on the **Source** tab of the copy activity.



When you enable partitioned copy, copy activity runs parallel queries against your SQL MI source to load data by partitions. The parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your SQL MI.

You are suggested to enable parallel copy with data partitioning especially when you load large amount of data from your SQL MI. The following are suggested configurations for different scenarios. When copying data into

file-based data store, it's recommended to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

SCENARIO	SUGGESTED SETTINGS
Full load from large table, with physical partitions.	**Partition option**: Physical partitions of table. During execution, the service automatically detects the physical partitions, and copies data by partitions. To check if your table has physical partition or not, you can refer to this query.
Full load from large table, without physical partitions, while with an integer or datetime column for data partitioning.	**Partition options**: Dynamic range partition. **Partition column** (optional): Specify the column used to partition data. If not specified, the index or primary key column is used. **Partition upper bound** and **partition lower bound** (optional): Specify if you want to determine the partition stride. This is not for filtering the rows in table, all rows in the table will be partitioned and copied. If not specified, copy activity auto detect the values. For example, if your partition column "ID" has values range from 1 to 100, and you set the lower bound as 20 and the upper bound as 80, with parallel copy as 4, the service retrieves data by 4 partitions - IDs in range <=20, [21, 50], [51, 80], and >=81, respectively.

SCENARIO	SUGGESTED SETTINGS

Load a large amount of data by using a custom query, without physical partitions, while with an integer or date/datetime column for data partitioning.	**Partition options**: Dynamic range partition.
	Query:
	`SELECT * FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	.
	Partition column: Specify the column used to partition data.
	Partition upper bound and **partition lower bound** (optional): Specify if you want to determine the partition stride. This is not for filtering the rows in table, all rows in the query result will be partitioned and copied. If not specified, copy activity auto detect the value.
	
	During execution, the service replaces `?AdfRangePartitionColumnName` with the actual column name and value ranges for each partition, and sends to SQL MI.
	For example, if your partition column "ID" has values range from 1 to 100, and you set the lower bound as 20 and the upper bound as 80, with parallel copy as 4, the service retrieves data by 4 partitions- IDs in range <=20, [21, 50], [51, 80], and >=81, respectively.
	
	Here are more sample queries for different scenarios:
	1. Query the whole table:
	`SELECT * FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition`
	2. Query from a table with column selection and additional where-clause filters:
	`SELECT <column_list> FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	3. Query with subqueries:
	`SELECT <column_list> FROM (<your_sub_query>) AS T WHERE ?AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	4. Query with partition in subquery:
	`SELECT <column_list> FROM (SELECT <your_sub_query_column_list> FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition) AS T`

Best practices to load data with partition option:

1. Choose distinctive column as partition column (like primary key or unique key) to avoid data skew.
2. If the table has built-in partition, use partition option "Physical partitions of table" to get better performance.
3. If you use Azure Integration Runtime to copy data, you can set larger "Data Integration Units (DIU)" (>4) to utilize more computing resource. Check the applicable scenarios there.
4. "Degree of copy parallelism" control the partition numbers, setting this number too large sometime hurts the performance, recommend setting this number as (DIU or number of Self-hosted IR nodes) * (2 to 4).

Example: full load from large table with physical partitions

```
"source": {
    "type": "SqlMISource",
    "partitionOption": "PhysicalPartitionsOfTable"
}
```

Example: query with dynamic range partition

```
"source": {
    "type": "SqlMISource",
    "query":"SELECT * FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition AND
<your_additional_where_clause>",
    "partitionOption": "DynamicRange",
    "partitionSettings": {
        "partitionColumnName": "<partition_column_name>",
        "partitionUpperBound": "<upper_value_of_partition_column (optional) to decide the partition stride,
not as data filter>",
        "partitionLowerBound": "<lower_value_of_partition_column (optional) to decide the partition stride,
not as data filter>"
    }
}
```

Sample query to check physical partition

```
SELECT DISTINCT s.name AS SchemaName, t.name AS TableName, pf.name AS PartitionFunctionName, c.name AS
ColumnName, iif(pf.name is null, 'no', 'yes') AS HasPartition
FROM sys.tables AS t
LEFT JOIN sys.objects AS o ON t.object_id = o.object_id
LEFT JOIN sys.schemas AS s ON o.schema_id = s.schema_id
LEFT JOIN sys.indexes AS i ON t.object_id = i.object_id
LEFT JOIN sys.index_columns AS ic ON ic.partition_ordinal > 0 AND ic.index_id = i.index_id AND ic.object_id
= t.object_id
LEFT JOIN sys.columns AS c ON c.object_id = ic.object_id AND c.column_id = ic.column_id
LEFT JOIN sys.partition_schemes ps ON i.data_space_id = ps.data_space_id
LEFT JOIN sys.partition_functions pf ON pf.function_id = ps.function_id
WHERE s.name='[your schema]' AND t.name = '[your table name]'
```

If the table has physical partition, you would see "HasPartition" as "yes" like the following.

SchemaName	TableName	PartitionFunctionName	ColumnName	HasPartition
dbo	PerfWithPartition	RangePF1	Id	yes

Best practice for loading data into SQL Managed Instance

When you copy data into SQL Managed Instance, you might require different write behavior:

- Append: My source data has only new records.
- Upsert: My source data has both inserts and updates.
- Overwrite: I want to reload the entire dimension table each time.
- Write with custom logic: I need extra processing before the final insertion into the destination table.

See the respective sections for how to configure and best practices.

Append data

Appending data is the default behavior of the SQL Managed Instance sink connector. The service does a bulk insert to write to your table efficiently. You can configure the source and sink accordingly in the copy activity.

Upsert data

Copy activity now supports natively loading data into a database temporary table and then update the data in sink table if key exists and otherwise insert new data. To learn more about upsert settings in copy activities, see SQL Managed Instance as a sink.

Overwrite the entire table

You can configure the **preCopyScript** property in a copy activity sink. In this case, for each copy activity that runs, the service runs the script first. Then it runs the copy to insert the data. For example, to overwrite the entire table with the latest data, specify a script to first delete all the records before you bulk load the new data from

the source.

Write data with custom logic

The steps to write data with custom logic are similar to those described in the Upsert data section. When you need to apply extra processing before the final insertion of source data into the destination table, you can load to a staging table then invoke stored procedure activity, or invoke a stored procedure in copy activity sink to apply data.

Invoke a stored procedure from a SQL sink

When you copy data into SQL Managed Instance, you also can configure and invoke a user-specified stored procedure with additional parameters on each batch of the source table. The stored procedure feature takes advantage of table-valued parameters.

You can use a stored procedure when built-in copy mechanisms don't serve the purpose. An example is when you want to apply extra processing before the final insertion of source data into the destination table. Some extra processing examples are when you want to merge columns, look up additional values, and insert into more than one table.

The following sample shows how to use a stored procedure to do an upsert into a table in the SQL Server database. Assume that the input data and the sink **Marketing** table each have three columns: **ProfileID**, **State**, and **Category**. Do the upsert based on the **ProfileID** column, and only apply it for a specific category called "ProductA".

1. In your database, define the table type with the same name as **sqlWriterTableType**. The schema of the table type is the same as the schema returned by your input data.

   ```
   CREATE TYPE [dbo].[MarketingType] AS TABLE(
       [ProfileID] [varchar](256) NOT NULL,
       [State] [varchar](256) NOT NULL,
       [Category] [varchar](256) NOT NULL
   )
   ```

2. In your database, define the stored procedure with the same name as **sqlWriterStoredProcedureName**. It handles input data from your specified source and merges into the output table. The parameter name of the table type in the stored procedure is the same as **tableName** defined in the dataset.

   ```
   CREATE PROCEDURE spOverwriteMarketing @Marketing [dbo].[MarketingType] READONLY, @category
   varchar(256)
   AS
   BEGIN
   MERGE [dbo].[Marketing] AS target
   USING @Marketing AS source
   ON (target.ProfileID = source.ProfileID and target.Category = @category)
   WHEN MATCHED THEN
       UPDATE SET State = source.State
   WHEN NOT MATCHED THEN
       INSERT (ProfileID, State, Category)
       VALUES (source.ProfileID, source.State, source.Category);
   END
   ```

3. In your pipeline, define the **SQL MI sink** section in the copy activity as follows:

```
"sink": {
    "type": "SqlMISink",
    "sqlWriterStoredProcedureName": "spOverwriteMarketing",
    "storedProcedureTableTypeParameterName": "Marketing",
    "sqlWriterTableType": "MarketingType",
    "storedProcedureParameters": {
        "category": {
            "value": "ProductA"
        }
    }
}
```

Mapping data flow properties

When transforming data in mapping data flow, you can read and write to tables from Azure SQL Managed Instance. For more information, see the source transformation and sink transformation in mapping data flows.

Source transformation

The below table lists the properties supported by Azure SQL Managed Instance source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Table	If you select Table as input, data flow fetches all the data from the table specified in the dataset.	No	-	-
Query	If you select Query as input, specify a SQL query to fetch data from source, which overrides any table you specify in dataset. Using queries is a great way to reduce rows for testing or lookups. **Order By** clause is not supported, but you can set a full SELECT FROM statement. You can also use user-defined table functions. **select * from udfGetData()** is a UDF in SQL that returns a table that you can use in data flow. Query example: `Select * from MyTable where customerId > 1000 and customerId < 2000`	No	String	query

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Batch size	Specify a batch size to chunk large data into reads.	No	Integer	batchSize
Isolation Level	Choose one of the following isolation levels: - Read Committed - Read Uncommitted (default) - Repeatable Read - Serializable - None (ignore isolation level)	No	READ_COMMITTED READ_UNCOMMITTED REPEATABLE_READ SERIALIZABLE NONE	isolationLevel

TIP

The common table expression (CTE) in SQL is not supported in the mapping data flow **Query** mode, because the prerequisite of using this mode is that queries can be used in the SQL query FROM clause but CTEs cannot do this. To use CTEs, you need to create a stored procedure using the following query:

```
CREATE PROC CTESP @query nvarchar(max)
AS
BEGIN
EXECUTE sp_executesql @query;
END
```

Then use the **Stored procedure** mode in the source transformation of the mapping data flow and set the `@query` like example `with CTE as (select 'test' as a) select * from CTE`. Then you can use CTEs as expected.

Azure SQL Managed Instance source script example

When you use Azure SQL Managed Instance as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
    validateSchema: false,
    isolationLevel: 'READ_UNCOMMITTED',
    query: 'select * from MYTABLE',
    format: 'query') ~> SQLMISource
```

Sink transformation

The below table lists the properties supported by Azure SQL Managed Instance sink. You can edit these properties in the **Sink options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Update method	Specify what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an Alter row transformation is required to tag rows for those actions.	Yes	`true` or `false`	deletable insertable updateable upsertable
Key columns	For updates, upserts and deletes, key column(s) must be set to determine which row to alter. The column name that you pick as the key will be used as part of the subsequent update, upsert, delete. Therefore, you must pick a column that exists in the Sink mapping.	No	Array	keys
Skip writing key columns	If you wish to not write the value to the key column, select "Skip writing key columns".	No	`true` or `false`	skipKeyWrites
Table action	Determines whether to recreate or remove all rows from the destination table prior to writing. - **None**: No action will be done to the table. - **Recreate**: The table will get dropped and recreated. Required if creating a new table dynamically. - **Truncate**: All rows from the target table will get removed.	No	`true` or `false`	recreate truncate

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Batch size	Specify how many rows are being written in each batch. Larger batch sizes improve compression and memory optimization, but risk out of memory exceptions when caching data.	No	Integer	batchSize
Pre and Post SQL scripts	Specify multi-line SQL scripts that will execute before (pre-processing) and after (post-processing) data is written to your Sink database.	No	String	preSQLs postSQLs

TIP

1. It's recommended to break single batch scripts with multiple commands into multiple batches.
2. Only Data Definition Language (DDL) and Data Manipulation Language (DML) statements that return a simple update count can be run as part of a batch. Learn more from Performing batch operations

Azure SQL Managed Instance sink script example

When you use Azure SQL Managed Instance as sink type, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    deletable:false,
    insertable:true,
    updateable:true,
    upsertable:true,
    keys:['keyColumn'],
    format: 'table',
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> SQLMISink
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity

Data type mapping for SQL Managed Instance

When data is copied to and from SQL Managed Instance using copy activity, the following mappings are used from SQL Managed Instance data types to interim data types used internally within the service. To learn how the copy activity maps from the source schema and data type to the sink, see Schema and data type mappings.

SQL MANAGED INSTANCE DATA TYPE	INTERIM SERVICE DATA TYPE
bigint	Int64
binary	Byte[]
bit	Boolean
char	String, Char[]
date	DateTime
Datetime	DateTime
datetime2	DateTime
Datetimeoffset	DateTimeOffset
Decimal	Decimal
FILESTREAM attribute (varbinary(max))	Byte[]
Float	Double
image	Byte[]
int	Int32
money	Decimal
nchar	String, Char[]
ntext	String, Char[]
numeric	Decimal
nvarchar	String, Char[]
real	Single
rowversion	Byte[]
smalldatetime	DateTime
smallint	Int16
smallmoney	Decimal
sql_variant	Object
text	String, Char[]

SQL MANAGED INSTANCE DATA TYPE	INTERIM SERVICE DATA TYPE
time	TimeSpan
timestamp	Byte[]
tinyint	Int16
uniqueidentifier	Guid
varbinary	Byte[]
varchar	String, Char[]
xml	String

> **NOTE**
> For data types that map to the Decimal interim type, currently Copy activity supports precision up to 28. If you have data that requires precision larger than 28, consider converting to a string in a SQL query.

Using Always Encrypted

When you copy data from/to SQL Managed Instance with Always Encrypted, follow below steps:

1. Store the Column Master Key (CMK) in an Azure Key Vault. Learn more on how to configure Always Encrypted by using Azure Key Vault
2. Make sure to great access to the key vault where the Column Master Key (CMK) is stored. Refer to this article for required permissions.
3. Create linked service to connect to your SQL database and enable 'Always Encrypted' function by using either managed identity or service principal.

> **NOTE**
> SQL Managed Instance Always Encrypted supports below scenarios:
>
> 1. Either source or sink data stores is using managed identity or service principal as key provider authentication type.
> 2. Both source and sink data stores are using managed identity as key provider authentication type.
> 3. Both source and sink data stores are using the same service principal as key provider authentication type.

> **NOTE**
> Currently, SQL Managed Instance **Always Encrypted** is only supported for source transformation in mapping data flows.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy and transform data in Azure Synapse Analytics by using Azure Data Factory or Synapse pipelines

8/19/2022 • 40 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory or Synapse pipelines to copy data from and to Azure Synapse Analytics, and use Data Flow to transform data in Azure Data Lake Storage Gen2. To learn about Azure Data Factory, read the introductory article.

Supported capabilities

This Azure Synapse Analytics connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/sink)	① ②	✓
Mapping data flow (source/sink)	①	✓
Lookup activity	① ②	✓
GetMetadata activity	① ②	✓
Script activity	① ②	✓
Stored procedure activity	① ②	✓

① Azure integration runtime ② Self-hosted integration runtime

For Copy activity, this Azure Synapse Analytics connector supports these functions:

- Copy data by using SQL authentication and Azure Active Directory (Azure AD) Application token authentication with a service principal or managed identities for Azure resources.
- As a source, retrieve data by using a SQL query or stored procedure. You can also choose to parallel copy from an Azure Synapse Analytics source, see the Parallel copy from Azure Synapse Analytics section for details.
- As a sink, load data by using COPY statement or PolyBase or bulk insert. We recommend COPY statement or PolyBase for better copy performance. The connector also supports automatically creating destination table with DISTRIBUTION = ROUND_ROBIN if not exists based on the source schema.

> **IMPORTANT**
> If you copy data by using an Azure Integration Runtime, configure a server-level firewall rule so that Azure services can access the logical SQL server. If you copy data by using a self-hosted integration runtime, configure the firewall to allow the appropriate IP range. This range includes the machine's IP that is used to connect to Azure Synapse Analytics.

Get started

> **TIP**
>
> To achieve best performance, use PolyBase or COPY statement to load data into Azure Synapse Analytics. The Use PolyBase to load data into Azure Synapse Analytics and Use COPY statement to load data into Azure Synapse Analytics sections have details. For a walkthrough with a use case, see Load 1 TB into Azure Synapse Analytics under 15 minutes with Azure Data Factory.

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

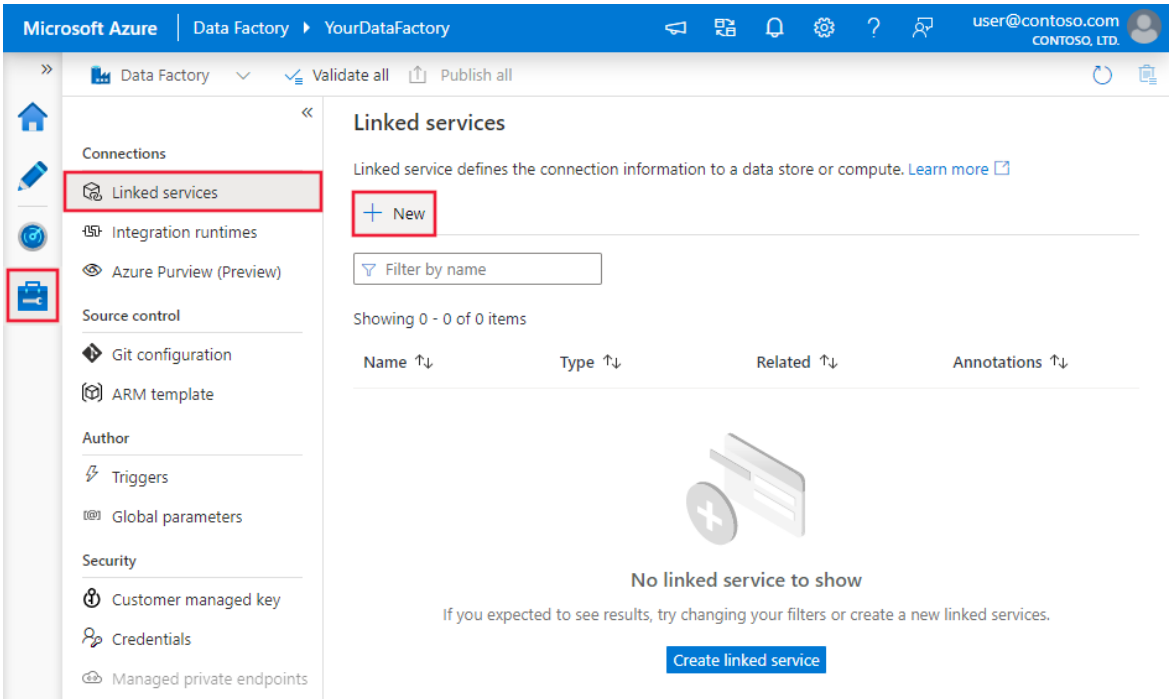
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create an Azure Synapse Analytics linked service using UI

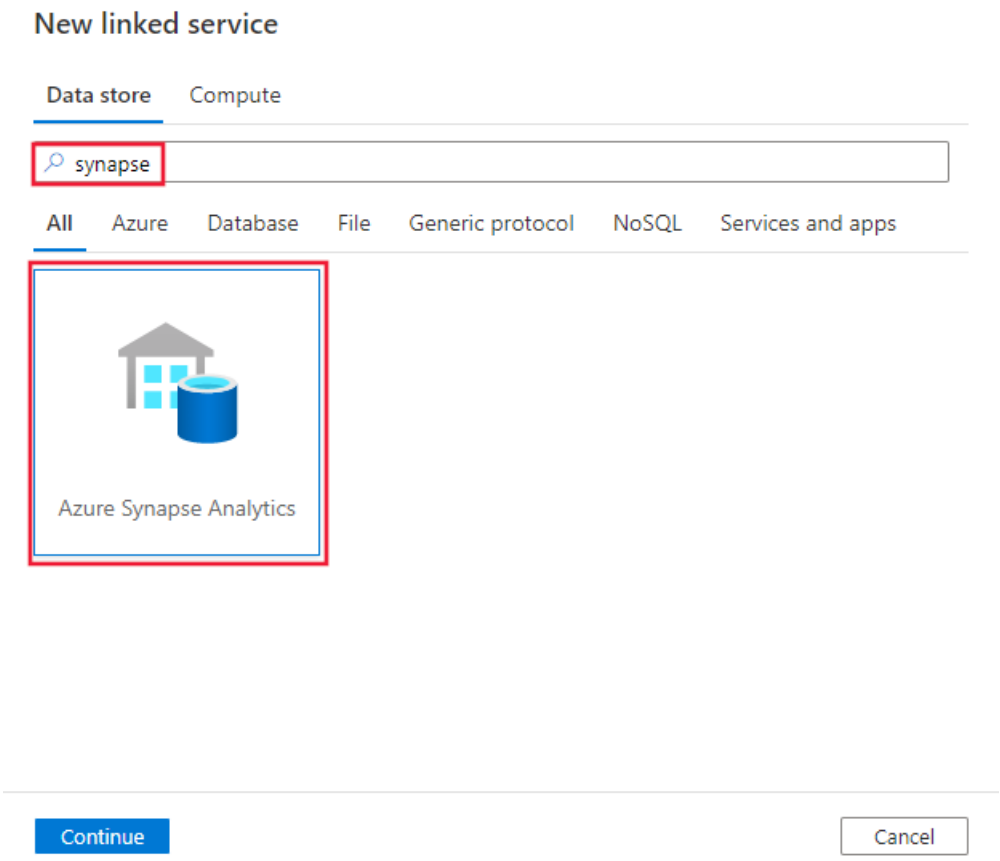
Use the following steps to create an Azure Synapse Analytics linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

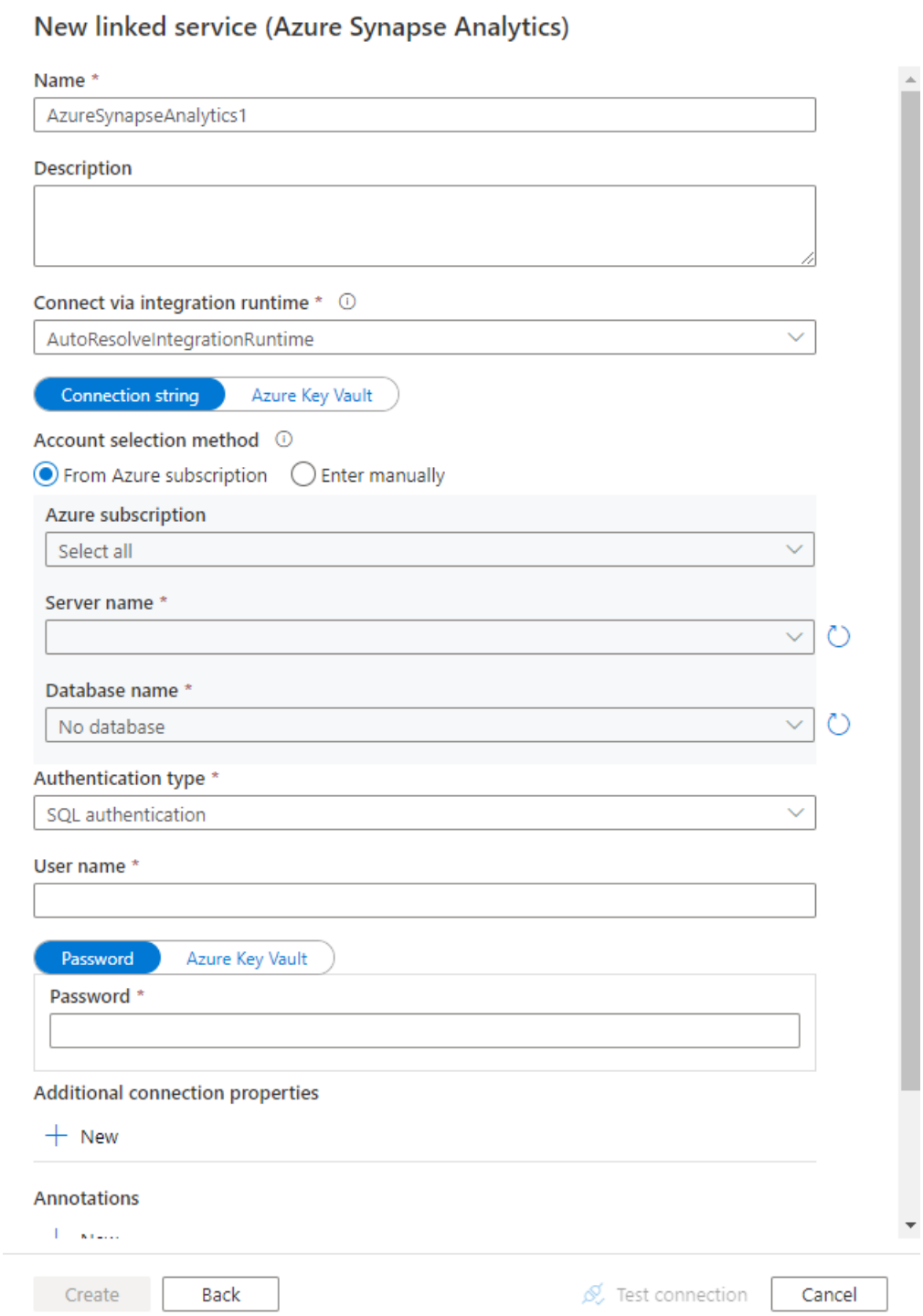
   - Azure Data Factory
   - Azure Synapse



2. Search for Synapse and select the Azure Synapse Analytics connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that define Data Factory and Synapse pipeline entities specific to an Azure Synapse Analytics connector.

Linked service properties

These generic properties are supported for an Azure Synapse Analytics linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AzureSqlDW**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	Specify the information needed to connect to the Azure Synapse Analytics instance for the **connectionString** property. Mark this field as a SecureString to store it securely. You can also put password/service principal key in Azure Key Vault, and if it's SQL authentication pull the `password` configuration out of the connection string. See the JSON example below the table and Store credentials in Azure Key Vault article with more details.	Yes
azureCloudType	For service principal authentication, specify the type of Azure cloud environment to which your Azure AD application is registered. Allowed values are `AzurePublic`, `AzureChina`, `AzureUsGovernment`, and `AzureGermany`. By default, the data factory or Synapse pipeline's cloud environment is used.	No
connectVia	The integration runtime to be used to connect to the data store. You can use Azure Integration Runtime or a self-hosted integration runtime (if your data store is located in a private network). If not specified, it uses the default Azure Integration Runtime.	No

For different authentication types, refer to the following sections on specific properties, prerequisites and JSON samples, respectively:

- SQL authentication
- Service principal authentication
- System-assigned managed identity authentication
- User-assigned managed identity authentication

> **TIP**
> When creating linked service for Azure Synapse **serverless** SQL pool from UI, choose "enter manually" instead of browsing from subscription.

> **TIP**
> If you hit error with error code as "UserErrorFailedToConnectToSqlServer" and message like "The session limit for the database is XXX and has been reached.", add `Pooling=false` to your connection string and try again.

SQL authentication

To use SQL authentication authentication type, specify the generic properties that are described in the preceding section.

Linked service example that uses SQL authentication

```
{
    "name": "AzureSqlDWLinkedService",
    "properties": {
        "type": "AzureSqlDW",
        "typeProperties": {
            "connectionString": "Server=tcp:<servername>.database.windows.net,1433;Database=
<databasename>;User ID=<username>@<servername>;Password=
<password>;Trusted_Connection=False;Encrypt=True;Connection Timeout=30"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Password in Azure Key Vault:

```
{
    "name": "AzureSqlDWLinkedService",
    "properties": {
        "type": "AzureSqlDW",
        "typeProperties": {
            "connectionString": "Server=tcp:<servername>.database.windows.net,1433;Database=
<databasename>;User ID=<username>@<servername>;Trusted_Connection=False;Encrypt=True;Connection Timeout=30",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Service principal authentication

To use service principal authentication, in addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. You can retrieve it by hovering the mouse in the top-right corner of the Azure portal.	Yes

You also need to follow the steps below:

1. **Create an Azure Active Directory application** from the Azure portal. Make note of the application name and the following values that define the linked service:

   - Application ID
   - Application key
   - Tenant ID

2. **Provision an Azure Active Directory administrator** for your server in the Azure portal if you haven't already done so. The Azure AD administrator can be an Azure AD user or Azure AD group. If you grant the group with managed identity an admin role, skip steps 3 and 4. The administrator will have full access to the database.

3. **Create contained database users** for the service principal. Connect to the data warehouse from or to which you want to copy data by using tools like SSMS, with an Azure AD identity that has at least ALTER ANY USER permission. Run the following T-SQL:

   ```
   CREATE USER [your_application_name] FROM EXTERNAL PROVIDER;
   ```

4. **Grant the service principal needed permissions** as you normally do for SQL users or others. Run the following code, or refer to more options here. If you want to use PolyBase to load the data, learn the required database permission.

   ```
   EXEC sp_addrolemember db_owner, [your application name];
   ```

5. **Configure an Azure Synapse Analytics linked service** in an Azure Data Factory or Synapse workspace.

Linked service example that uses service principal authentication

```
{
    "name": "AzureSqlDWLinkedService",
    "properties": {
        "type": "AzureSqlDW",
        "typeProperties": {
            "connectionString": "Server=tcp:<servername>.database.windows.net,1433;Database=
<databasename>;Connection Timeout=30",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

System-assigned managed identities for Azure resources authentication

A data factory or Synapse workspace can be associated with a system-assigned managed identity for Azure resources that represents the resource. You can use this managed identity for Azure Synapse Analytics authentication. The designated resource can access and copy data from or to your data warehouse by using this identity.

To use system-assigned managed identity authentication, specify the generic properties that are described in the

preceding section, and follow these steps.

1. Provision an Azure Active Directory administrator for your server on the Azure portal if you haven't already done so. The Azure AD administrator can be an Azure AD user or Azure AD group. If you grant the group with system-assigned managed identity an admin role, skip steps 3 and 4. The administrator will have full access to the database.
2. Create contained database users for the system-assigned managed identity. Connect to the data warehouse from or to which you want to copy data by using tools like SSMS, with an Azure AD identity that has at least ALTER ANY USER permission. Run the following T-SQL.

   ```
   CREATE USER [your_resource_name] FROM EXTERNAL PROVIDER;
   ```

3. **Grant the system-assigned managed identity needed permissions** as you normally do for SQL users and others. Run the following code, or refer to more options here. If you want to use PolyBase to load the data, learn the required database permission.

   ```
   EXEC sp_addrolemember db_owner, [your_resource_name];
   ```

4. **Configure an Azure Synapse Analytics linked service**.

Example:

```
{
    "name": "AzureSqlDWLinkedService",
    "properties": {
        "type": "AzureSqlDW",
        "typeProperties": {
            "connectionString": "Server=tcp:<servername>.database.windows.net,1433;Database=
<databasename>;Connection Timeout=30"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

User-assigned managed identity authentication

A data factory or Synapse workspace can be associated with a user-assigned managed identities that represents the resource. You can use this managed identity for Azure Synapse Analytics authentication. The designated resource can access and copy data from or to your data warehouse by using this identity.

To use user-assigned managed identity authentication, in addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
credentials	Specify the user-assigned managed identity as the credential object.	Yes

You also need to follow the steps below:

1. Provision an Azure Active Directory administrator for your server on the Azure portal if you haven't already done so. The Azure AD administrator can be an Azure AD user or Azure AD group. If you grant the group with user-assigned managed identity an admin role, skip steps 3. The administrator will

have full access to the database.

2. **Create contained database users** for the user-assigned managed identity. Connect to the data warehouse from or to which you want to copy data by using tools like SSMS, with an Azure AD identity that has at least ALTER ANY USER permission. Run the following T-SQL.

   ```
   CREATE USER [your_resource_name] FROM EXTERNAL PROVIDER;
   ```

3. Create one or multiple user-assigned managed identities and **grant the user-assigned managed identity needed permissions** as you normally do for SQL users and others. Run the following code, or refer to more options here. If you want to use PolyBase to load the data, learn the required database permission.

   ```
   EXEC sp_addrolemember db_owner, [your_resource_name];
   ```

4. Assign one or multiple user-assigned managed identities to your data factory and create credentials for each user-assigned managed identity.

5. **Configure an Azure Synapse Analytics linked service**.

Example:

```
{
    "name": "AzureSqlDWLinkedService",
    "properties": {
        "type": "AzureSqlDW",
        "typeProperties": {
            "connectionString": "Server=tcp:<servername>.database.windows.net,1433;Database=
<databasename>;Connection Timeout=30",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article.

The following properties are supported for Azure Synapse Analytics dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the dataset must be set to **AzureSqlDWTable**.	Yes
schema	Name of the schema.	No for source, Yes for sink
table	Name of the table/view.	No for source, Yes for sink

PROPERTY	DESCRIPTION	REQUIRED
tableName	Name of the table/view with schema. This property is supported for backward compatibility. For new workload, use `schema` and `table`.	No for source, Yes for sink

Dataset properties example

```
{
    "name": "AzureSQLDWDataset",
    "properties":
    {
        "type": "AzureSqlDWTable",
        "linkedServiceName": {
            "referenceName": "<Azure Synapse Analytics linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "typeProperties": {
            "schema": "<schema_name>",
            "table": "<table_name>"
        }
    }
}
```

Copy Activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the Azure Synapse Analytics source and sink.

Azure Synapse Analytics as the source

> **TIP**
> To load data from Azure Synapse Analytics efficiently by using data partitioning, learn more from Parallel copy from Azure Synapse Analytics.

To copy data from Azure Synapse Analytics, set the **type** property in the Copy Activity source to **SqlDWSource**. The following properties are supported in the Copy Activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the Copy Activity source must be set to **SqlDWSource**.	Yes
sqlReaderQuery	Use the custom SQL query to read data. Example: `select * from MyTable`.	No
sqlReaderStoredProcedureName	The name of the stored procedure that reads data from the source table. The last SQL statement must be a SELECT statement in the stored procedure.	No

PROPERTY	DESCRIPTION	REQUIRED
storedProcedureParameters	Parameters for the stored procedure. Allowed values are name or value pairs. Names and casing of parameters must match the names and casing of the stored procedure parameters.	No
isolationLevel	Specifies the transaction locking behavior for the SQL source. The allowed values are: **ReadCommitted**, **ReadUncommitted**, **RepeatableRead**, **Serializable**, **Snapshot**. If not specified, the database's default isolation level is used. For more information, see system.data.isolationlevel.	No
partitionOptions	Specifies the data partitioning options used to load data from Azure Synapse Analytics. Allowed values are: **None** (default), **PhysicalPartitionsOfTable**, and **DynamicRange**. When a partition option is enabled (that is, not `None`), the degree of parallelism to concurrently load data from an Azure Synapse Analytics is controlled by the `parallelCopies` setting on the copy activity.	No
partitionSettings	Specify the group of the settings for data partitioning. Apply when the partition option isn't `None`.	No
Under `partitionSettings`***:***		
partitionColumnName	Specify the name of the source column **in integer or date/datetime type** (`int`, `smallint`, `bigint`, `date`, `smalldatetime`, `datetime`, `datetime2`, or `datetimeoffset`) that will be used by range partitioning for parallel copy. If not specified, the index or the primary key of the table is detected automatically and used as the partition column. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfDynamicRangePartitionCondition` in the WHERE clause. For an example, see the Parallel copy from SQL database section.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionUpperBound	The maximum value of the partition column for partition range splitting. This value is used to decide the partition stride, not for filtering the rows in table. All rows in the table or query result will be partitioned and copied. If not specified, copy activity auto detect the value. Apply when the partition option is `DynamicRange`. For an example, see the Parallel copy from SQL database section.	No
partitionLowerBound	The minimum value of the partition column for partition range splitting. This value is used to decide the partition stride, not for filtering the rows in table. All rows in the table or query result will be partitioned and copied. If not specified, copy activity auto detect the value. Apply when the partition option is `DynamicRange`. For an example, see the Parallel copy from SQL database section.	No

Note the following point:

- When using stored procedure in source to retrieve data, note if your stored procedure is designed as returning different schema when different parameter value is passed in, you may encounter failure or see unexpected result when importing schema from UI or when copying data to SQL database with auto table creation.

Example: using SQL query

```
"activities":[
    {
        "name": "CopyFromAzureSQLDW",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Azure Synapse Analytics input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SqlDWSource",
                "sqlReaderQuery": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Example: using stored procedure

```
"activities":[
    {
        "name": "CopyFromAzureSQLDW",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Azure Synapse Analytics input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SqlDWSource",
                "sqlReaderStoredProcedureName": "CopyTestSrcStoredProcedureWithParameters",
                "storedProcedureParameters": {
                    "stringData": { "value": "str3" },
                    "identifier": { "value": "$$Text.Format('{0:yyyy}', <datetime parameter>)", "type":
"Int"}
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Sample stored procedure:

```
CREATE PROCEDURE CopyTestSrcStoredProcedureWithParameters
(
    @stringData varchar(20),
    @identifier int
)
AS
SET NOCOUNT ON;
BEGIN
    select *
    from dbo.UnitTestSrcTable
    where dbo.UnitTestSrcTable.stringData != stringData
    and dbo.UnitTestSrcTable.identifier != identifier
END
GO
```

Azure Synapse Analytics as sink

Azure Data Factory and Synapse pipelines support three ways to load data into Azure Synapse Analytics.

- Use COPY statement
- Use PolyBase
- Use bulk insert

The fastest and most scalable way to load data is through the COPY statement or the PolyBase.

To copy data to Azure Synapse Analytics, set the sink type in Copy Activity to **SqlDWSink**. The following properties are supported in the Copy Activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the Copy Activity sink must be set to **SqlDWSink**.	Yes
allowPolyBase	Indicates whether to use PolyBase to load data into Azure Synapse Analytics. `allowCopyCommand` and `allowPolyBase` cannot be both true. See Use PolyBase to load data into Azure Synapse Analytics section for constraints and details. Allowed values are **True** and **False** (default).	No. Apply when using PolyBase.
polyBaseSettings	A group of properties that can be specified when the `allowPolybase` property is set to **true**.	No. Apply when using PolyBase.

PROPERTY	DESCRIPTION	REQUIRED
allowCopyCommand	Indicates whether to use COPY statement to load data into Azure Synapse Analytics. `allowCopyCommand` and `allowPolyBase` cannot be both true. See Use COPY statement to load data into Azure Synapse Analytics section for constraints and details. Allowed values are **True** and **False** (default).	No. Apply when using COPY.
copyCommandSettings	A group of properties that can be specified when `allowCopyCommand` property is set to TRUE.	No. Apply when using COPY.
writeBatchSize	Number of rows to inserts into the SQL table **per batch**. The allowed value is **integer** (number of rows). By default, the service dynamically determines the appropriate batch size based on the row size.	No. Apply when using bulk insert.
writeBatchTimeout	Wait time for the batch insert operation to finish before it times out. The allowed value is **timespan**. Example: "00:30:00" (30 minutes).	No. Apply when using bulk insert.
preCopyScript	Specify a SQL query for Copy Activity to run before writing data into Azure Synapse Analytics in each run. Use this property to clean up the preloaded data.	No
tableOption	Specifies whether to automatically create the sink table if not exists based on the source schema. Allowed values are: `none` (default), `autoCreate`.	No
disableMetricsCollection	The service collects metrics such as Azure Synapse Analytics DWUs for copy performance optimization and recommendations, which introduce additional master DB access. If you are concerned with this behavior, specify `true` to turn it off.	No (default is `false`)
maxConcurrentConnections	The upper limit of concurrent connecti ons established to the data store durin g the activity run. Specify a value only when you want to limit concurrent con nections.	No

PROPERTY	DESCRIPTION	REQUIRED
WriteBehavior	Specify the write behavior for copy activity to load data into Azure SQL Database. The allowed value is **Insert** and **Upsert**. By default, the service uses insert to load data.	No
upsertSettings	Specify the group of the settings for write behavior. Apply when the WriteBehavior option is `Upert`.	No
Under `upsertSettings`***:***		
keys	Specify the column names for unique row identification. Either a single key or a series of keys can be used. If not specified, the primary key is used.	No
interimSchemaName	Specify the interim schema for creating interim table. Note: user need to have the permission for creating and deleting table. By default, interim table will share the same schema as sink table.	No

Example 1: Azure Synapse Analytics sink

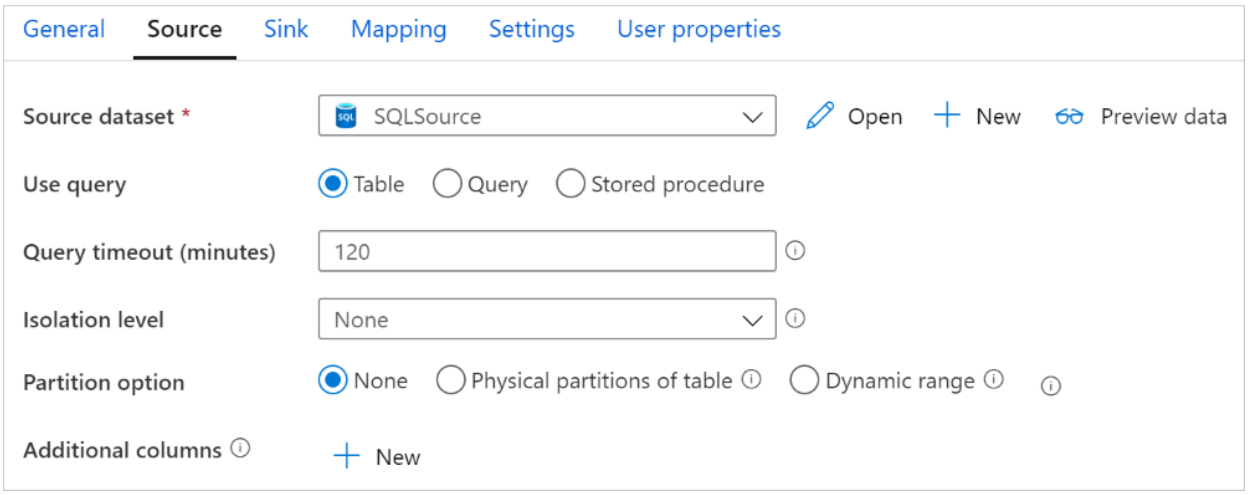
```
"sink": {
    "type": "SqlDWSink",
    "allowPolyBase": true,
    "polyBaseSettings":
    {
        "rejectType": "percentage",
        "rejectValue": 10.0,
        "rejectSampleValue": 100,
        "useTypeDefault": true
    }
}
```

Example 2: Upsert data

```
"sink": {
    "type": "SqlDWSink",
    "writeBehavior": "Upsert",
    "upsertSettings": {
        "keys": [
             "<column name>"
        ],
        "interimSchemaName": "<interim schema name>"
    },
}
```

Parallel copy from Azure Synapse Analytics

The Azure Synapse Analytics connector in copy activity provides built-in data partitioning to copy data in parallel. You can find data partitioning options on the **Source** tab of the copy activity.



When you enable partitioned copy, copy activity runs parallel queries against your Azure Synapse Analytics source to load data by partitions. The parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your Azure Synapse Analytics.

You are suggested to enable parallel copy with data partitioning especially when you load large amount of data from your Azure Synapse Analytics. The following are suggested configurations for different scenarios. When copying data into file-based data store, it's recommended to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

SCENARIO	SUGGESTED SETTINGS
Full load from large table, with physical partitions.	**Partition option**: Physical partitions of table. During execution, the service automatically detects the physical partitions, and copies data by partitions. To check if your table has physical partition or not, you can refer to this query.
Full load from large table, without physical partitions, while with an integer or datetime column for data partitioning.	**Partition options**: Dynamic range partition. **Partition column** (optional): Specify the column used to partition data. If not specified, the index or primary key column is used. **Partition upper bound** and **partition lower bound** (optional): Specify if you want to determine the partition stride. This is not for filtering the rows in table, all rows in the table will be partitioned and copied. If not specified, copy activity auto detect the values. For example, if your partition column "ID" has values range from 1 to 100, and you set the lower bound as 20 and the upper bound as 80, with parallel copy as 4, the service retrieves data by 4 partitions - IDs in range <=20, [21, 50], [51, 80], and >=81, respectively.

SCENARIO	SUGGESTED SETTINGS

Load a large amount of data by using a custom query, without physical partitions, while with an integer or date/datetime column for data partitioning.	**Partition options**: Dynamic range partition.
	Query:
	`SELECT * FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	.
	Partition column: Specify the column used to partition data.
	Partition upper bound and **partition lower bound** (optional): Specify if you want to determine the partition stride. This is not for filtering the rows in table, all rows in the query result will be partitioned and copied. If not specified, copy activity auto detect the value.
	
	During execution, the service replaces `?AdfRangePartitionColumnName` with the actual column name and value ranges for each partition, and sends to Azure Synapse Analytics.
	For example, if your partition column "ID" has values range from 1 to 100, and you set the lower bound as 20 and the upper bound as 80, with parallel copy as 4, the service retrieves data by 4 partitions- IDs in range <=20, [21, 50], [51, 80], and >=81, respectively.
	
	Here are more sample queries for different scenarios:
	1. Query the whole table:
	`SELECT * FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition`
	2. Query from a table with column selection and additional where-clause filters:
	`SELECT <column_list> FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	3. Query with subqueries:
	`SELECT <column_list> FROM (<your_sub_query>) AS T WHERE ?AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	4. Query with partition in subquery:
	`SELECT <column_list> FROM (SELECT <your_sub_query_column_list> FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition) AS T`

Best practices to load data with partition option:

1. Choose distinctive column as partition column (like primary key or unique key) to avoid data skew.
2. If the table has built-in partition, use partition option "Physical partitions of table" to get better performance.
3. If you use Azure Integration Runtime to copy data, you can set larger "Data Integration Units (DIU)" (>4) to utilize more computing resource. Check the applicable scenarios there.
4. "Degree of copy parallelism" control the partition numbers, setting this number too large sometime hurts the performance, recommend setting this number as (DIU or number of Self-hosted IR nodes) * (2 to 4).
5. Note Azure Synapse Analytics can execute a maximum of 32 queries at a moment, setting "Degree of copy parallelism" too large may cause a Synapse throttling issue.

Example: full load from large table with physical partitions

```
"source": {
    "type": "SqlDWSource",
    "partitionOption": "PhysicalPartitionsOfTable"
}
```

Example: query with dynamic range partition

```
"source": {
    "type": "SqlDWSource",
    "query":"SELECT * FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition AND
<your_additional_where_clause>",
    "partitionOption": "DynamicRange",
    "partitionSettings": {
        "partitionColumnName": "<partition_column_name>",
        "partitionUpperBound": "<upper_value_of_partition_column (optional) to decide the partition stride,
not as data filter>",
        "partitionLowerBound": "<lower_value_of_partition_column (optional) to decide the partition stride,
not as data filter>"
    }
}
```

Sample query to check physical partition

```
SELECT DISTINCT s.name AS SchemaName, t.name AS TableName, c.name AS ColumnName, CASE WHEN c.name IS NULL
THEN 'no' ELSE 'yes' END AS HasPartition
FROM sys.tables AS t
LEFT JOIN sys.objects AS o ON t.object_id = o.object_id
LEFT JOIN sys.schemas AS s ON o.schema_id = s.schema_id
LEFT JOIN sys.indexes AS i ON t.object_id = i.object_id
LEFT JOIN sys.index_columns AS ic ON ic.partition_ordinal > 0 AND ic.index_id = i.index_id AND ic.object_id
= t.object_id
LEFT JOIN sys.columns AS c ON c.object_id = ic.object_id AND c.column_id = ic.column_id
LEFT JOIN sys.types AS y ON c.system_type_id = y.system_type_id
WHERE s.name='[your schema]' AND t.name = '[your table name]'
```

If the table has physical partition, you would see "HasPartition" as "yes".

Use COPY statement to load data into Azure Synapse Analytics

Using COPY statement is a simple and flexible way to load data into Azure Synapse Analytics with high throughput. To learn more details, check Bulk load data using the COPY statement

- If your source data is in **Azure Blob or Azure Data Lake Storage Gen2**, and the **format is COPY statement compatible**, you can use copy activity to directly invoke COPY statement to let Azure Synapse Analytics pull the data from source. For details, see **Direct copy by using COPY statement**.
- If your source data store and format isn't originally supported by COPY statement, use the **Staged copy by using COPY statement** feature instead. The staged copy feature also provides you better throughput. It automatically converts the data into COPY statement compatible format, stores the data in Azure Blob storage, then calls COPY statement to load data into Azure Synapse Analytics.

> **TIP**
> When using COPY statement with Azure Integration Runtime, effective Data Integration Units (DIU) is always 2. Tuning the DIU doesn't impact the performance, as loading data from storage is powered by the Azure Synapse engine.

Direct copy by using COPY statement

Azure Synapse Analytics COPY statement directly supports Azure Blob, Azure Data Lake Storage Gen1 and

Azure Data Lake Storage Gen2. If your source data meets the criteria described in this section, use COPY statement to copy directly from the source data store to Azure Synapse Analytics. Otherwise, use Staged copy by using COPY statement. the service checks the settings and fails the copy activity run if the criteria is not met.

1. The **source linked service and format** are with the following types and authentication methods:

SUPPORTED SOURCE DATA STORE TYPE	SUPPORTED FORMAT	SUPPORTED SOURCE AUTHENTICATION TYPE
Azure Blob	Delimited text	Account key authentication, shared access signature authentication, service principal authentication, managed identity authentication
	Parquet	Account key authentication, shared access signature authentication
	ORC	Account key authentication, shared access signature authentication
Azure Data Lake Storage Gen2	Delimited text Parquet ORC	Account key authentication, service principal authentication, managed identity authentication

> **IMPORTANT**
> - When you use managed identity authentication for your storage linked service, learn the needed configurations for Azure Blob and Azure Data Lake Storage Gen2 respectively.
> - If your Azure Storage is configured with VNet service endpoint, you must use managed identity authentication with "allow trusted Microsoft service" enabled on storage account, refer to Impact of using VNet Service Endpoints with Azure storage.

2. Format settings are with the following:
   a. For **Parquet**: `compression` can be **no compression**, **Snappy**, or `GZip`.
   b. For **ORC**: `compression` can be **no compression**, `zlib`, or **Snappy**.
   c. For **Delimited text**:
      a. `rowDelimiter` is explicitly set as **single character** or "**\r\n**", the default value is not supported.
      b. `nullValue` is left as default or set to **empty string** ("").
      c. `encodingName` is left as default or set to **utf-8 or utf-16**.
      d. `escapeChar` must be same as `quoteChar`, and is not empty.
      e. `skipLineCount` is left as default or set to 0.
      f. `compression` can be **no compression** or `GZip`.
3. If your source is a folder, `recursive` in copy activity must be set to true, and `wildcardFilename` need to be `*` or `*.*`.
4. `wildcardFolderPath`, `wildcardFilename` (other than `*` or `*.*`), `modifiedDateTimeStart`, `modifiedDateTimeEnd`, `prefix`, `enablePartitionDiscovery` and `additionalColumns` are not specified.

The following COPY statement settings are supported under `allowCopyCommand` in copy activity:

PROPERTY	DESCRIPTION	REQUIRED
defaultValues	Specifies the default values for each target column in Azure Synapse Analytics. The default values in the property overwrite the DEFAULT constraint set in the data warehouse, and identity column cannot have a default value.	No
additionalOptions	Additional options that will be passed to an Azure Synapse Analytics COPY statement directly in "With" clause in COPY statement. Quote the value as needed to align with the COPY statement requirements.	No

```
"activities":[
    {
        "name": "CopyFromAzureBlobToSQLDataWarehouseViaCOPY",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "ParquetDataset",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "AzureSQLDWDataset",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "ParquetSource",
                "storeSettings":{
                    "type": "AzureBlobStorageReadSettings",
                    "recursive": true
                }
            },
            "sink": {
                "type": "SqlDWSink",
                "allowCopyCommand":true,
                "copyCommandSettings":{
                    "defaultValues":[
                        {
                            "columnName":"col_string",
                            "defaultValue":"DefaultStringValue"
                        }
                    ],
                    "additionalOptions":{
                        "MAXERRORS":"10000",
                        "DATEFORMAT":"'ymd'"
                    }
                }
            },
            "enableSkipIncompatibleRow": true
        }
    }
]
```

Staged copy by using COPY statement

When your source data is not natively compatible with COPY statement, enable data copying via an interim staging Azure Blob or Azure Data Lake Storage Gen2 (it can't be Azure Premium Storage). In this case, the service automatically converts the data to meet the data format requirements of COPY statement. Then it invokes COPY statement to load data into Azure Synapse Analytics. Finally, it cleans up your temporary data from the storage. See Staged copy for details about copying data via a staging.

To use this feature, create an Azure Blob Storage linked service or Azure Data Lake Storage Gen2 linked service with **account key or system-managed identity authentication** that refers to the Azure storage account as the interim storage.

> **IMPORTANT**
>
> - When you use managed identity authentication for your staging linked service, learn the needed configurations for Azure Blob and Azure Data Lake Storage Gen2 respectively.
> - If your staging Azure Storage is configured with VNet service endpoint, you must use managed identity authentication with "allow trusted Microsoft service" enabled on storage account, refer to Impact of using VNet Service Endpoints with Azure storage.

> **IMPORTANT**
>
> If your staging Azure Storage is configured with Managed Private Endpoint and has the storage firewall enabled, you must use managed identity authentication and grant Storage Blob Data Reader permissions to the Synapse SQL Server to ensure it can access the staged files during the COPY statement load.

```
"activities":[
    {
        "name": "CopyFromSQLServerToSQLDataWarehouseViaCOPYstatement",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "SQLServerDataset",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "AzureSQLDWDataset",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SqlSource",
            },
            "sink": {
                "type": "SqlDWSink",
                "allowCopyCommand": true
            },
            "stagingSettings": {
                "linkedServiceName": {
                    "referenceName": "MyStagingStorage",
                    "type": "LinkedServiceReference"
                }
            }
        }
    }
]
```

Use PolyBase to load data into Azure Synapse Analytics

Using PolyBase is an efficient way to load a large amount of data into Azure Synapse Analytics with high throughput. You'll see a large gain in the throughput by using PolyBase instead of the default BULKINSERT mechanism. For a walkthrough with a use case, see Load 1 TB into Azure Synapse Analytics.

- If your source data is in **Azure Blob, Azure Data Lake Storage Gen1 or Azure Data Lake Storage Gen2**, and the **format is PolyBase compatible**, you can use copy activity to directly invoke PolyBase to let Azure Synapse Analytics pull the data from source. For details, see **Direct copy by using PolyBase**.
- If your source data store and format isn't originally supported by PolyBase, use the **Staged copy by using PolyBase** feature instead. The staged copy feature also provides you better throughput. It automatically converts the data into PolyBase-compatible format, stores the data in Azure Blob storage, then calls PolyBase to load data into Azure Synapse Analytics.

> **TIP**
> Learn more on Best practices for using PolyBase. When using PolyBase with Azure Integration Runtime, effective Data Integration Units (DIU) for direct or staged storage-to-Synapse is always 2. Tuning the DIU doesn't impact the performance, as loading data from storage is powered by Synapse engine.

The following PolyBase settings are supported under `polyBaseSettings` in copy activity:

PROPERTY	DESCRIPTION	REQUIRED
rejectValue	Specifies the number or percentage of rows that can be rejected before the query fails. Learn more about PolyBase's reject options in the Arguments section of CREATE EXTERNAL TABLE (Transact-SQL). Allowed values are 0 (default), 1, 2, etc.	No
rejectType	Specifies whether the **rejectValue** option is a literal value or a percentage. Allowed values are **Value** (default) and **Percentage**.	No
rejectSampleValue	Determines the number of rows to retrieve before PolyBase recalculates the percentage of rejected rows. Allowed values are 1, 2, etc.	Yes, if the **rejectType** is **percentage**.

PROPERTY	DESCRIPTION	REQUIRED
useTypeDefault	Specifies how to handle missing values in delimited text files when PolyBase retrieves data from the text file. Learn more about this property from the Arguments section in CREATE EXTERNAL FILE FORMAT (Transact-SQL). Allowed values are **True** and **False** (default).	No

Direct copy by using PolyBase

Azure Synapse Analytics PolyBase directly supports Azure Blob, Azure Data Lake Storage Gen1 and Azure Data Lake Storage Gen2. If your source data meets the criteria described in this section, use PolyBase to copy directly from the source data store to Azure Synapse Analytics. Otherwise, use Staged copy by using PolyBase.

> **TIP**
> To copy data efficiently to Azure Synapse Analytics, learn more from Azure Data Factory makes it even easier and convenient to uncover insights from data when using Data Lake Store with Azure Synapse Analytics.

If the requirements aren't met, the service checks the settings and automatically falls back to the BULKINSERT mechanism for the data movement.

1. The **source linked service** is with the following types and authentication methods:

SUPPORTED SOURCE DATA STORE TYPE	SUPPORTED SOURCE AUTHENTICATION TYPE
Azure Blob	Account key authentication, managed identity authentication
Azure Data Lake Storage Gen1	Service principal authentication
Azure Data Lake Storage Gen2	Account key authentication, managed identity authentication

> **IMPORTANT**
> - When you use managed identity authentication for your storage linked service, learn the needed configurations for Azure Blob and Azure Data Lake Storage Gen2 respectively.
> - If your Azure Storage is configured with VNet service endpoint, you must use managed identity authentication with "allow trusted Microsoft service" enabled on storage account, refer to Impact of using VNet Service Endpoints with Azure storage.

2. The **source data format** is of **Parquet**, **ORC**, or **Delimited text**, with the following configurations:

   a. Folder path doesn't contain wildcard filter.

   b. File name is empty, or points to a single file. If you specify wildcard file name in copy activity, it can only be `*` or `*.*`.

   c. `rowDelimiter` is **default**, **\n**, **\r\n**, or **\r**.

d. `nullValue` is left as default or set to **empty string** (""), and `treatEmptyAsNull` is left as default or set to true.

e. `encodingName` is left as default or set to **utf-8**.

f. `quoteChar`, `escapeChar`, and `skipLineCount` aren't specified. PolyBase support skip header row, which can be configured as `firstRowAsHeader`.

g. `compression` can be **no compression**, `GZip`, or **Deflate**.

3. If your source is a folder, `recursive` in copy activity must be set to true.

4. `wildcardFolderPath`, `wildcardFilename`, `modifiedDateTimeStart`, `modifiedDateTimeEnd`, `prefix`, `enablePartitionDiscovery`, and `additionalColumns` are not specified.

> **NOTE**
> If your source is a folder, note PolyBase retrieves files from the folder and all of its subfolders, and it doesn't retrieve data from files for which the file name begins with an underline (_) or a period (.), as documented here - LOCATION argument.

```
"activities":[
    {
        "name": "CopyFromAzureBlobToSQLDataWarehouseViaPolyBase",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "ParquetDataset",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "AzureSQLDWDataset",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "ParquetSource",
                "storeSettings":{
                    "type": "AzureBlobStorageReadSettings",
                    "recursive": true
                }
            },
            "sink": {
                "type": "SqlDWSink",
                "allowPolyBase": true
            }
        }
    }
]
```

Staged copy by using PolyBase

When your source data is not natively compatible with PolyBase, enable data copying via an interim staging Azure Blob or Azure Data Lake Storage Gen2 (it can't be Azure Premium Storage). In this case, the service automatically converts the data to meet the data format requirements of PolyBase. Then it invokes PolyBase to load data into Azure Synapse Analytics. Finally, it cleans up your temporary data from the storage. See Staged copy for details about copying data via a staging.

To use this feature, create an Azure Blob Storage linked service or Azure Data Lake Storage Gen2 linked service with **account key or managed identity authentication** that refers to the Azure storage account as the interim storage.

IMPORTANT

- When you use managed identity authentication for your staging linked service, learn the needed configurations for Azure Blob and Azure Data Lake Storage Gen2 respectively.
- If your staging Azure Storage is configured with VNet service endpoint, you must use managed identity authentication with "allow trusted Microsoft service" enabled on storage account, refer to Impact of using VNet Service Endpoints with Azure storage.

IMPORTANT

If your staging Azure Storage is configured with Managed Private Endpoint and has the storage firewall enabled, you must use managed identity authentication and grant Storage Blob Data Reader permissions to the Synapse SQL Server to ensure it can access the staged files during the PolyBase load.

```
"activities":[
    {
        "name": "CopyFromSQLServerToSQLDataWarehouseViaPolyBase",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "SQLServerDataset",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "AzureSQLDWDataset",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SqlSource",
            },
            "sink": {
                "type": "SqlDWSink",
                "allowPolyBase": true
            },
            "enableStaging": true,
            "stagingSettings": {
                "linkedServiceName": {
                    "referenceName": "MyStagingStorage",
                    "type": "LinkedServiceReference"
                }
            }
        }
    }
]
```

Best practices for using PolyBase

The following sections provide best practices in addition to those practices mentioned in Best practices for Azure Synapse Analytics.

Required database permission

To use PolyBase, the user that loads data into Azure Synapse Analytics must have "CONTROL" permission on the target database. One way to achieve that is to add the user as a member of the **db_owner** role. Learn how to do that in the Azure Synapse Analytics overview.

Row size and data type limits

PolyBase loads are limited to rows smaller than 1 MB. It cannot be used to load to VARCHR(MAX), NVARCHAR(MAX), or VARBINARY(MAX). For more information, see Azure Synapse Analytics service capacity limits.

When your source data has rows greater than 1 MB, you might want to vertically split the source tables into several small ones. Make sure that the largest size of each row doesn't exceed the limit. The smaller tables can then be loaded by using PolyBase and merged together in Azure Synapse Analytics.

Alternatively, for data with such wide columns, you can use non-PolyBase to load the data by turning off "allow PolyBase" setting.

Azure Synapse Analytics resource class

To achieve the best possible throughput, assign a larger resource class to the user that loads data into Azure Synapse Analytics via PolyBase.

PolyBase troubleshooting

Loading to Decimal column

If your source data is in text format or other non-PolyBase compatible stores (using staged copy and PolyBase), and it contains empty value to be loaded into Azure Synapse Analytics Decimal column, you may get the following error:

```
ErrorCode=FailedDbOperation, ......HadoopSqlException: Error converting data type VARCHAR to
DECIMAL.....Detailed Message=Empty string can't be converted to DECIMAL.....
```

The solution is to unselect "**Use type default**" option (as false) in copy activity sink -> PolyBase settings. "USE_TYPE_DEFAULT" is a PolyBase native configuration, which specifies how to handle missing values in delimited text files when PolyBase retrieves data from the text file.

Check the tableName property in Azure Synapse Analytics

The following table gives examples of how to specify the **tableName** property in the JSON dataset. It shows several combinations of schema and table names.

DB SCHEMA	TABLE NAME	TABLENAME JSON PROPERTY
dbo	MyTable	MyTable or dbo.MyTable or [dbo].[MyTable]
dbo1	MyTable	dbo1.MyTable or [dbo1].[MyTable]
dbo	My.Table	[My.Table] or [dbo].[My.Table]
dbo1	My.Table	[dbo1].[My.Table]

If you see the following error, the problem might be the value you specified for the **tableName** property. See the preceding table for the correct way to specify values for the **tableName** JSON property.

```
Type=System.Data.SqlClient.SqlException,Message=Invalid object name 'stg.Account_test'.,Source=.Net
SqlClient Data Provider
```

Columns with default values

Currently, the PolyBase feature accepts only the same number of columns as in the target table. An example is a table with four columns where one of them is defined with a default value. The input data still needs to have four columns. A three-column input dataset yields an error similar to the following message:

```
All columns of the table must be specified in the INSERT BULK statement.
```

The NULL value is a special form of the default value. If the column is nullable, the input data in the blob for that column might be empty. But it can't be missing from the input dataset. PolyBase inserts NULL for missing values in Azure Synapse Analytics.

External file access failed

If you receive the following error, ensure that you are using managed identity authentication and have granted Storage Blob Data Reader permissions to the Azure Synapse workspace's managed identity.

```
Job failed due to reason: at Sink '[SinkName]':
shaded.msdataflow.com.microsoft.sqlserver.jdbc.SQLServerException: External file access failed due to
internal error: 'Error occurred while accessing HDFS: Java exception raised on call to
HdfsBridge_IsDirExist. Java exception message:\r\nHdfsBridge::isDirExist
```

For more information, see Grant permissions to managed identity after workspace creation.

Mapping data flow properties

When transforming data in mapping data flow, you can read and write to tables from Azure Synapse Analytics. For more information, see the source transformation and sink transformation in mapping data flows.

Source transformation

Settings specific to Azure Synapse Analytics are available in the **Source Options** tab of the source transformation.

Input Select whether you point your source at a table (equivalent of `Select * from <table-name>`) or enter a custom SQL query.

Enable Staging It is highly recommended that you use this option in production workloads with Azure Synapse Analytics sources. When you execute a data flow activity with Azure Synapse Analytics sources from a pipeline, you will be prompted for a staging location storage account and will use that for staged data loading. It is the fastest mechanism to load data from Azure Synapse Analytics.

- When you use managed identity authentication for your storage linked service, learn the needed configurations for Azure Blob and Azure Data Lake Storage Gen2 respectively.
- If your Azure Storage is configured with VNet service endpoint, you must use managed identity authentication with "allow trusted Microsoft service" enabled on storage account, refer to Impact of using VNet Service Endpoints with Azure storage.
- When you use Azure Synapse **serverless** SQL pool as source, enable staging is not supported.

Query: If you select Query in the input field, enter a SQL query for your source. This setting overrides any table that you've chosen in the dataset. **Order By** clauses aren't supported here, but you can set a full SELECT FROM statement. You can also use user-defined table functions. **select * from udfGetData()** is a UDF in SQL that returns a table. This query will produce a source table that you can use in your data flow. Using queries is also a great way to reduce rows for testing or for lookups.

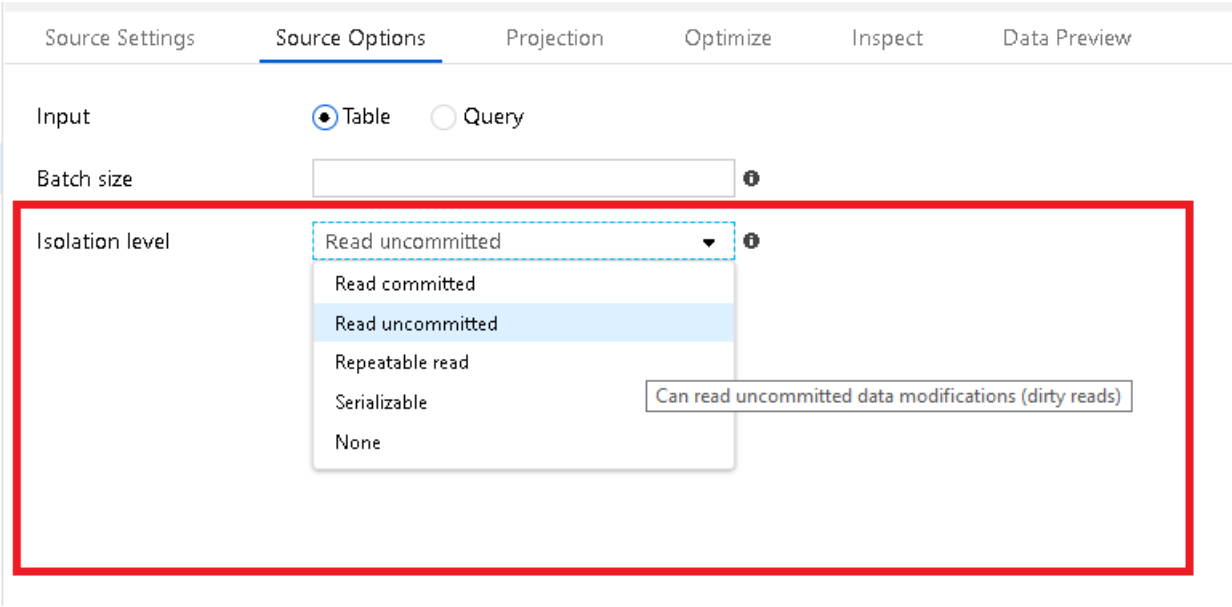
SQL Example: `Select * from MyTable where customerId > 1000 and customerId < 2000`

Batch size: Enter a batch size to chunk large data into reads. In data flows, this setting will be used to set Spark columnar caching. This is an option field, which will use Spark defaults if it is left blank.

Isolation Level: The default for SQL sources in mapping data flow is read uncommitted. You can change the isolation level here to one of these values:

- Read Committed

- Read Uncommitted
- Repeatable Read
- Serializable
- None (ignore isolation level)



Sink transformation

Settings specific to Azure Synapse Analytics are available in the **Settings** tab of the sink transformation.

Update method: Determines what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an alter-row transformation is required to tag rows for those actions. For updates, upserts and deletes, a key column or columns must be set to determine which row to alter.

Table action: Determines whether to recreate or remove all rows from the destination table prior to writing.

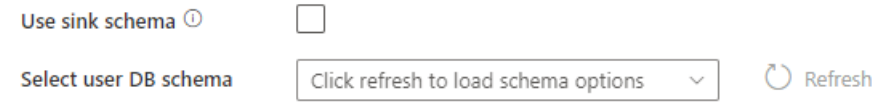
- None: No action will be done to the table.
- Recreate: The table will get dropped and recreated. Required if creating a new table dynamically.
- Truncate: All rows from the target table will get removed.

Enable staging: This enables loading into Azure Synapse Analytics SQL Pools using the copy command and is recommended for most Synapse sinks. The staging storage is configured in Execute Data Flow activity.

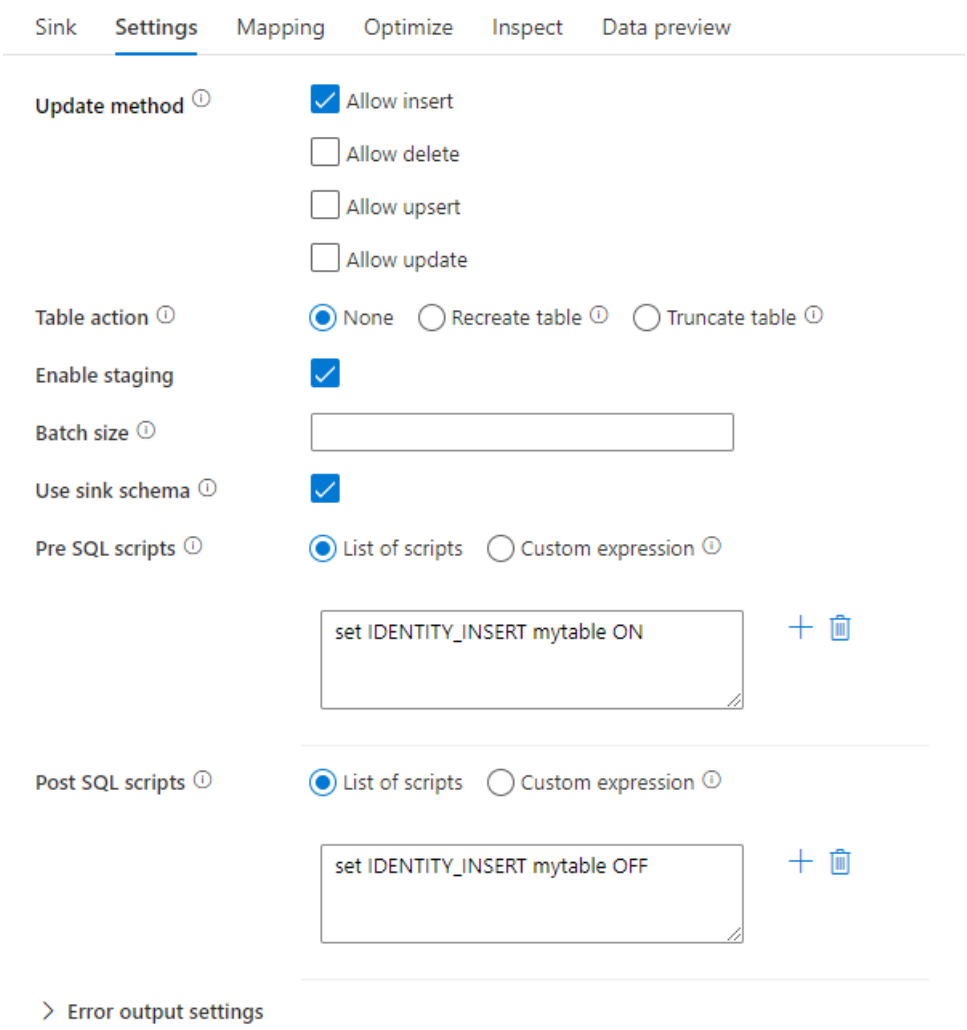
- When you use managed identity authentication for your storage linked service, learn the needed configurations for Azure Blob and Azure Data Lake Storage Gen2 respectively.
- If your Azure Storage is configured with VNet service endpoint, you must use managed identity authentication with "allow trusted Microsoft service" enabled on storage account, refer to Impact of using VNet Service Endpoints with Azure storage.

Batch size: Controls how many rows are being written in each bucket. Larger batch sizes improve compression and memory optimization, but risk out of memory exceptions when caching data.

Use sink schema: By default, a temporary table will be created under the sink schema as staging. You can alternatively uncheck the **Use sink schema** option and instead, in **Select user DB schema**, specify a schema name under which Data Factory will create a staging table to load upstream data and automatically clean them up upon completion. Make sure you have create table permission in the database and alter permission on the schema.



Pre and Post SQL scripts: Enter multi-line SQL scripts that will execute before (pre-processing) and after (post-processing) data is written to your Sink database



> **TIP**
>
> 1. It's recommended to break single batch scripts with multiple commands into multiple batches.
> 2. Only Data Definition Language (DDL) and Data Manipulation Language (DML) statements that return a simple update count can be run as part of a batch. Learn more from Performing batch operations

Error row handling

When writing to Azure Synapse Analytics, certain rows of data may fail due to constraints set by the destination. Some common errors include:

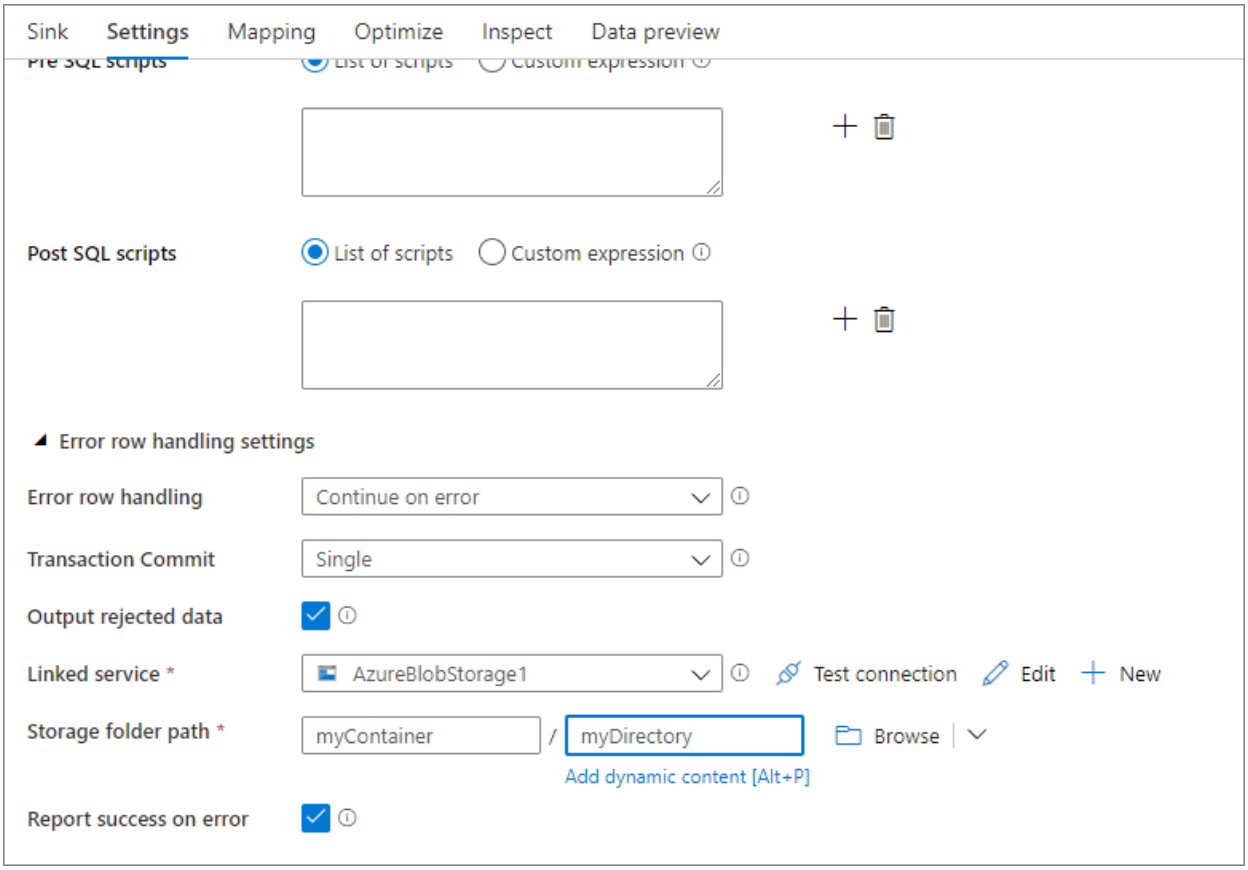
- String or binary data would be truncated in table
- Cannot insert the value NULL into column
- Conversion failed when converting the value to data type

By default, a data flow run will fail on the first error it gets. You can choose to **Continue on error** that allows your data flow to complete even if individual rows have errors. The service provides different options for you to handle these error rows.

Transaction Commit: Choose whether your data gets written in a single transaction or in batches. Single transaction will provide better performance and no data written will be visible to others until the transaction completes. Batch transactions have worse performance but can work for large datasets.

Output rejected data: If enabled, you can output the error rows into a csv file in Azure Blob Storage or an Azure Data Lake Storage Gen2 account of your choosing. This will write the error rows with three additional columns: the SQL operation like INSERT or UPDATE, the data flow error code, and the error message on the row.

Report success on error: If enabled, the data flow will be marked as a success even if error rows are found.



Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity

Data type mapping for Azure Synapse Analytics

When you copy data from or to Azure Synapse Analytics, the following mappings are used from Azure Synapse Analytics data types to Azure Data Factory interim data types. These mappings are also used when copying data from or to Azure Synapse Analytics using Synapse pipelines, since pipelines also implement Azure Data Factory within Azure Synapse. See schema and data type mappings to learn how Copy Activity maps the source schema and data type to the sink.

> **TIP**
> Refer to Table data types in Azure Synapse Analytics article on Azure Synapse Analytics supported data types and the workarounds for unsupported ones.

AZURE SYNAPSE ANALYTICS DATA TYPE	DATA FACTORY INTERIM DATA TYPE
bigint	Int64
binary	Byte[]
bit	Boolean
char	String, Char[]
date	DateTime
Datetime	DateTime
datetime2	DateTime
Datetimeoffset	DateTimeOffset
Decimal	Decimal
FILESTREAM attribute (varbinary(max))	Byte[]
Float	Double
image	Byte[]
int	Int32
money	Decimal
nchar	String, Char[]
numeric	Decimal
nvarchar	String, Char[]
real	Single
rowversion	Byte[]
smalldatetime	DateTime
smallint	Int16
smallmoney	Decimal
time	TimeSpan
tinyint	Byte
uniqueidentifier	Guid

AZURE SYNAPSE ANALYTICS DATA TYPE	DATA FACTORY INTERIM DATA TYPE
varbinary	Byte[]
varchar	String, Char[]

Next steps

For a list of data stores supported as sources and sinks by Copy Activity, see supported data stores and formats.

Copy data to and from Azure Table storage using Azure Data Factory or Synapse Analytics

8/19/2022 • 11 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data to and from Azure Table storage. It builds on the Copy Activity overview article that presents a general overview of Copy Activity.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Supported capabilities

This Azure Table storage connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
Copy activity (source/sink)	① ②	✓ Exclude storage account V1
Lookup activity	① ②	✓ Exclude storage account V1

① Azure integration runtime ② Self-hosted integration runtime

You can copy data from any supported source data store to Table storage. You also can copy data from Table storage to any supported sink data store. For a list of data stores that are supported as sources or sinks by the copy activity, see the Supported data stores table.

Specifically, this Azure Table connector supports copying data by using account key and service shared access signature authentications.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

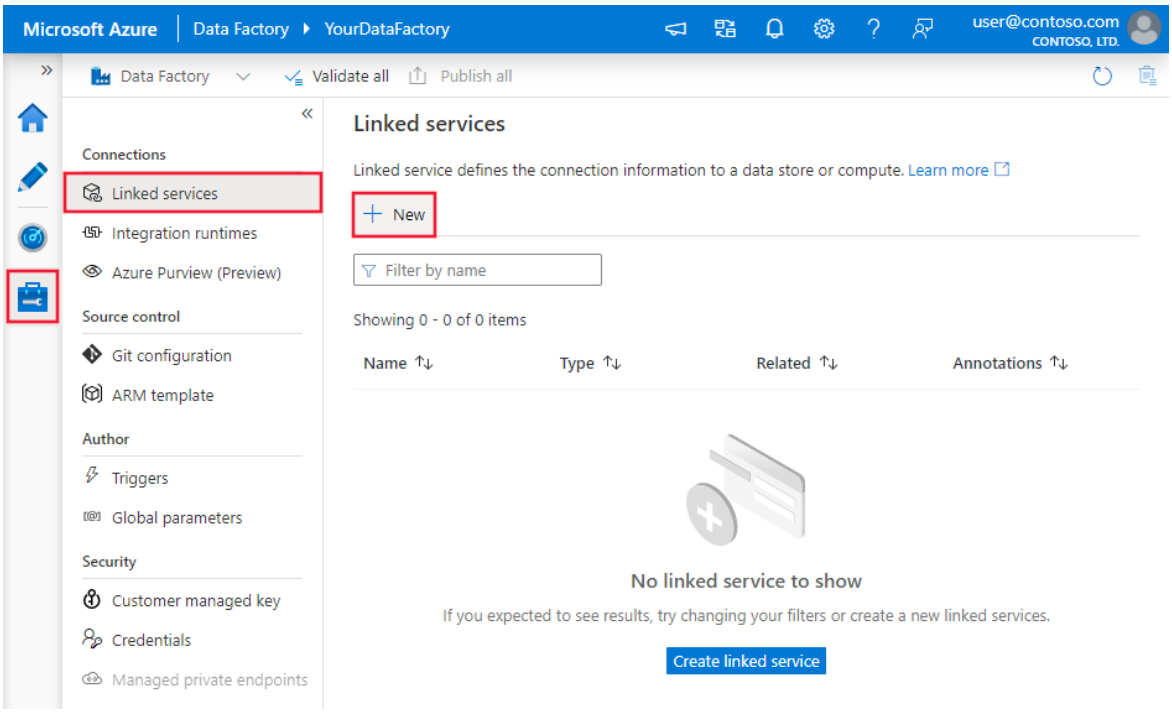
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create an Azure Table storage linked service using UI

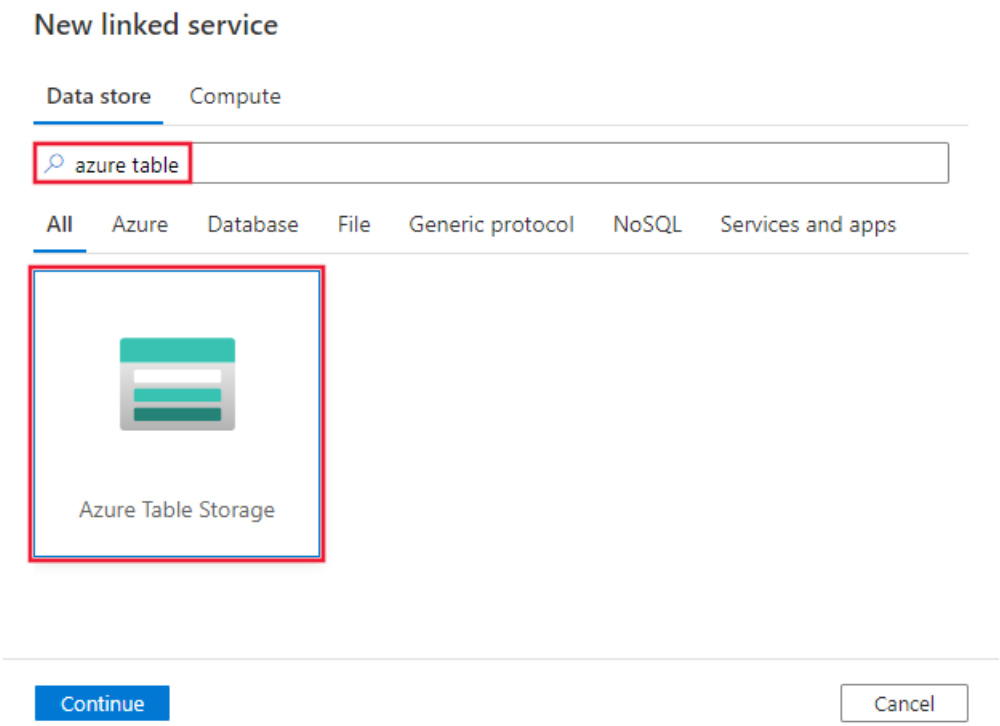
Use the following steps to create an Azure Table storage linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

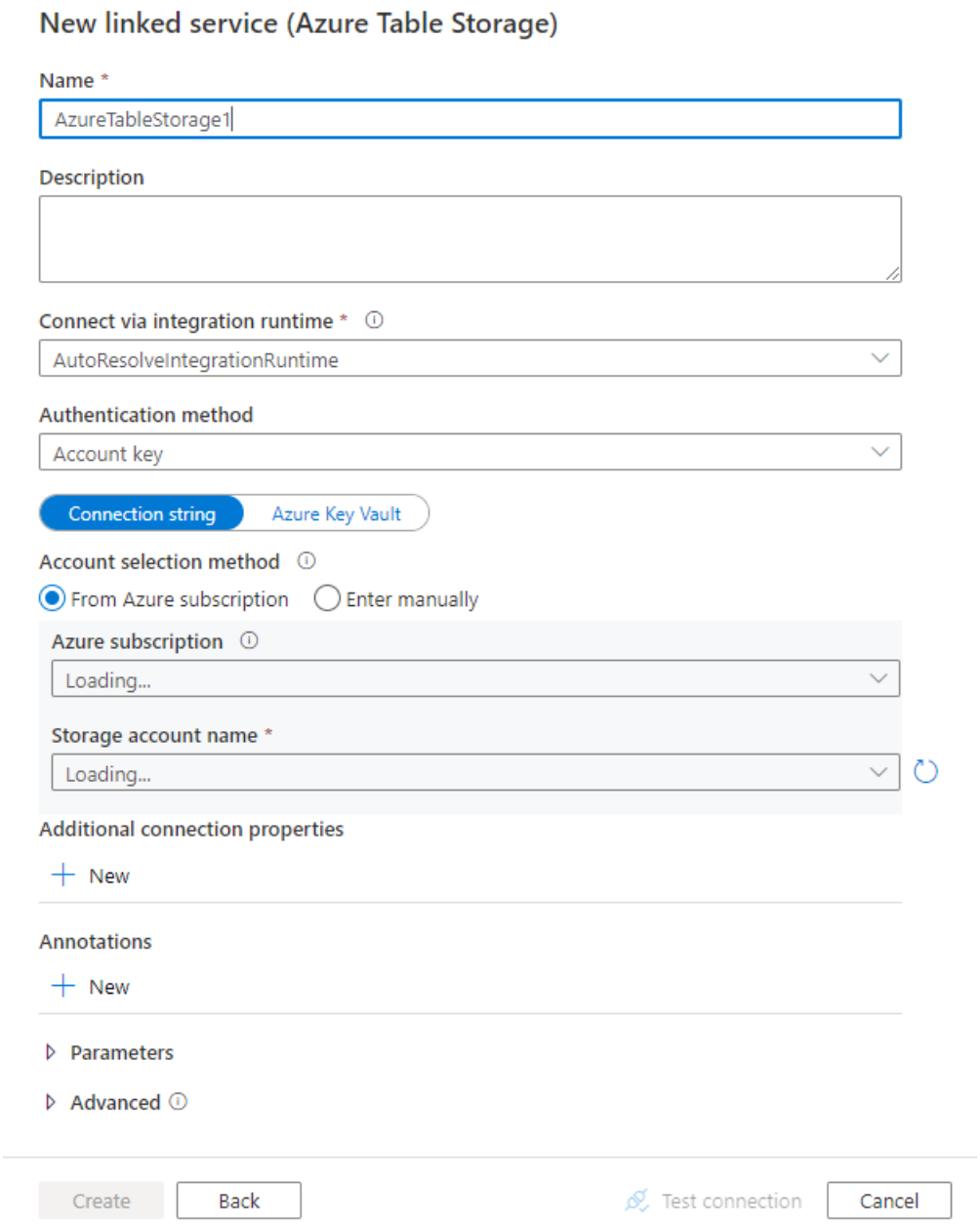
   - Azure Data Factory
   - Azure Synapse



2. Search for Azure Table and select the Azure Table storage connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to Azure Table storage.

Linked service properties

Use an account key

You can create an Azure Storage linked service by using the account key. It provides the the service with global access to Storage. The following properties are supported.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AzureTableStorage**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	Specify the information needed to connect to Storage for the connectionString property. You can also put account key in Azure Key Vault and pull the `accountKey` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is located in a private network). If not specified, it uses the default Azure Integration Runtime.	No

> **NOTE**
> If you were using "AzureStorage" type linked service, it is still supported as-is, while you are suggested to use this new "AzureTableStorage" linked service type going forward.

Example:

```
{
    "name": "AzureTableStorageLinkedService",
    "properties": {
        "type": "AzureTableStorage",
        "typeProperties": {
            "connectionString": "DefaultEndpointsProtocol=https;AccountName=<accountname>;AccountKey=
<accountkey>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store account key in Azure Key Vault

```
{
    "name": "AzureTableStorageLinkedService",
    "properties": {
        "type": "AzureTableStorage",
        "typeProperties": {
            "connectionString": "DefaultEndpointsProtocol=https;AccountName=<accountname>;",
            "accountKey": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Use shared access signature authentication

You also can create a Storage linked service by using a shared access signature. It provides the service with restricted/time-bound access to all/specific resources in the storage.

A shared access signature provides delegated access to resources in your storage account. You can use it to grant a client limited permissions to objects in your storage account for a specified time and with a specified set of permissions. You don't have to share your account access keys. The shared access signature is a URI that encompasses in its query parameters all the information necessary for authenticated access to a storage resource. To access storage resources with the shared access signature, the client only needs to pass in the shared access signature to the appropriate constructor or method. For more information about shared access signatures, see Shared access signatures: Understand the shared access signature model.

> **NOTE**
>
> Both **service shared access signatures** and **account shared access signatures** are now supported. For more information about shared access signatures, see Grant limited access to Azure Storage resources using shared access signatures (SAS).

> **TIP**
>
> To generate a service shared access signature for your storage account, you can execute the following PowerShell commands. Replace the placeholders and grant the needed permission.
>
> ```
> $context = New-AzStorageContext -StorageAccountName <accountName> -StorageAccountKey <accountKey>
> New-AzStorageContainerSASToken -Name <containerName> -Context $context -Permission rwdl -StartTime <startTime> -ExpiryTime <endTime> -FullUri
> ```

To use shared access signature authentication, the following properties are supported.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **AzureTableStorage**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
sasUri	Specify SAS URI of the shared access signature URI to the table. Mark this field as a SecureString to store it securely. You can also put SAS token in Azure Key Vault to leverage auto rotation and remove the token portion. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure Integration Runtime or the Self-hosted Integration Runtime (if your data store is located in a private network). If not specified, it uses the default Azure Integration Runtime.	No

NOTE

If you were using "AzureStorage" type linked service, it is still supported as-is, while you are suggested to use this new "AzureTableStorage" linked service type going forward.

Example:

```
{
    "name": "AzureTableStorageLinkedService",
    "properties": {
        "type": "AzureTableStorage",
        "typeProperties": {
            "sasUri": {
                "type": "SecureString",
                "value": "<SAS URI of the Azure Storage resource e.g.
https://<account>.table.core.windows.net/<table>?sv=<storage version>&amp;st=<start time>&amp;se=<expire
time>&amp;sr=<resource>&amp;sp=<permissions>&amp;sip=<ip range>&amp;spr=<protocol>&amp;sig=<signature>>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store account key in Azure Key Vault

```
{
    "name": "AzureTableStorageLinkedService",
    "properties": {
        "type": "AzureTableStorage",
        "typeProperties": {
            "sasUri": {
                "type": "SecureString",
                "value": "<SAS URI of the Azure Storage resource without token e.g.
https://<account>.table.core.windows.net/<table>>"
            },
            "sasToken": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

When you create a shared access signature URI, consider the following points:

- Set appropriate read/write permissions on objects based on how the linked service (read, write, read/write) is used.
- Set **Expiry time** appropriately. Make sure that the access to Storage objects doesn't expire within the active period of the pipeline.
- The URI should be created at the right table level based on the need.

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the Azure Table dataset.

To copy data to and from Azure Table, set the type property of the dataset to **AzureTable**. The following properties are supported.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **AzureTable**.	Yes
tableName	The name of the table in the Table storage database instance that the linked service refers to.	Yes

Example:

```
{
    "name": "AzureTableDataset",
    "properties":
    {
        "type": "AzureTable",
        "typeProperties": {
            "tableName": "MyTable"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Azure Table storage linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Schema inference by the service

For schema-free data stores such as Azure Table, the service infers the schema in one of the following ways:

- If you specify the column mapping in copy activity, the service uses the source side column list to retrieve data. In this case, if a row doesn't contain a value for a column, a null value is provided for it.
- If you don't specify the column mapping in copy activity, the service infers the schema by using the first row in the data. In this case, if the first row doesn't contain the full schema (e.g. some columns have null value), some columns are missed in the result of the copy operation.

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the Azure Table source and sink.

Azure Table as a source type

To copy data from Azure Table, set the source type in the copy activity to **AzureTableSource**. The following properties are supported in the copy activity **source** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **AzureTableSource**.	Yes
azureTableSourceQuery	Use the custom Table storage query to read data. The source query is a direct map from the `$filter` query option supported by Azure Table Storage, learn more about the syntax from this doc, and see the examples in the following azureTableSourceQuery examples section.	No
azureTableSourceIgnoreTableNotFound	Indicates whether to allow the exception of the table to not exist. Allowed values are **True** and **False** (default).	No

azureTableSourceQuery examples

> **NOTE**
>
> Azure Table query operation times out in 30 seconds as enforced by Azure Table service. Learn how to optimize the query from Design for querying article.

If you want to filter the data against a datetime type column, refer to this example:

```
"azureTableSourceQuery": "LastModifiedTime gt datetime'2017-10-01T00:00:00' and LastModifiedTime le
datetime'2017-10-02T00:00:00'"
```

If you want to filter the data against a string type column, refer to this example:

```
"azureTableSourceQuery": "LastModifiedTime ge '201710010000_0000' and LastModifiedTime le
'201710010000_9999'"
```

If you use the pipeline parameter, cast the datetime value to proper format according to the previous samples.

Azure Table as a sink type

To copy data to Azure Table, set the sink type in the copy activity to **AzureTableSink**. The following properties are supported in the copy activity **sink** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to **AzureTableSink**.	Yes
azureTableDefaultPartitionKeyValue	The default partition key value that can be used by the sink.	No
azureTablePartitionKeyName	Specify the name of the column whose values are used as partition keys. If not specified, "AzureTableDefaultPartitionKeyValue" is used as the partition key.	No
azureTableRowKeyName	Specify the name of the column whose column values are used as the row key. If not specified, use a GUID for each row.	No
azureTableInsertType	The mode to insert data into Azure Table. This property controls whether existing rows in the output table with matching partition and row keys have their values replaced or merged. Allowed values are **merge** (default) and **replace**. This setting applies at the row level not the table level. Neither option deletes rows in the output table that do not exist in the input. To learn about how the merge and replace settings work, see Insert or merge entity and Insert or replace entity.	No

PROPERTY	DESCRIPTION	REQUIRED
writeBatchSize	Inserts data into Azure Table when writeBatchSize or writeBatchTimeout is hit. Allowed values are integer (number of rows).	No (default is 10,000)
writeBatchTimeout	Inserts data into Azure Table when writeBatchSize or writeBatchTimeout is hit. Allowed values are timespan. An example is "00:20:00" (20 minutes).	No (default is 90 seconds, storage client's default timeout)
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToAzureTable",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Azure Table output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureTableSink",
                "azureTablePartitionKeyName": "<column name>",
                "azureTableRowKeyName": "<column name>"
            }
        }
    }
]
```

azureTablePartitionKeyName

Map a source column to a destination column by using the **"translator"** property before you can use the destination column as azureTablePartitionKeyName.

In the following example, source column DivisionID is mapped to the destination column DivisionID:

```
"translator": {
    "type": "TabularTranslator",
    "columnMappings": "DivisionID: DivisionID, FirstName: FirstName, LastName: LastName"
}
```

"DivisionID" is specified as the partition key.

```
"sink": {
    "type": "AzureTableSink",
    "azureTablePartitionKeyName": "DivisionID"
}
```

Data type mapping for Azure Table

When you copy data from and to Azure Table, the following mappings are used from Azure Table data types to interim data types used internally within the service. To learn about how the copy activity maps the source schema and data type to the sink, see Schema and data type mappings.

When you move data to and from Azure Table, the following mappings defined by Azure Table are used from Azure Table OData types to .NET type and vice versa.

AZURE TABLE DATA TYPE	INTERIM SERVICE DATA TYPE	DETAILS
Edm.Binary	byte[]	An array of bytes up to 64 KB.
Edm.Boolean	bool	A Boolean value.
Edm.DateTime	DateTime	A 64-bit value expressed as Coordinated Universal Time (UTC). The supported DateTime range begins midnight, January 1, 1601 A.D. (C.E.), UTC. The range ends December 31, 9999.
Edm.Double	double	A 64-bit floating point value.
Edm.Guid	Guid	A 128-bit globally unique identifier.
Edm.Int32	Int32	A 32-bit integer.
Edm.Int64	Int64	A 64-bit integer.
Edm.String	String	A UTF-16-encoded value. String values can be up to 64 KB.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Binary format in Azure Data Factory and Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Binary format is supported for the following connectors: Amazon S3, Amazon S3 Compatible Storage, Azure Blob, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure Files, File System, FTP, Google Cloud Storage, HDFS, HTTP, Oracle Cloud Storage and SFTP.

You can use Binary dataset in Copy activity, GetMetadata activity, or Delete activity. When using Binary dataset, the service does not parse file content but treat it as-is.

> **NOTE**
> When using Binary dataset in copy activity, you can only copy from Binary dataset to Binary dataset.

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the Binary dataset.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **Binary**.	Yes
location	Location settings of the file(s). Each file-based connector has its own location type and supported properties under `location`. **See details in connector article -> Dataset properties section**.	Yes
compression	Group of properties to configure file compression. Configure this section when you want to do compression/decompression during activity execution.	No

PROPERTY	DESCRIPTION	REQUIRED
type	The compression codec used to read/write binary files. Allowed values are **bzip2**, **gzip**, **deflate**, **ZipDeflate**, **Tar**, or **TarGzip**. **Note** when using copy activity to decompress **ZipDeflate/TarGzip/Tar** file(s) and write to file-based sink data store, by default files are extracted to the folder: `<path specified in dataset>/<folder named as source compressed file>/`, use `preserveZipFileNameAsFolder` / `preserveCompressionFileNameAsFolder` on copy activity source to control whether to preserve the name of the compressed file(s) as folder structure.	No
level	The compression ratio. Apply when dataset is used in Copy activity sink. Allowed values are **Optimal** or **Fastest**. - **Fastest:** The compression operation should complete as quickly as possible, even if the resulting file is not optimally compressed. - **Optimal**: The compression operation should be optimally compressed, even if the operation takes a longer time to complete. For more information, see Compression Level topic.	No

Below is an example of Binary dataset on Azure Blob Storage:

```
{
    "name": "BinaryDataset",
    "properties": {
        "type": "Binary",
        "linkedServiceName": {
            "referenceName": "<Azure Blob Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "container": "containername",
                "folderPath": "folder/subfolder",
            },
            "compression": {
                "type": "ZipDeflate"
            }
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the Binary source and sink.

> **NOTE**
>
> When using Binary dataset in copy activity, you can only copy from Binary dataset to Binary dataset.

Binary as source

The following properties are supported in the copy activity **source** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **BinarySource**.	Yes
formatSettings	A group of properties. Refer to **Binary read settings** table below.	No
storeSettings	A group of properties on how to read data from a data store. Each file-based connector has its own supported read settings under `storeSettings`. **See details in connector article -> Copy activity properties section**.	No

Supported **binary read settings** under `formatSettings`:

PROPERTY	DESCRIPTION	REQUIRED
type	The type of formatSettings must be set to **BinaryReadSettings**.	Yes
compressionProperties	A group of properties on how to decompress data for a given compression codec.	No
preserveZipFileNameAsFolder (*under* `compressionProperties` -> `type` *as* `ZipDeflateReadSettings`)	Applies when input dataset is configured with **ZipDeflate** compression. Indicates whether to preserve the source zip file name as folder structure during copy. - When set to **true (default)**, The service writes unzipped files to `<path specified in dataset>/<folder named as source zip file>/`. - When set to **false**, the service writes unzipped files directly to `<path specified in dataset>`. Make sure you don't have duplicated file names in different source zip files to avoid racing or unexpected behavior.	No

PROPERTY	DESCRIPTION	REQUIRED
preserveCompressionFileNameAsFolder (*under* `compressionProperties` -> `type` *as* `TarGZipReadSettings` *or* `TarReadSettings`)	Applies when input dataset is configured with **TarGzip/Tar** compression. Indicates whether to preserve the source compressed file name as folder structure during copy. - When set to **true (default)**, the service writes decompressed files to `<path specified in dataset>/<folder named as source compressed file>/`. - When set to **false**, the service writes decompressed files directly to `<path specified in dataset>`. Make sure you don't have duplicated file names in different source files to avoid racing or unexpected behavior.	No

```
"activities": [
    {
        "name": "CopyFromBinary",
        "type": "Copy",
        "typeProperties": {
            "source": {
                "type": "BinarySource",
                "storeSettings": {
                    "type": "AzureBlobStorageReadSettings",
                    "recursive": true,
                    "deleteFilesAfterCompletion": true
                },
                "formatSettings": {
                    "type": "BinaryReadSettings",
                    "compressionProperties": {
                        "type": "ZipDeflateReadSettings",
                        "preserveZipFileNameAsFolder": false
                    }
                }
            },
            ...
        }
        ...
    }
]
```

Binary as sink

The following properties are supported in the copy activity ***sink*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **BinarySink**.	Yes
storeSettings	A group of properties on how to write data to a data store. Each file-based connector has its own supported write settings under `storeSettings`. **See details in connector article -> Copy activity properties section**.	No

Next steps

- Copy activity overview
- GetMetadata activity
- Delete activity

Copy data from Cassandra using Azure Data Factory or Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from a Cassandra database. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Cassandra connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Specifically, this Cassandra connector supports:

- Cassandra **versions 2.x and 3.x**.
- Copying data using **Basic** or **Anonymous** authentication.

> **NOTE**
> For activity running on Self-hosted Integration Runtime, Cassandra 3.x is supported since IR version 3.7 and above.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

The Integration Runtime provides a built-in Cassandra driver, therefore you don't need to manually install any driver when copying data from/to Cassandra.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

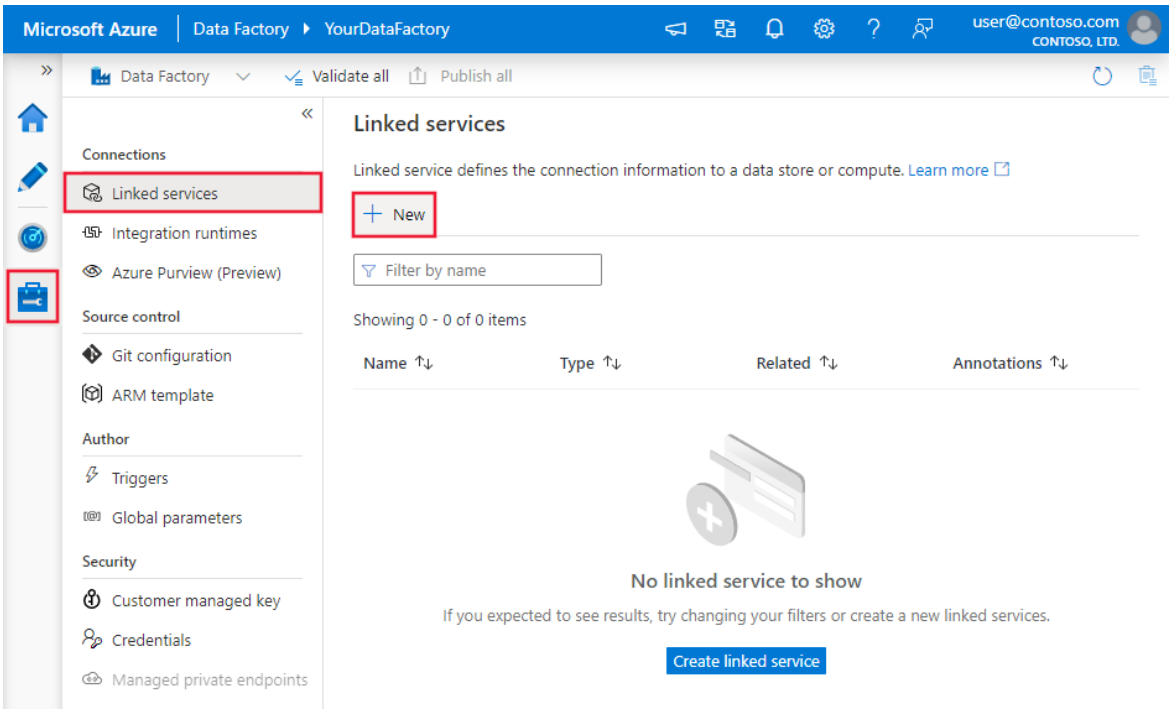
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Cassandra using UI

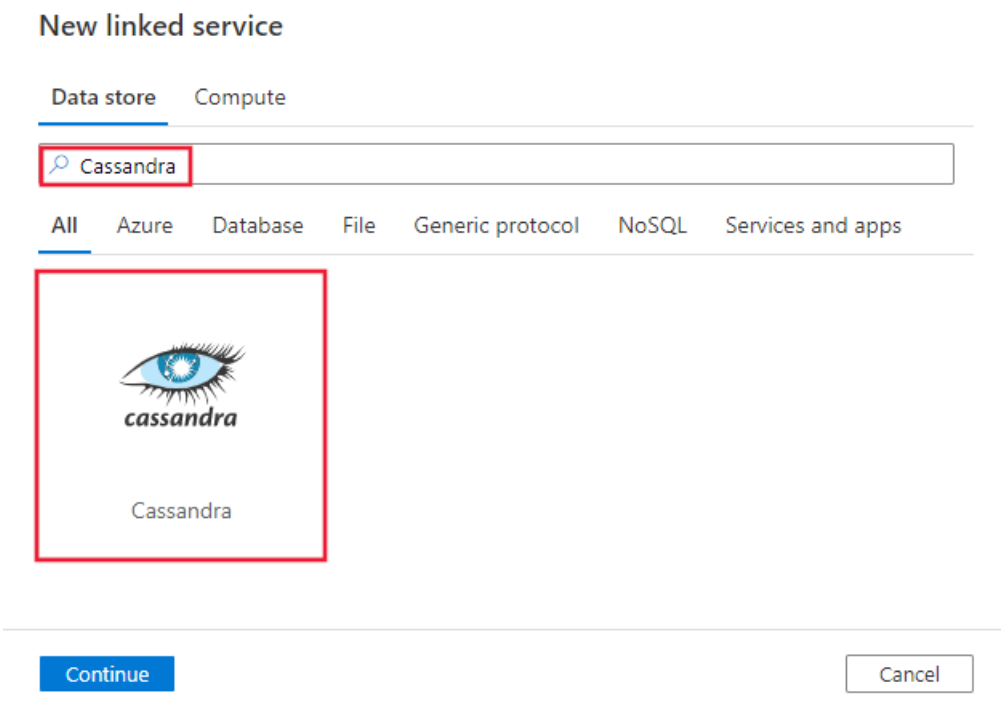
Use the following steps to create a linked service to Cassandra in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

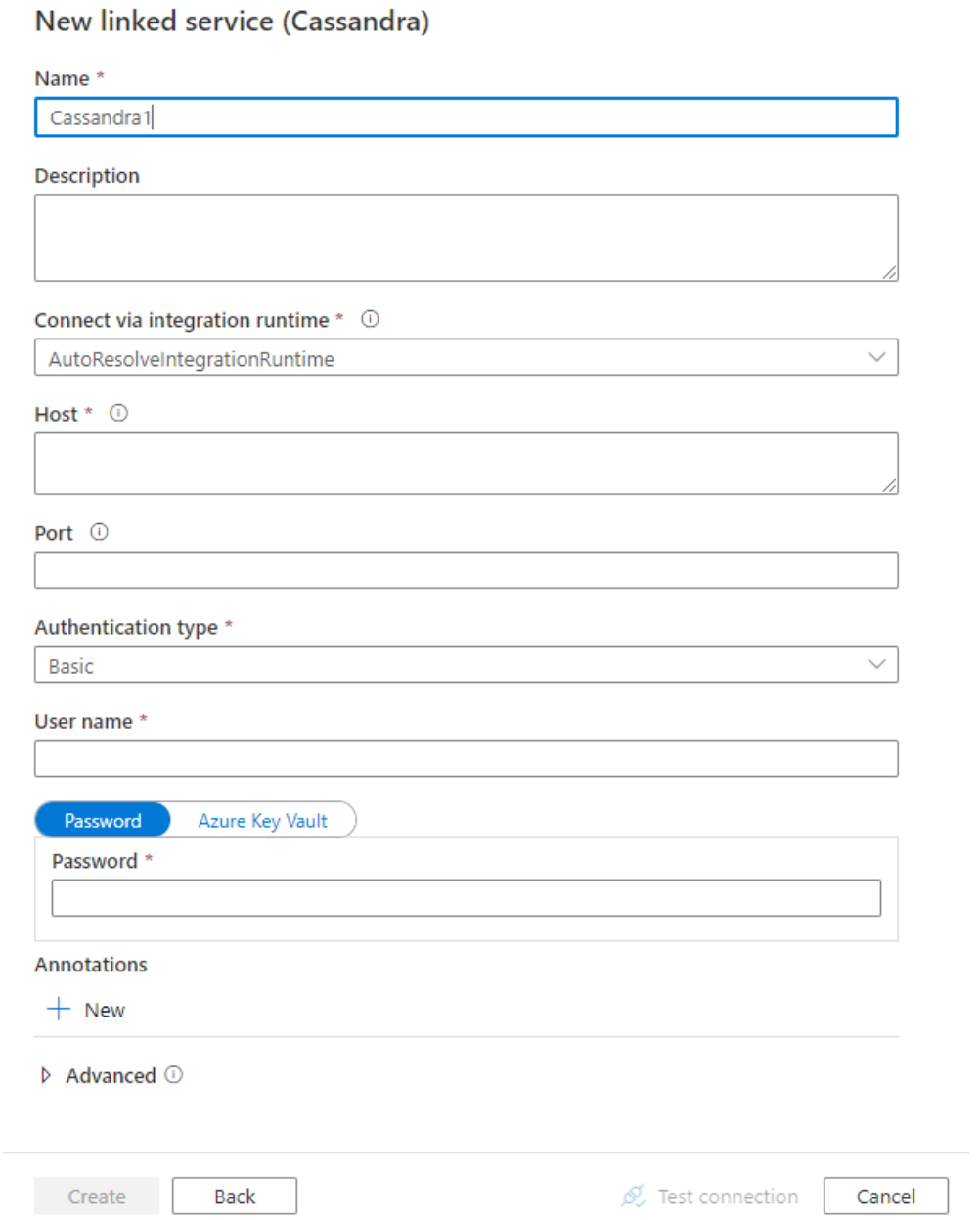
   - Azure Data Factory
   - Azure Synapse



2. Search for Cassandra and select the Cassandra connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Cassandra connector.

Linked service properties

The following properties are supported for Cassandra linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Cassandra**	Yes
host	One or more IP addresses or host names of Cassandra servers. Specify a comma-separated list of IP addresses or host names to connect to all servers concurrently.	Yes
port	The TCP port that the Cassandra server uses to listen for client connections.	No (default is 9042)
authenticationType	Type of authentication used to connect to the Cassandra database. Allowed values are: **Basic**, and **Anonymous**.	Yes
username	Specify user name for the user account.	Yes, if authenticationType is set to Basic.
password	Specify password for the user account. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes, if authenticationType is set to Basic.
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

> **NOTE**
> Currently connection to Cassandra using TLS is not supported.

Example:

```
{
    "name": "CassandraLinkedService",
    "properties": {
        "type": "Cassandra",
        "typeProperties": {
            "host": "<host>",
            "authenticationType": "Basic",
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Cassandra dataset.

To copy data from Cassandra, set the type property of the dataset to **CassandraTable**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **CassandraTable**	Yes
keyspace	Name of the keyspace or schema in Cassandra database.	No (if "query" for "CassandraSource" is specified)
tableName	Name of the table in Cassandra database.	No (if "query" for "CassandraSource" is specified)

Example:

```
{
    "name": "CassandraDataset",
    "properties": {
        "type": "CassandraTable",
        "typeProperties": {
            "keySpace": "<keyspace name>",
            "tableName": "<table name>"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Cassandra linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Cassandra source.

Cassandra as source

To copy data from Cassandra, set the source type in the copy activity to **CassandraSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **CassandraSource**	Yes
query	Use the custom query to read data. SQL-92 query or CQL query. See CQL reference. When using SQL query, specify **keyspace name.table name** to represent the table you want to query.	No (if "tableName" and "keyspace" in dataset are specified).
consistencyLevel	The consistency level specifies how many replicas must respond to a read request before returning data to the client application. Cassandra checks the specified number of replicas for data to satisfy the read request. See Configuring data consistency for details. Allowed values are: **ONE**, **TWO**, **THREE**, **QUORUM**, **ALL**, **LOCAL_QUORUM**, **EACH_QUORUM**, and **LOCAL_ONE**.	No (default is `ONE`)

Example:

```
"activities":[
    {
        "name": "CopyFromCassandra",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Cassandra input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "CassandraSource",
                "query": "select id, firstname, lastname from mykeyspace.mytable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Data type mapping for Cassandra

When copying data from Cassandra, the following mappings are used from Cassandra data types to interim data types used internally within the service. See Schema and data type mappings to learn about how copy activity maps the source schema and data type to the sink.

CASSANDRA DATA TYPE	INTERIM SERVICE DATA TYPE
ASCII	String
BIGINT	Int64
BLOB	Byte[]
BOOLEAN	Boolean
DECIMAL	Decimal
DOUBLE	Double
FLOAT	Single
INET	String
INT	Int32
TEXT	String
TIMESTAMP	DateTime

CASSANDRA DATA TYPE	INTERIM SERVICE DATA TYPE
TIMEUUID	Guid
UUID	Guid
VARCHAR	String
VARINT	Decimal

> **NOTE**
>
> For collection types (map, set, list, etc.), refer to Work with Cassandra collection types using virtual table section.
>
> User-defined types are not supported.
>
> The length of Binary Column and String Column lengths cannot be greater than 4000.

Work with collections using virtual table

The service uses a built-in ODBC driver to connect to and copy data from your Cassandra database. For collection types including map, set and list, the driver renormalizes the data into corresponding virtual tables. Specifically, if a table contains any collection columns, the driver generates the following virtual tables:

- A **base table**, which contains the same data as the real table except for the collection columns. The base table uses the same name as the real table that it represents.
- A **virtual table** for each collection column, which expands the nested data. The virtual tables that represent collections are named using the name of the real table, a separator "*vt*" and the name of the column.

Virtual tables refer to the data in the real table, enabling the driver to access the de-normalized data. See Example section for details. You can access the content of Cassandra collections by querying and joining the virtual tables.

Example

For example, the following "ExampleTable" is a Cassandra database table that contains an integer primary key column named "pk_int", a text column named value, a list column, a map column, and a set column (named "StringSet").

PK_INT	VALUE	LIST	MAP	STRINGSET
1	"sample value 1"	["1", "2", "3"]	{"S1": "a", "S2": "b"}	{"A", "B", "C"}
3	"sample value 3"	["100", "101", "102", "105"]	{"S1": "t"}	{"A", "E"}

The driver would generate multiple virtual tables to represent this single table. The foreign key columns in the virtual tables reference the primary key columns in the real table, and indicate which real table row the virtual table row corresponds to.

The first virtual table is the base table named "ExampleTable" is shown in the following table:

PK_INT	VALUE
1	"sample value 1"

PK_INT	VALUE
3	"sample value 3"

The base table contains the same data as the original database table except for the collections, which are omitted from this table and expanded in other virtual tables.

The following tables show the virtual tables that renormalize the data from the List, Map, and StringSet columns. The columns with names that end with "_index" or "_key" indicate the position of the data within the original list or map. The columns with names that end with "_value" contain the expanded data from the collection.

Table "ExampleTable_vt_List":

PK_INT	LIST_INDEX	LIST_VALUE
1	0	1
1	1	2
1	2	3
3	0	100
3	1	101
3	2	102
3	3	103

Table "ExampleTable_vt_Map":

PK_INT	MAP_KEY	MAP_VALUE
1	S1	A
1	S2	b
3	S1	t

Table "ExampleTable_vt_StringSet":

PK_INT	STRINGSET_VALUE
1	A
1	B
1	C
3	A
3	E

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Common Data Model format in Azure Data Factory and Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

The Common Data Model (CDM) metadata system makes it possible for data and its meaning to be easily shared across applications and business processes. To learn more, see the Common Data Model overview.

In Azure Data Factory and Synapse pipelines, users can transform data from CDM entities in both model.json and manifest form stored in Azure Data Lake Store Gen2 (ADLS Gen2) using mapping data flows. You can also sink data in CDM format using CDM entity references that will land your data in CSV or Parquet format in partitioned folders.

Mapping data flow properties

The Common Data Model is available as an inline dataset in mapping data flows as both a source and a sink.

> **NOTE**
> When writing CDM entities, you must have an existing CDM entity definition (metadata schema) already defined to use as a reference. The data flow sink will read that CDM entity file and import the schema into your sink for field mapping.

Source properties

The below table lists the properties supported by a CDM source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Format	Format must be `cdm`	yes	`cdm`	format
Metadata format	Where the entity reference to the data is located. If using CDM version 1.0, choose manifest. If using a CDM version before 1.0, choose model.json.	Yes	`'manifest'` or `'model'`	manifestType
Root location: container	Container name of the CDM folder	yes	String	fileSystem
Root location: folder path	Root folder location of CDM folder	yes	String	folderPath
Manifest file: Entity path	Folder path of the entity within the root folder	no	String	entityPath

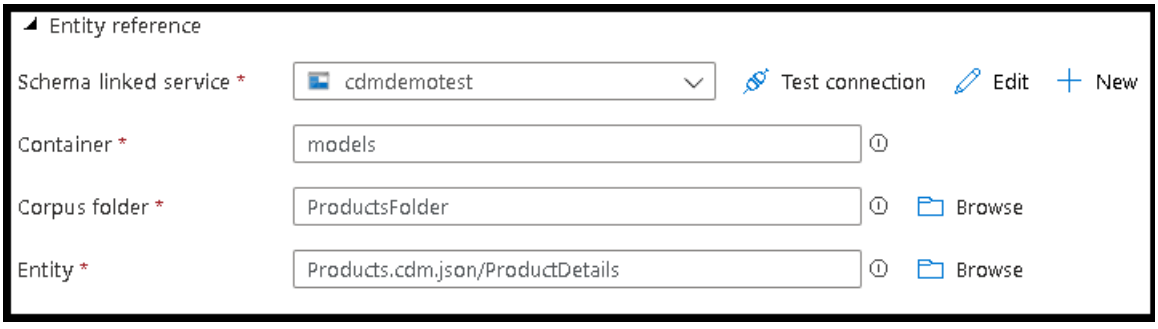
NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Manifest file: Manifest name	Name of the manifest file. Default value is 'default'	No	String	manifestName
Filter by last modified	Choose to filter files based upon when they were last altered	no	Timestamp	modifiedAfter modifiedBefore
Schema linked service	The linked service where the corpus is located	yes, if using manifest	`'adlsgen2'` or `'github'`	corpusStore
Entity reference container	Container corpus is in	yes, if using manifest and corpus in ADLS Gen2	String	adlsgen2_fileSystem
Entity reference Repository	GitHub repository name	yes, if using manifest and corpus in GitHub	String	github_repository
Entity reference Branch	GitHub repository branch	yes, if using manifest and corpus in GitHub	String	github_branch
Corpus folder	the root location of the corpus	yes, if using manifest	String	corpusPath
Corpus entity	Path to entity reference	yes	String	entity
Allow no files found	If true, an error is not thrown if no files are found	no	`true` or `false`	ignoreNoFilesFound

When selecting "Entity Reference" both in the Source and Sink transformations, you can select from these three options for the location of your entity reference:

- Local uses the entity defined in the manifest file already being used by the service
- Custom will ask you to point to an entity manifest file that is different from the manifest file the service is using
- Standard will use an entity reference from the standard library of CDM entities maintained in `GitHub`.

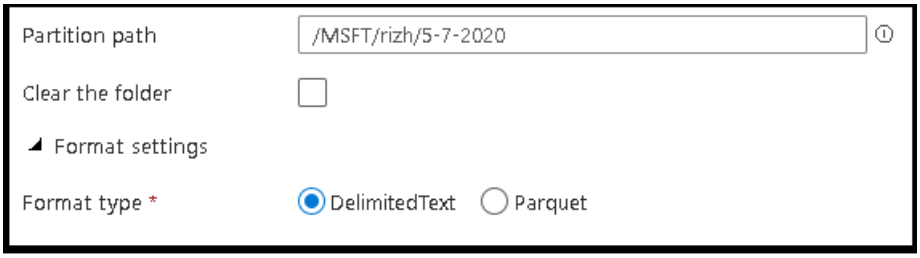
Sink settings

- Point to the CDM entity reference file that contains the definition of the entity you would like to write.



- Define the partition path and format of the output files that you want the service to use for writing your

entities.

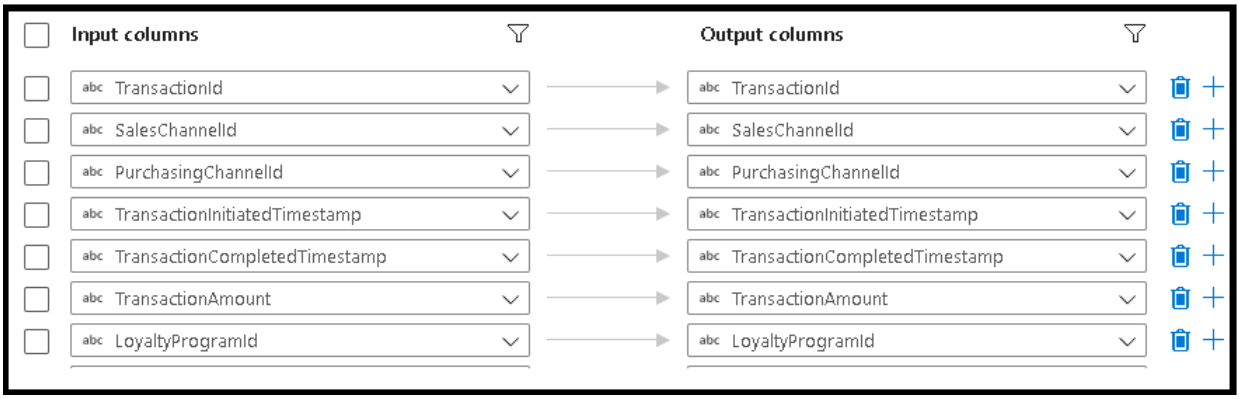


- Set the output file location and the location and name for the manifest file.

Import schema

CDM is only available as an inline dataset and, by default, doesn't have an associated schema. To get column metadata, click the **Import schema** button in the **Projection** tab. This will allow you to reference the column names and data types specified by the corpus. To import the schema, a data flow debug session must be active and you must have an existing CDM entity definition file to point to.

When mapping data flow columns to entity properties in the Sink transformation, click on the "Mapping" tab and select "Import Schema". The service will read the entity reference that you pointed to in your Sink options, allowing you to map to the target CDM schema.



> **NOTE**
> When using model.json source type that originates from Power BI or Power Platform dataflows, you may encounter "corpus path is null or empty" errors from the source transformation. This is likely due to formatting issues of the partition location path in the model.json file. To fix this, follow these steps:

1. Open the model.json file in a text editor
2. Find the partitions.Location property
3. Change "blob.core.windows.net" to "dfs.core.windows.net"
4. Fix any "%2F" encoding in the URL to "/"
5. If using ADF Data Flows, Special characters in the partition file path must be replaced with alpha-numeric values, or switch to Azure Synapse Data Flows

CDM source data flow script example

```
source(output(
        ProductSizeId as integer,
        ProductColor as integer,
        CustomerId as string,
        Note as string,
        LastModifiedDate as timestamp
    ),
    allowSchemaDrift: true,
    validateSchema: false,
    entity: 'Product.cdm.json/Product',
    format: 'cdm',
    manifestType: 'manifest',
    manifestName: 'ProductManifest',
    entityPath: 'Product',
    corpusPath: 'Products',
    corpusStore: 'adlsgen2',
    adlsgen2_fileSystem: 'models',
    folderPath: 'ProductData',
    fileSystem: 'data') ~> CDMSource
```

Sink properties

The below table lists the properties supported by a CDM sink. You can edit these properties in the **Settings** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Format	Format must be `cdm`	yes	`cdm`	format
Root location: container	Container name of the CDM folder	yes	String	fileSystem
Root location: folder path	Root folder location of CDM folder	yes	String	folderPath
Manifest file: Entity path	Folder path of the entity within the root folder	no	String	entityPath
Manifest file: Manifest name	Name of the manifest file. Default value is 'default'	No	String	manifestName
Schema linked service	The linked service where the corpus is located	yes	`'adlsgen2'` or `'github'`	corpusStore
Entity reference container	Container corpus is in	yes, if corpus in ADLS Gen2	String	adlsgen2_fileSystem
Entity reference Repository	GitHub repository name	yes, if corpus in GitHub	String	github_repository
Entity reference Branch	GitHub repository branch	yes, if corpus in GitHub	String	github_branch
Corpus folder	the root location of the corpus	yes	String	corpusPath

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Corpus entity	Path to entity reference	yes	String	entity
Partition path	Location where the partition will be written	no	String	partitionPath
Clear the folder	If the destination folder is cleared prior to write	no	`true` or `false`	truncate
Format type	Choose to specify parquet format	no	`parquet` if specified	subformat
Column delimiter	If writing to DelimitedText, how to delimit columns	yes, if writing to DelimitedText	String	columnDelimiter
First row as header	If using DelimitedText, whether the column names are added as a header	no	`true` or `false`	columnNamesAsHeader

CDM sink data flow script example

The associated data flow script is:

```
CDMSource sink(allowSchemaDrift: true,
    validateSchema: false,
    entity: 'Product.cdm.json/Product',
    format: 'cdm',
    entityPath: 'ProductSize',
    manifestName: 'ProductSizeManifest',
    corpusPath: 'Products',
    partitionPath: 'adf',
    folderPath: 'ProductSizeData',
    fileSystem: 'cdm',
    subformat: 'parquet',
    corpusStore: 'adlsgen2',
    adlsgen2_fileSystem: 'models',
    truncate: true,
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> CDMSink
```

Next steps

Create a source transformation in mapping data flow.

Copy data from Concur using Azure Data Factory or Synapse Analytics(Preview)

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Concur. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Concur connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

> **NOTE**
> Partner account is currently not supported.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

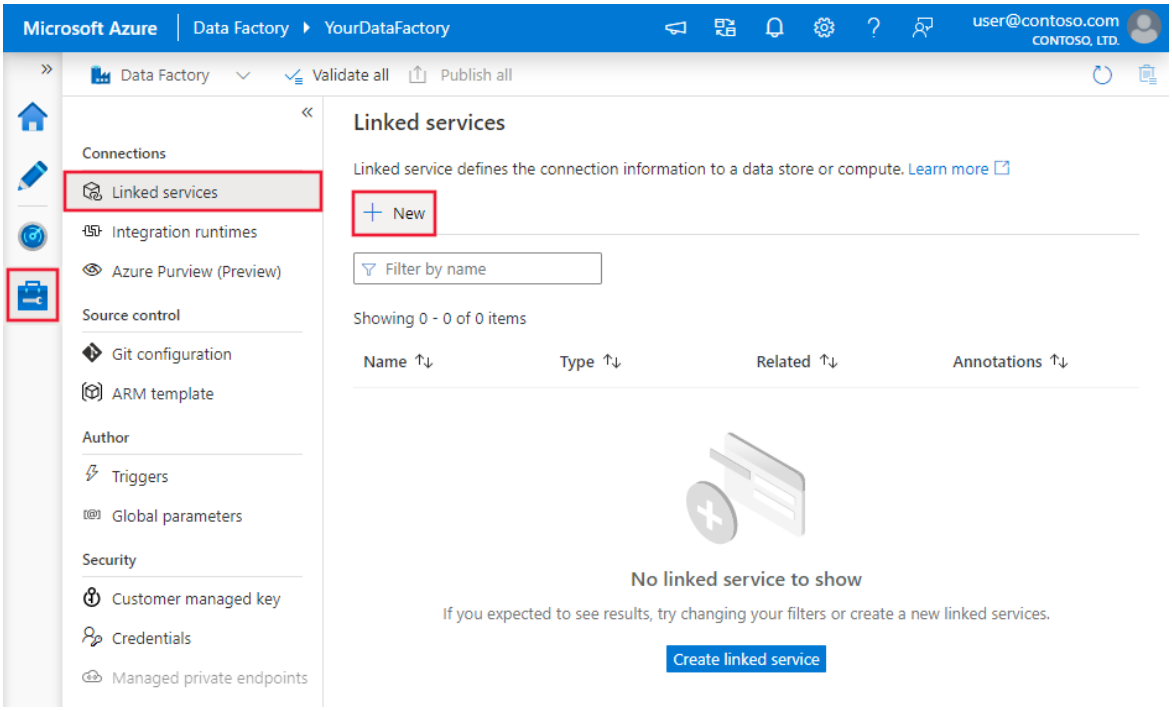
Create a linked service to Concur using UI

Use the following steps to create a linked service to Concur in the Azure portal UI.

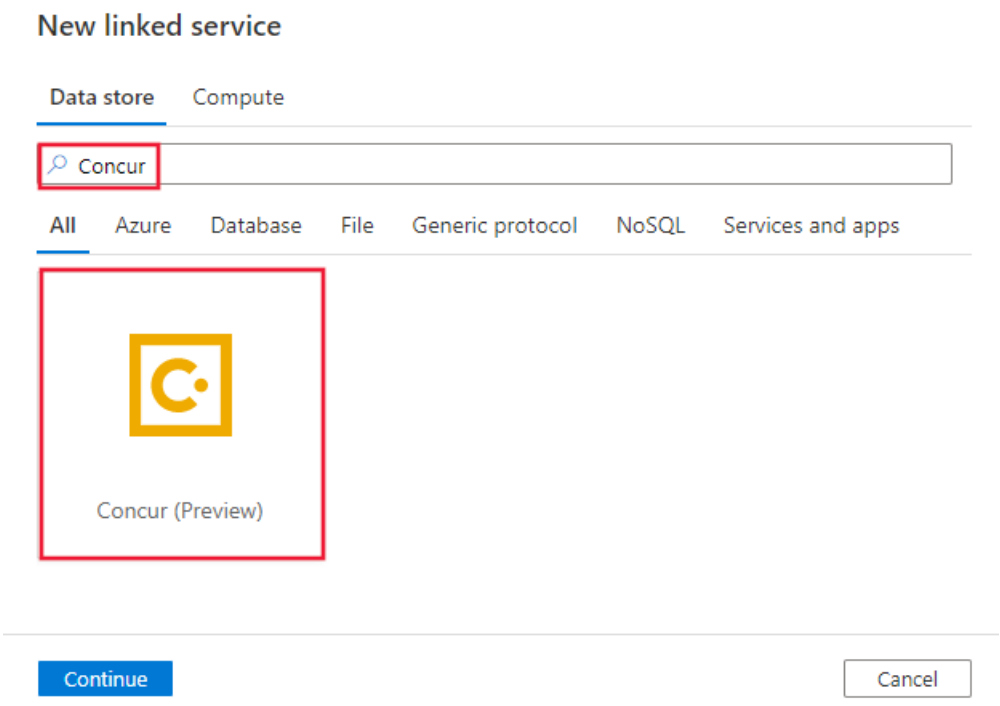
1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services,

then click New:

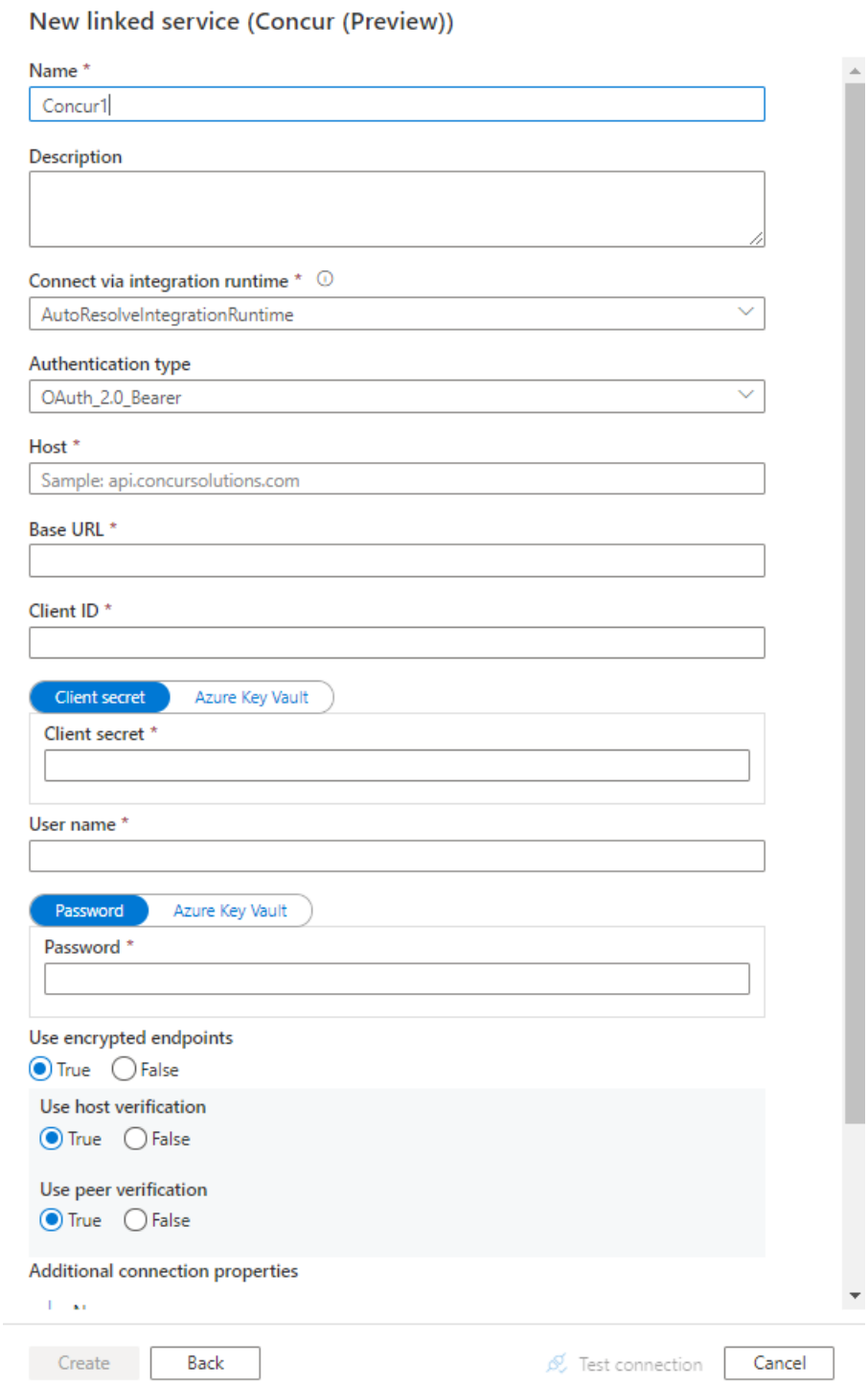
- Azure Data Factory
- Azure Synapse



2. Search for Concur and select the Concur connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Concur connector.

Linked service properties

The following properties are supported for Concur linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Concur**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionProperties	A group of properties that defines how to connect to Concur.	Yes
Under `connectionProperties`*:*		
authenticationType	Allowed values are `OAuth_2.0_Bearer` and `OAuth_2.0` (legacy). The OAuth 2.0 authentication option works with the old Concur API which was deprecated since Feb 2017.	Yes
host	The endpoint of the Concur server, e.g. `implementation.concursolutions.com`.	Yes
baseUrl	The base URL of your Concur's authorization URL.	Yes for `OAuth_2.0_Bearer` authentication
clientId	Application client ID supplied by Concur App Management.	Yes
clientSecret	The client secret corresponding to the client ID. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes for `OAuth_2.0_Bearer` authentication
username	The user name that you use to access Concur service.	Yes
password	The password corresponding to the user name that you provided in the username field. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
"name":"ConcurLinkedService",
"properties":{
"type":"Concur",
"typeProperties":{
"connectionProperties":{
"host":"<host e.g. implementation.concursolutions.com>",
"baseUrl":"<base URL for authorization e.g. us-impl.api.concursolutions.com>",
"authenticationType":"OAuth_2.0_Bearer",
"clientId":"<client id>",
"clientSecret":{
                    "type": "SecureString",
                    "value": "<client secret>"
                },
"username":"fakeUserName",
"password":{
                    "type": "SecureString",
                    "value": "<password>"
                },
"useEncryptedEndpoints":true,
"useHostVerification":true,
"usePeerVerification":true
}
}
}
}
```

Example (legacy):

Note the following is a legacy linked service model without `connectionProperties` and using `OAuth_2.0` authentication.

```
{
    "name": "ConcurLinkedService",
    "properties": {
        "type": "Concur",
        "typeProperties": {
            "clientId" : "<clientId>",
            "username" : "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Concur dataset.

To copy data from Concur, set the type property of the dataset to **ConcurObject**. There is no additional type-specific property in this type of dataset. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **ConcurObject**	Yes

PROPERTY	DESCRIPTION	REQUIRED
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "ConcurDataset",
    "properties": {
        "type": "ConcurObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Concur linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Concur source.

ConcurSource as source

To copy data from Concur, set the source type in the copy activity to **ConcurSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **ConcurSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM Opportunities where Id = xxx "`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromConcur",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Concur input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "ConcurSource",
                "query": "SELECT * FROM Opportunities where Id = xxx"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Couchbase using Azure Data Factory (Preview)

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Couchbase. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Couchbase connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

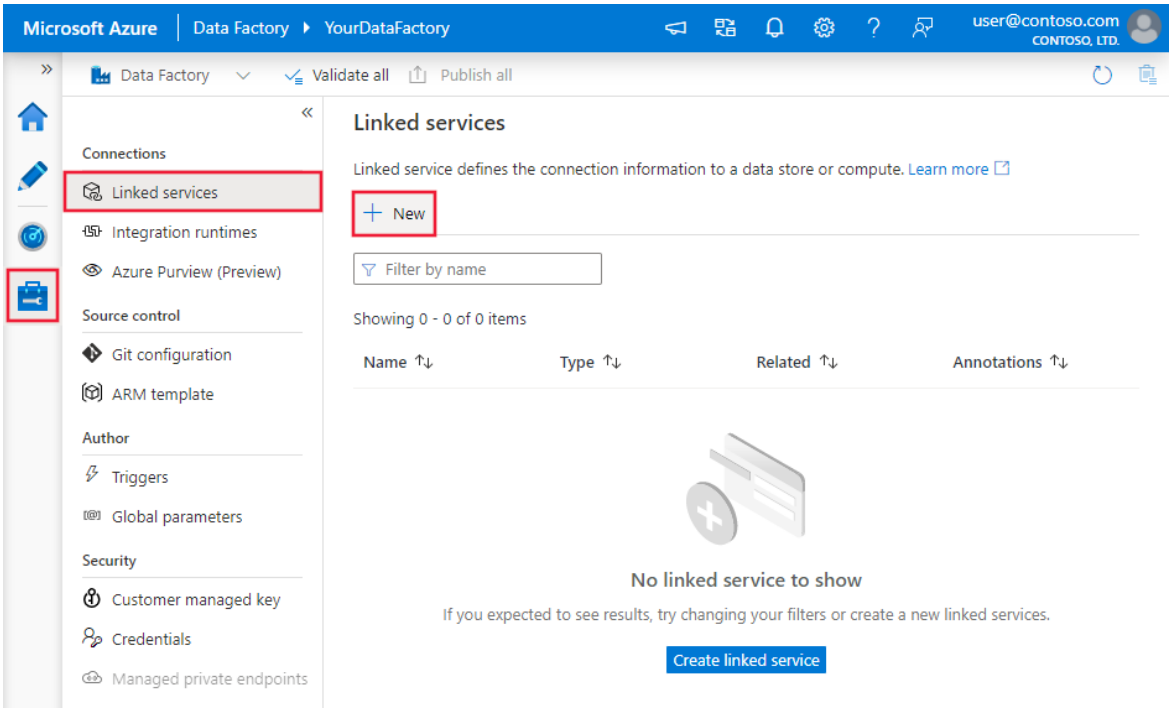
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Couchbase using UI

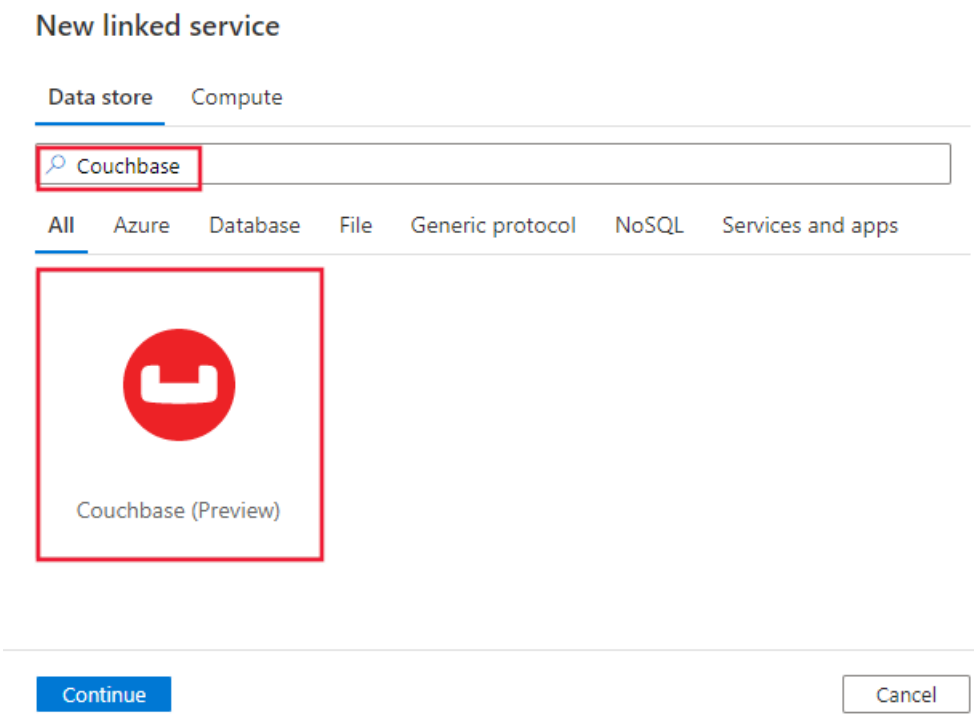
Use the following steps to create a linked service to Couchbase in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

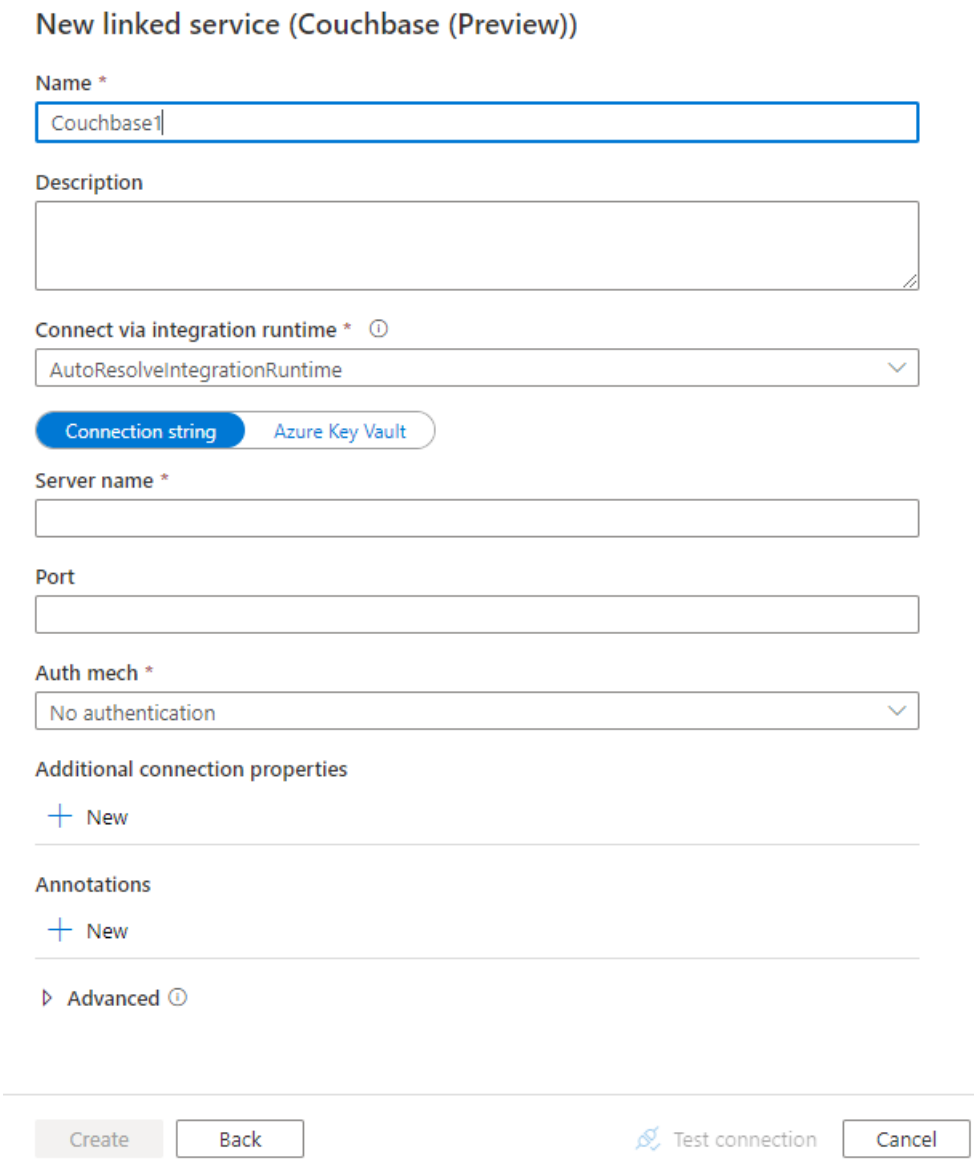
   - Azure Data Factory
   - Azure Synapse



2. Search for Couchbase and select the Couchbase connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Couchbase connector.

Linked service properties

The following properties are supported for Couchbase linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Couchbase**	Yes
connectionString	An ODBC connection string to connect to Couchbase. You can also put credential string in Azure Key Vault and pull the `credString` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "CouchbaseLinkedService",
    "properties": {
        "type": "Couchbase",
        "typeProperties": {
            "connectionString": "Server=<server>; Port=<port>;AuthMech=1;CredString=[{\"user\": \"JSmith\",
\"pass\":\"access123\"}, {\"user\": \"Admin\", \"pass\":\"simba123\"}];"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store credential string in Azure Key Vault

```
{
    "name": "CouchbaseLinkedService",
    "properties": {
        "type": "Couchbase",
        "typeProperties": {
            "connectionString": "Server=<server>; Port=<port>;AuthMech=1;",
            "credString": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Couchbase dataset.

To copy data from Couchbase, set the type property of the dataset to **CouchbaseTable**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **CouchbaseTable**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "CouchbaseDataset",
    "properties": {
        "type": "CouchbaseTable",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Couchbase linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Couchbase source.

CouchbaseSource as source

To copy data from Couchbase, set the source type in the copy activity to **CouchbaseSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **CouchbaseSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromCouchbase",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Couchbase input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "CouchbaseSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Transform data in data.world (Preview) using Azure Data Factory or Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Data Flow to transform data in data.world (Preview). To learn more, read the introductory article for Azure Data Factory or Azure Synapse Analytics.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This data.world connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Mapping data flow (source/-)	①

① Azure integration runtime ② Self-hosted integration runtime

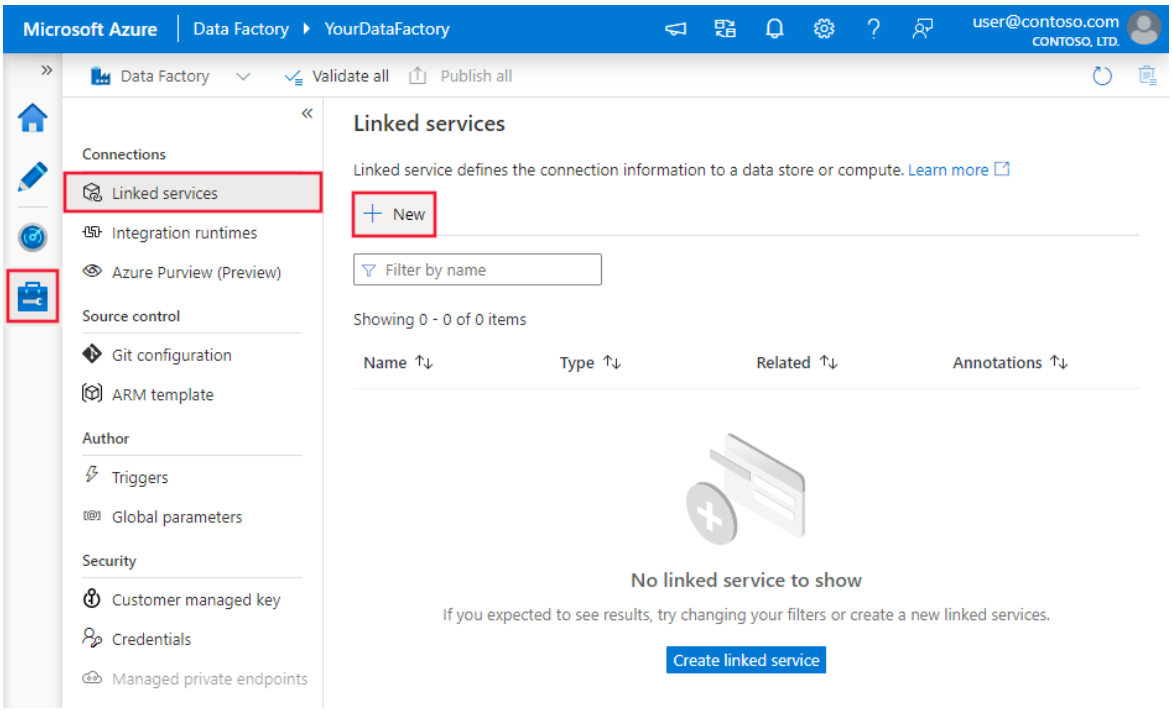
For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Create a data.world linked service using UI

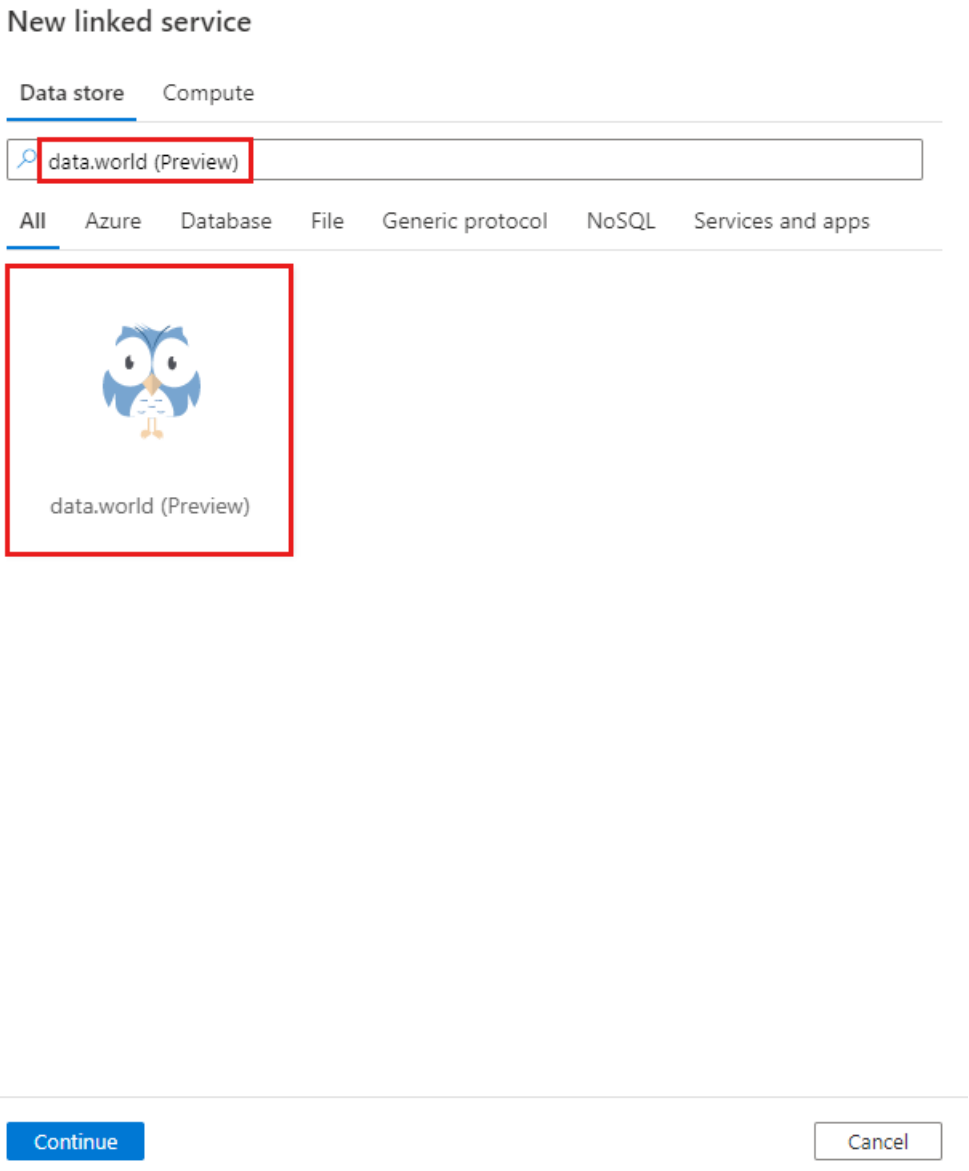
Use the following steps to create a data.world linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then select New:

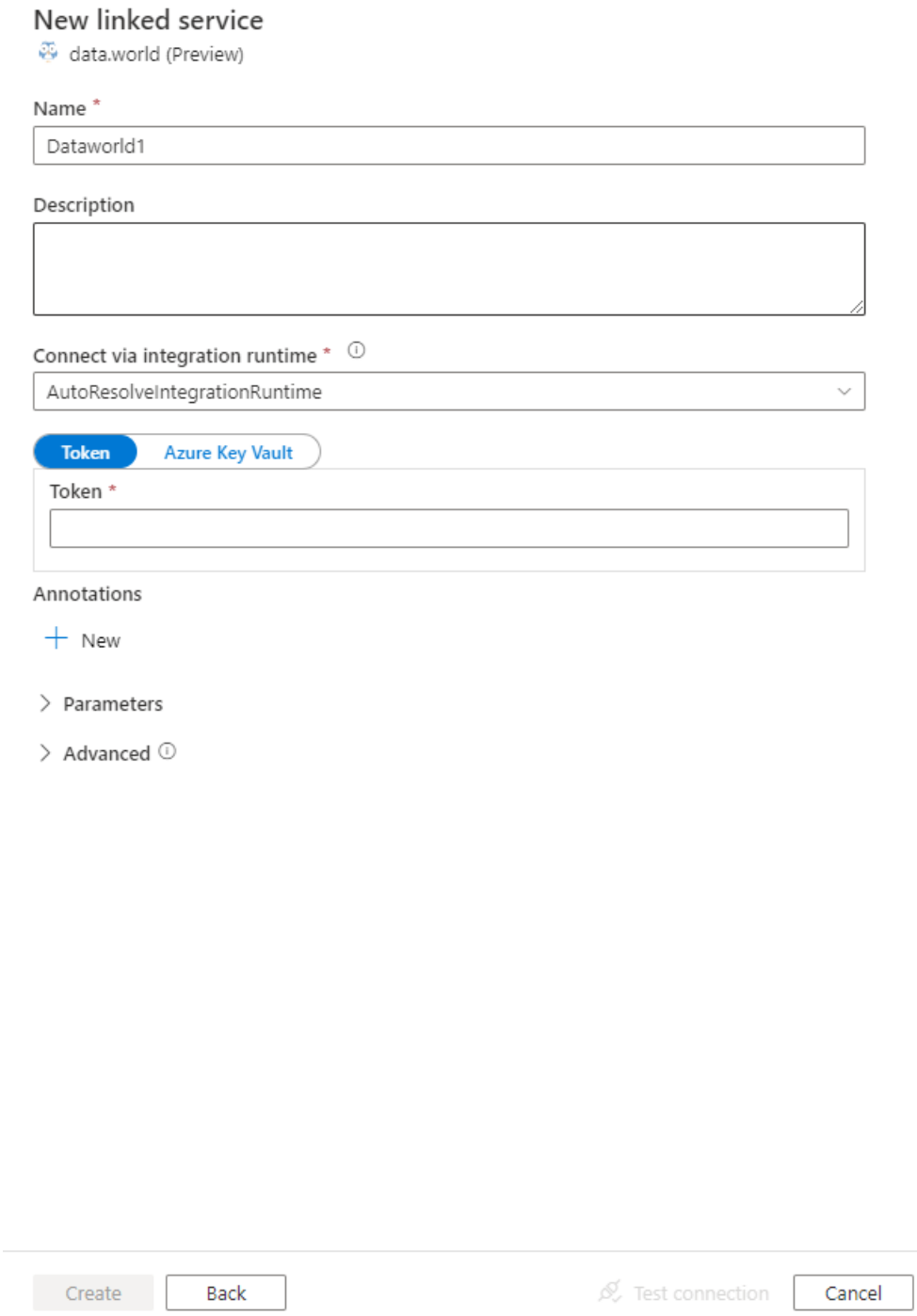
   - Azure Data Factory
   - Azure Synapse



2. Search for data.world (Preview) and select the data.world (Preview) connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide information about properties that are used to define Data Factory and Synapse pipeline entities specific to data.world.

Linked service properties

The following properties are supported for the data.world linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **Dataworld**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
apiToken	Specify an API token for the data.world. Mark this field as **SecureString** to store it securely. Or, you can reference a secret stored in Azure Key Vault.	Yes

Example:

```
{
    "name": "DataworldLinkedService",
    "properties": {
        "type": "Dataworld",
        "typeProperties": {
            "apiToken": {
                "type": "SecureString",
                "value": "<API token>"
            }
        }
    }
}
```

Mapping data flow properties

When transforming data in mapping data flow, you can read tables from data.world. For more information, see the source transformation in mapping data flows. You can only use an inline dataset as source type.

Source transformation

The below table lists the properties supported by data.world source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Dataset name	The ID of the dataset in data.world.	Yes	String	datasetId
Table name	The ID of the table within the dataset in data.world.	No (if `query` is specified)	String	tableId
Query	Enter a SQL query to fetch data from data.world. An example is `select * from MyTable`.	No (if `tableId` is specified)	String	query
Owner	The owner of the dataset in data.world.	Yes	String	owner

data.world source script example

When you use data.world as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 store: 'dataworld',
 format: 'rest',
 owner: 'owner1',
 datasetId: 'dataset1',
 tableId: 'MyTable') ~> DataworldSource
```

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy and transform data in Dynamics 365 (Microsoft Dataverse) or Dynamics CRM using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 17 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use a copy activity in Azure Data Factory or Synapse pipelines to copy data from and to Dynamics 365 (Microsoft Dataverse) or Dynamics CRM, and use a data flow to transform data in Dynamics 365 (Microsoft Dataverse) or Dynamics CRM. To learn more, read the Azure Data Factory and the Azure Synapse Analytics introduction articles.

Supported capabilities

This connector is supported for the following activities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Mapping data flow (source/sink)	①
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that a copy activity supports as sources and sinks, see the Supported data stores table.

> **NOTE**
> Effective November 2020, Common Data Service has been renamed to Microsoft Dataverse. This article is updated to reflect the latest terminology.

This Dynamics connector supports Dynamics versions 7 through 9 for both online and on-premises. More specifically:

- Version 7 maps to Dynamics CRM 2015.
- Version 8 maps to Dynamics CRM 2016 and the early version of Dynamics 365.
- Version 9 maps to the later version of Dynamics 365.

Refer to the following table of supported authentication types and configurations for Dynamics versions and products.

DYNAMICS VERSIONS	AUTHENTICATION TYPES	LINKED SERVICE SAMPLES
Dataverse Dynamics 365 online Dynamics CRM online	Azure Active Directory (Azure AD) service principal Office 365	Dynamics online and Azure AD service-principal or Office 365 authentication
Dynamics 365 on-premises with internet-facing deployment (IFD) Dynamics CRM 2016 on-premises with IFD Dynamics CRM 2015 on-premises with IFD	IFD	Dynamics on-premises with IFD and IFD authentication

NOTE

With the deprecation of regional Discovery Service, the service has upgraded to leverage global Discovery Service while using Office 365 Authentication.

IMPORTANT

If your tenant and user is configured in Azure Active Directory for conditional access and/or Multi-Factor Authentication is required, you will not be able to use Office 365 Authentication type. For those situations, you must use an Azure Active Directory (Azure AD) service principal authentication.

For Dynamics 365 specifically, the following application types are supported:

- Dynamics 365 for Sales
- Dynamics 365 for Customer Service
- Dynamics 365 for Field Service
- Dynamics 365 for Project Service Automation
- Dynamics 365 for Marketing This connector doesn't support other application types like Finance, Operations, and Talent.

TIP

To copy data from Dynamics 365 Finance and Operations, you can use the Dynamics AX connector.

This Dynamics connector is built on top of Dynamics XRM tooling.

Prerequisites

To use this connector with Azure AD service-principal authentication, you must set up server-to-server (S2S) authentication in Dataverse or Dynamics. First register the application user (Service Principal) in Azure Active Directory. You can find out how to do this here. During application registration you will need to create that user in Dataverse or Dynamics and grant permissions. Those permissions can either be granted directly or indirectly by adding the application user to a team which has been granted permissions in Dataverse or Dynamics. You can find more information on how to set up an application user to authenticate with Dataverse here.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

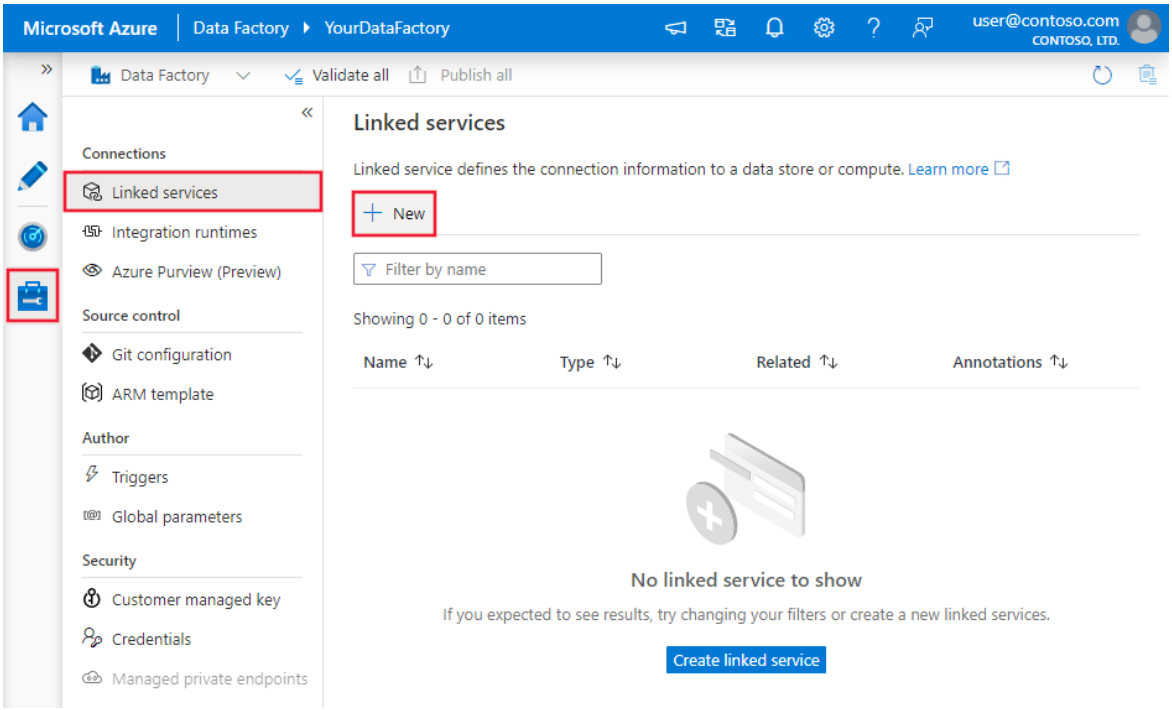
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Dynamics 365 (Microsoft Dataverse) or Dynamics CRM using UI

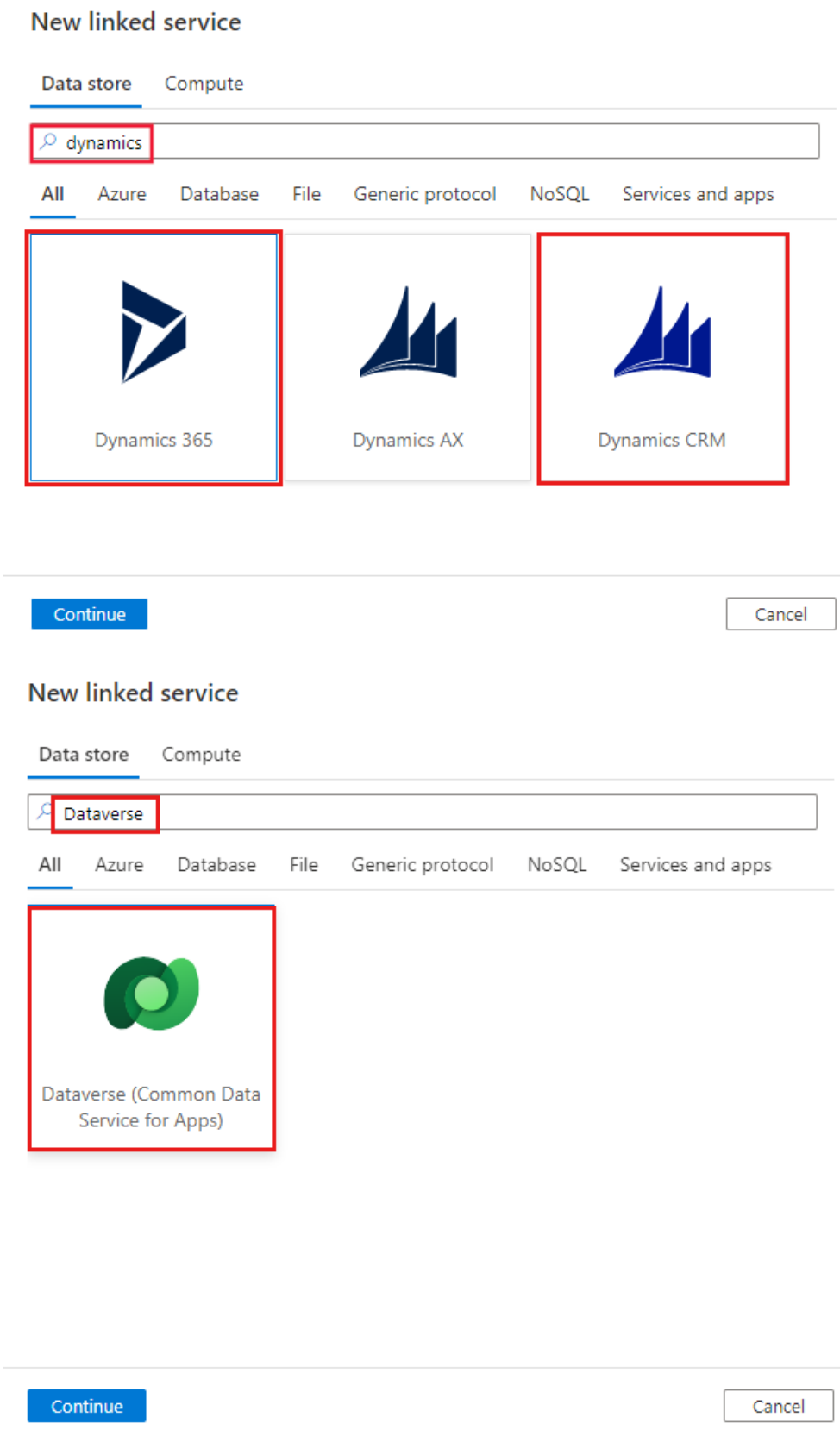
Use the following steps to create a linked service to Dynamics 365 in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

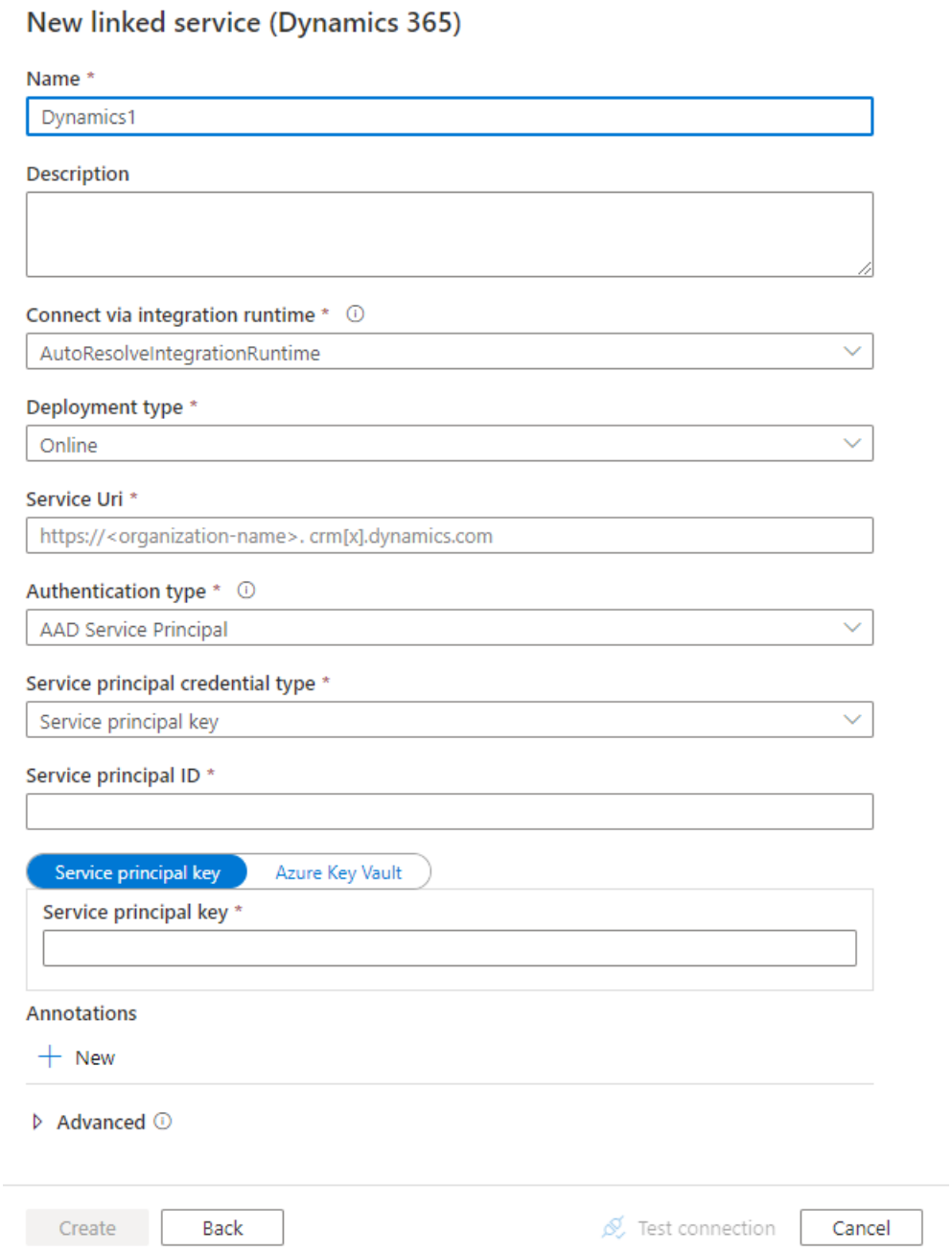
   - Azure Data Factory
   - Azure Synapse



2. Search for Dynamics or Dataverse and select the Dynamics 365 (Microsoft Dataverse) or Dynamics CRM connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to Dynamics.

Linked service properties

The following properties are supported for the Dynamics linked service.

Dynamics 365 and Dynamics CRM online

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to "Dynamics", "DynamicsCrm", or "CommonDataServiceForApps".	Yes
deploymentType	The deployment type of the Dynamics instance. The value must be "Online" for Dynamics online.	Yes

PROPERTY	DESCRIPTION	REQUIRED
serviceUri	The service URL of your Dynamics instance, the same one you access from browser. An example is "https://<organization-name>.crm[x].dynamics.com".	Yes
authenticationType	The authentication type to connect to a Dynamics server. Valid values are "AADServicePrincipal", "Office365" and "ManagedIdentity".	Yes
servicePrincipalId	The client ID of the Azure AD application.	Yes when authentication is "AADServicePrincipal"
servicePrincipalCredentialType	The credential type to use for service-principal authentication. Valid values are "ServicePrincipalKey" and "ServicePrincipalCert".	Yes when authentication is "AADServicePrincipal"
servicePrincipalCredential	The service-principal credential. When you use "ServicePrincipalKey" as the credential type, `servicePrincipalCredential` can be a string that the service encrypts upon linked service deployment. Or it can be a reference to a secret in Azure Key Vault. When you use "ServicePrincipalCert" as the credential, `servicePrincipalCredential` must be a reference to a certificate in Azure Key Vault, and ensure the certificate content type is **PKCS #12**.	Yes when authentication is "AADServicePrincipal"
username	The username to connect to Dynamics.	Yes when authentication is "Office365"
password	The password for the user account you specified as the username. Mark this field with "SecureString" to store it securely, or reference a secret stored in Azure Key Vault.	Yes when authentication is "Office365"
credentials	Specify the user-assigned managed identity as the credential object. Create one or multiple user-assigned managed identities, assign them to your data factory and create credentials for each user-assigned managed identity.	Yes when authentication is "ManagedIdentity"
connectVia	The integration runtime to be used to connect to the data store. If no value is specified, the property uses the default Azure integration runtime.	No

NOTE

The Dynamics connector formerly used the optional **organizationName** property to identify your Dynamics CRM or Dynamics 365 online instance. While that property still works, we suggest you specify the new **serviceUri** property instead to gain better performance for instance discovery.

Example: Dynamics online using Azure AD service-principal and key authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "AADServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalCredentialType": "ServicePrincipalKey",
            "servicePrincipalCredential": "<service principal key>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Dynamics online using Azure AD service-principal and certificate authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "AADServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalCredentialType": "ServicePrincipalCert",
            "servicePrincipalCredential": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<AKV reference>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<certificate name in AKV>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Dynamics online using Office 365 authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "Office365",
            "username": "test@contoso.onmicrosoft.com",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Dynamics online using user-assigned managed identity authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "ManagedIdentity",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dynamics 365 and Dynamics CRM on-premises with IFD

Additional properties that compare to Dynamics online are **hostName** and **port**.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to "Dynamics", "DynamicsCrm", or "CommonDataServiceForApps".	Yes.
deploymentType	The deployment type of the Dynamics instance. The value must be "OnPremisesWithIfd" for Dynamics on-premises with IFD.	Yes.
hostName	The host name of the on-premises Dynamics server.	Yes.

PROPERTY	DESCRIPTION	REQUIRED
port	The port of the on-premises Dynamics server.	No. The default value is 443.
organizationName	The organization name of the Dynamics instance.	Yes.
authenticationType	The authentication type to connect to the Dynamics server. Specify "Ifd" for Dynamics on-premises with IFD.	Yes.
username	The username to connect to Dynamics.	Yes.
password	The password for the user account you specified for the username. You can mark this field with "SecureString" to store it securely. Or you can store a password in Key Vault and let the copy activity pull from there when it does data copy. Learn more from Store credentials in Key Vault.	Yes.
connectVia	The integration runtime to be used to connect to the data store. If no value is specified, the property uses the default Azure integration runtime.	No

Example: Dynamics on-premises with IFD using IFD authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "description": "Dynamics on-premises with IFD linked service using IFD authentication",
        "typeProperties": {
            "deploymentType": "OnPremisesWithIFD",
            "hostName": "contosodynamicsserver.contoso.com",
            "port": 443,
            "organizationName": "admsDynamicsTest",
            "authenticationType": "Ifd",
            "username": "test@contoso.onmicrosoft.com",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by Dynamics dataset.

To copy data from and to Dynamics, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to "DynamicsEntity", "DynamicsCrmEntity", or "CommonDataServiceForAppsEntity".	Yes
entityName	The logical name of the entity to retrieve.	No for source if the activity source is specified as "query" and yes for sink

Example

```
{
    "name": "DynamicsDataset",
    "properties": {
        "type": "DynamicsEntity",
        "schema": [],
        "typeProperties": {
            "entityName": "account"
        },
        "linkedServiceName": {
            "referenceName": "<Dynamics linked service name>",
            "type": "linkedservicereference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Dynamics source and sink types.

Dynamics as a source type

To copy data from Dynamics, the copy activity **source** section supports the following properties:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to "DynamicsSource", "DynamicsCrmSource", or "CommonDataServiceForAppsSource".	Yes
query	FetchXML is a proprietary query language that is used in Dynamics online and on-premises. See the following example. To learn more, see Build queries with FetchXML.	No if `entityName` in the dataset is specified

> **NOTE**
> The PK column will always be copied out even if the column projection you configure in the FetchXML query doesn't contain it.

IMPORTANT

- When you copy data from Dynamics, explicit column mapping from Dynamics to sink is optional. But we highly recommend the mapping to ensure a deterministic copy result.
- When the service imports a schema in the authoring UI, it infers the schema. It does so by sampling the top rows from the Dynamics query result to initialize the source column list. In that case, columns with no values in the top rows are omitted. The same behavior also applies to data preview and copy executions if there is no explicit mapping. You can review and add more columns into the mapping, which are honored during copy runtime.

Example

```
"activities":[
    {
        "name": "CopyFromDynamics",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Dynamics input dataset>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DynamicsSource",
                "query": "<FetchXML Query>"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Sample FetchXML query

```
<fetch>
  <entity name="account">
    <attribute name="accountid" />
    <attribute name="name" />
    <attribute name="marketingonly" />
    <attribute name="modifiedon" />
    <order attribute="modifiedon" descending="false" />
    <filter type="and">
      <condition attribute ="modifiedon" operator="between">
        <value>2017-03-10 18:40:00z</value>
        <value>2017-03-12 20:40:00z</value>
      </condition>
    </filter>
  </entity>
</fetch>
```

Dynamics as a sink type

To copy data to Dynamics, the copy activity **sink** section supports the following properties:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to "DynamicsSink", "DynamicsCrmSink", or "CommonDataServiceForAppsSink".	Yes.
writeBehavior	The write behavior of the operation. The value must be "Upsert".	Yes
alternateKeyName	The alternate key name defined on your entity to do an upsert.	No.
writeBatchSize	The row count of data written to Dynamics in each batch.	No. The default value is 10.
ignoreNullValues	Whether to ignore null values from input data other than key fields during a write operation. Valid values are **TRUE** and **FALSE**: • **TRUE**: Leave the data in the destination object unchanged when you do an upsert or update operation. Insert a defined default value when you do an insert operation. • **FALSE**: Update the data in the destination object to a null value when you do an upsert or update operation. Insert a null value when you do an insert operation.	No. The default value is **FALSE**.
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

NOTE

The default value for both the sink **writeBatchSize** and the copy activity **parallelCopies** for the Dynamics sink is 10. Therefore, 100 records are concurrently submitted by default to Dynamics.

For Dynamics 365 online, there's a limit of two concurrent batch calls per organization. If that limit is exceeded, a "Server Busy" exception is thrown before the first request is ever run. Keep **writeBatchSize** at 10 or less to avoid such throttling of concurrent calls.

The optimal combination of **writeBatchSize** and **parallelCopies** depends on the schema of your entity. Schema elements include the number of columns, row size, and number of plug-ins, workflows, or workflow activities hooked up to those calls. The default setting of **writeBatchSize** (10) × **parallelCopies** (10) is the recommendation according to the Dynamics service. This value works for most Dynamics entities, although it might not give the best performance. You can tune the performance by adjusting the combination in your copy activity settings.

Example

```
"activities":[
    {
        "name": "CopyToDynamics",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Dynamics output dataset>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "DynamicsSink",
                "writeBehavior": "Upsert",
                "writeBatchSize": 10,
                "ignoreNullValues": true
            }
        }
    }
]
```

Retrieving data from views

To retrieve data from Dynamics views, you need to get the saved query of the view, and use the query to get the data.

There are two entities which store different types of view: "saved query" stores system view and "user query" stores user view. To get the information of the views, refer to the following FetchXML query and replace the "TARGETENTITY" with `savedquery` or `userquery`. Each entity type has more available attributes that you can add to the query based on your need. Learn more about savedquery entity and userquery entity.

```
<fetch top="5000" >
  <entity name="<TARGETENTITY>">
    <attribute name="name" />
    <attribute name="fetchxml" />
    <attribute name="returnedtypecode" />
    <attribute name="querytype" />
  </entity>
</fetch>
```

You can also add filters to filter the views. For example, add the following filter to get a view named "My Active Accounts" in account entity.

```
<filter type="and" >
    <condition attribute="returnedtypecode" operator="eq" value="1" />
    <condition attribute="name" operator="eq" value="My Active Accounts" />
</filter>
```

Data type mapping for Dynamics

When you copy data from Dynamics, the following table shows mappings from Dynamics data types to interim data types within the service. To learn how a copy activity maps to a source schema and a data type maps to a sink, see Schema and data type mappings.

Configure the corresponding interim data type in a dataset structure that is based on your source Dynamics data type by using the following mapping table:

DYNAMICS DATA TYPE	SERVICE INTERIM DATA TYPE	SUPPORTED AS SOURCE	SUPPORTED AS SINK
AttributeTypeCode.BigInt	Long	✓	✓
AttributeTypeCode.Boolean	Boolean	✓	✓
AttributeType.Customer	GUID	✓	✓ (See guidance)
AttributeType.DateTime	Datetime	✓	✓
AttributeType.Decimal	Decimal	✓	✓
AttributeType.Double	Double	✓	✓
AttributeType.EntityName	String	✓	✓
AttributeType.Integer	Int32	✓	✓
AttributeType.Lookup	GUID	✓	✓ (See guidance)
AttributeType.ManagedProperty	Boolean	✓	
AttributeType.Memo	String	✓	✓
AttributeType.Money	Decimal	✓	✓
AttributeType.Owner	GUID	✓	✓ (See guidance)
AttributeType.Picklist	Int32	✓	✓
AttributeType.Uniqueidentifier	GUID	✓	✓
AttributeType.String	String	✓	✓
AttributeType.State	Int32	✓	✓
AttributeType.Status	Int32	✓	✓

NOTE

The Dynamics data types **AttributeType.CalendarRules**, **AttributeType.MultiSelectPicklist**, and **AttributeType.PartyList** aren't supported.

Writing data to a lookup field

To write data into a lookup field with multiple targets like Customer and Owner, follow this guidance and example:

1. Make your source contains both the field value and the corresponding target entity name.
   - If all records map to the same target entity, ensure one of the following conditions:
     - Your source data has a column that stores the target entity name.
     - You've added an additional column in the copy activity source to define the target entity.
   - If different records map to different target entities, make sure your source data has a column that stores the corresponding target entity name.
2. Map both the value and entity-reference columns from source to sink. The entity-reference column must be mapped to a virtual column with the special naming pattern `{lookup_field_name}@EntityReference`. The column doesn't actually exist in Dynamics. It's used to indicate this column is the metadata column of the given multitarget lookup field.

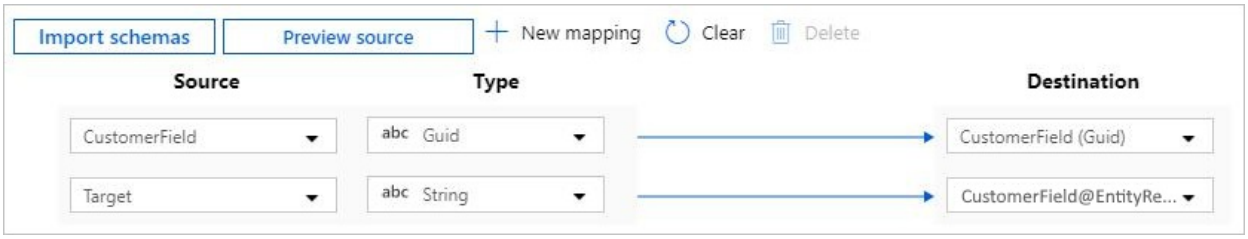
For example, assume the source has these two columns:

- **CustomerField** column of type **GUID**, which is the primary key value of the target entity in Dynamics.
- **Target** column of type **String**, which is the logical name of the target entity.

Also assume you want to copy such data to the sink Dynamics entity field **CustomerField** of type **Customer**.

In copy-activity column mapping, map the two columns as follows:

- **CustomerField** to **CustomerField**. This mapping is the normal field mapping.
- **Target** to **CustomerField@EntityReference**. The sink column is a virtual column representing the entity reference. Input such field names in a mapping, as they won't show up by importing schemas.



If all of your source records map to the same target entity and your source data doesn't contain the target entity name, here is a shortcut: in the copy activity source, add an additional column. Name the new column by using the pattern `{lookup_field_name}@EntityReference`, set the value to the target entity name, then proceed with column mapping as usual. If your source and sink column names are identical, you can also skip explicit column mapping because copy activity by default maps columns by name.

Mapping data flow properties

When transforming data in mapping data flow, you can read from and write to tables in Dynamics. For more information, see the source transformation and sink transformation in mapping data flows. You can choose to use a Dynamics dataset or an inline dataset as source and sink type.

Source transformation

The below table lists the properties supported by Dynamics. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Entity name	The logical name of the entity to retrieve.	Yes when use inline dataset	-	*(for inline dataset only)* entity
Query	FetchXML is a proprietary query language that is used in Dynamics online and on-premises. See the following example. To learn more, see Build queries with FetchXML.	No	String	query

> **NOTE**
> If you select **Query** as input type, the column type from tables can not be retrieved. It will be treated as string by default.

Dynamics source script example

When you use Dynamics dataset as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 query: '<fetch mapping='logical' count='3 paging-cookie=''><entity name='new_dataflow_crud_test'><attribute
name='new_name'/><attribute name='new_releasedate'/></entity></fetch>') ~> DynamicsSource
```

If you use inline dataset, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 store: 'dynamics',
 format: 'dynamicsformat',
 entity: 'Entity1',
 query: '<fetch mapping='logical' count='3 paging-cookie=''><entity name='new_dataflow_crud_test'><attribute
name='new_name'/><attribute name='new_releasedate'/></entity></fetch>') ~> DynamicsSource
```

Sink transformation

The below table lists the properties supported by Dynamics sink. You can edit these properties in the **Sink options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Alternate key name	The alternate key name defined on your entity to do an update, upsert or delete.	No	-	alternateKeyName

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Update method	Specify what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an Alter row transformation is required to tag rows for those actions.	Yes	`true` or `false`	insertable updateable upsertable deletable
Entity name	The logical name of the entity to write.	Yes when use inline dataset	-	*(for inline dataset only)* entity

Dynamics sink script example

When you use Dynamics dataset as sink type, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    deletable:true,
    insertable:true,
    updateable:true,
    upsertable:true,
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> DynamicsSink
```

If you use inline dataset, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    store: 'dynamics',
    format: 'dynamicsformat',
    entity: 'Entity1',
    deletable: true,
    insertable: true,
    updateable: true,
    upsertable: true,
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> DynamicsSink
```

Lookup activity properties

To learn details about the properties, see Lookup activity.

Next steps

For a list of supported data stores the copy activity as sources and sinks, see Supported data stores.

Copy data from DB2 using Azure Data Factory or Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from a DB2 database. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This DB2 connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks by the copy activity, see the Supported data stores table.

Specifically, this DB2 connector supports the following IBM DB2 platforms and versions with Distributed Relational Database Architecture (DRDA) SQL Access Manager (SQLAM) version 9, 10 and 11. It utilizes the DDM/DRDA protocol.

- IBM DB2 for z/OS 12.1
- IBM DB2 for z/OS 11.1
- IBM DB2 for z/OS 10.1
- IBM DB2 for i 7.3
- IBM DB2 for i 7.2
- IBM DB2 for i 7.1
- IBM DB2 for LUW 11
- IBM DB2 for LUW 10.5
- IBM DB2 for LUW 10.1

TIP

DB2 connector is built on top of Microsoft OLE DB Provider for DB2. To troubleshoot DB2 connector errors, refer to Data Provider Error Codes.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

The Integration Runtime provides a built-in DB2 driver, therefore you don't need to manually install any driver when copying data from DB2.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

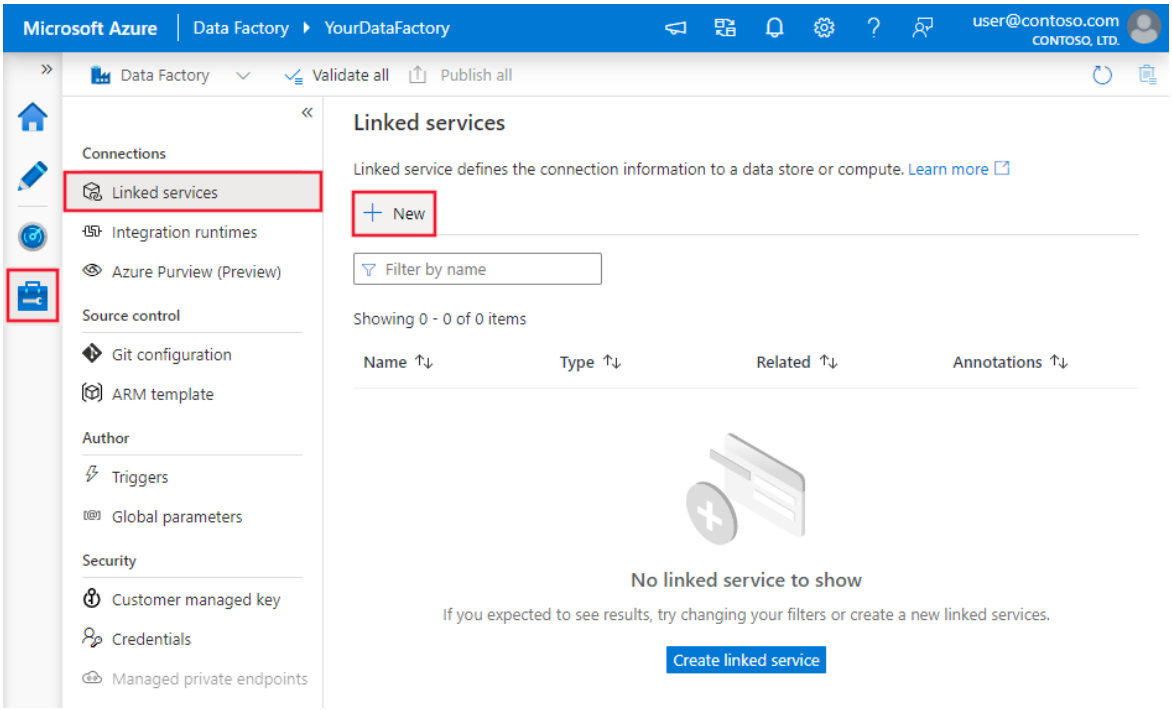
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to DB2 using UI

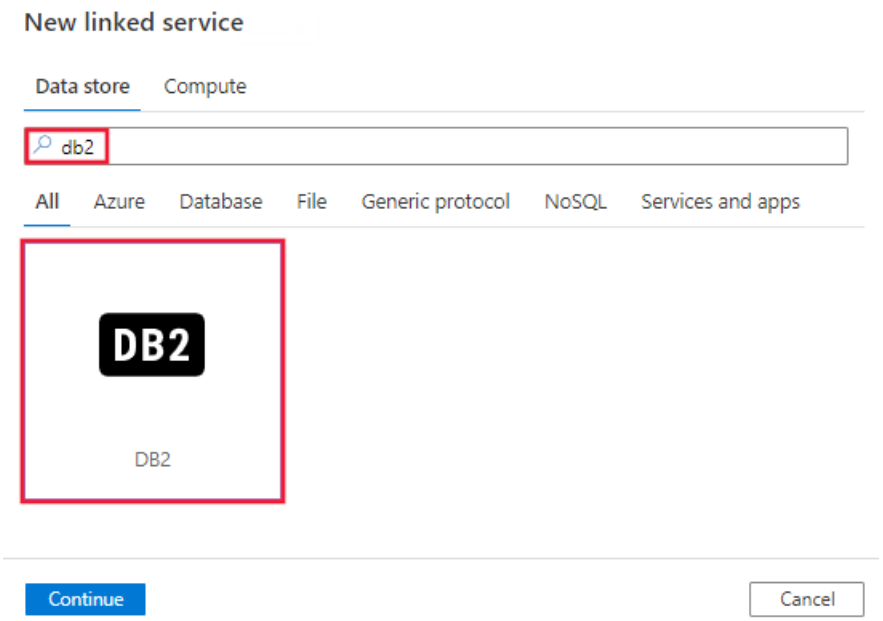
Use the following steps to create a linked service to DB2 in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

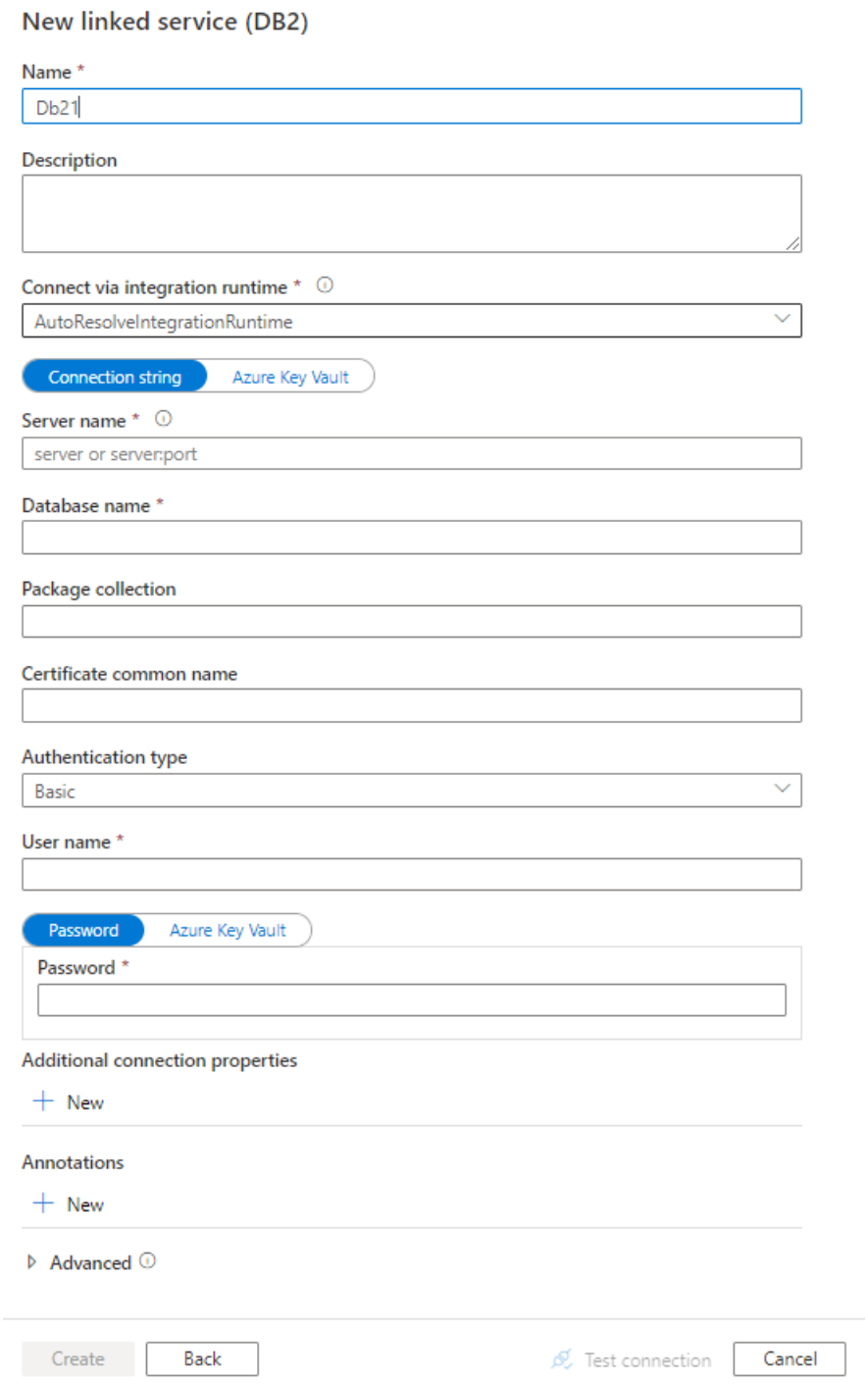
   - Azure Data Factory
   - Azure Synapse



2. Search for DB2 and select the DB2 connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to DB2 connector.

Linked service properties

The following properties are supported for DB2 linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Db2**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	Specify information needed to connect to the DB2 instance. You can also put password in Azure Key Vault and pull the `password` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Typical properties inside the connection string:

PROPERTY	DESCRIPTION	REQUIRED
server	Name of the DB2 server. You can specify the port number following the server name delimited by colon e.g. `server:port`. The DB2 connector utilizes the DDM/DRDA protocol, and by default uses port 50000 if not specified. The port your specific DB2 database uses might be different based on the version and your settings, e.g. for DB2 LUW the default port is 50000, for AS400 the default port is 446 or 448 when TLS enabled. Refer to the following DB2 documents on how the port is configured typically: DB2 z/OS, DB2 iSeries, and DB2 LUW.	Yes
database	Name of the DB2 database.	Yes
authenticationType	Type of authentication used to connect to the DB2 database. Allowed value is: **Basic**.	Yes
username	Specify user name to connect to the DB2 database.	Yes
password	Specify password for the user account you specified for the username. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
packageCollection	Specify under where the needed packages are auto created by the service when querying the database. If this is not set, the service uses the {username} as the default value.	No

PROPERTY	DESCRIPTION	REQUIRED
certificateCommonName	When you use Secure Sockets Layer (SSL) or Transport Layer Security (TLS) encryption, you must enter a value for Certificate common name.	No

TIP

If you receive an error message that states `The package corresponding to an SQL statement execution request was not found. SQLSTATE=51002 SQLCODE=-805`, the reason is a needed package is not created for the user. By default, the service will try to create the package under the collection named as the user you used to connect to the DB2. Specify the package collection property to indicate under where you want the service to create the needed packages when querying the database. If you can't determine the package collection name, try to set `packageCollection=NULLID`.

Example:

```
{
    "name": "Db2LinkedService",
    "properties": {
        "type": "Db2",
        "typeProperties": {
            "connectionString":"server=<server:port>;database=<database>;authenticationType=Basic;username=
<username>;password=<password>;packageCollection=<packagecollection>;certificateCommonName=<certname>;"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "Db2LinkedService",
    "properties": {
        "type": "Db2",
        "typeProperties": {
            "connectionString": "server=<server:port>;database=<database>;authenticationType=Basic;username=
<username>;packageCollection=<packagecollection>;certificateCommonName=<certname>;",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

If you were using DB2 linked service with the following payload, it is still supported as-is, while you are

suggested to use the new one going forward.

Previous payload:

```
{
    "name": "Db2LinkedService",
    "properties": {
        "type": "Db2",
        "typeProperties": {
            "server": "<servername:port>",
            "database": "<dbname>",
            "authenticationType": "Basic",
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by DB2 dataset.

To copy data from DB2, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **Db2Table**	Yes
schema	Name of the schema.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table with schema. This property is supported for backward compatibility. Use `schema` and `table` for new workload.	No (if "query" in activity source is specified)

Example

```
{
    "name": "DB2Dataset",
    "properties":
    {
        "type": "Db2Table",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<DB2 linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

If you were using `RelationalTable` typed dataset, it is still supported as-is, while you are suggested to use the new one going forward.

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by DB2 source.

DB2 as source

To copy data from DB2, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **Db2Source**	Yes
query	Use the custom SQL query to read data. For example: `"query": "SELECT * FROM \"DB2ADMIN\".\"Customers\""` .	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromDB2",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<DB2 input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "Db2Source",
                "query": "SELECT * FROM \"DB2ADMIN\".\"Customers\""
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

If you were using `RelationalSource` typed source, it is still supported as-is, while you are suggested to use the new one going forward.

Data type mapping for DB2

When copying data from DB2, the following mappings are used from DB2 data types to interim data types used internally within the service. See Schema and data type mappings to learn about how copy activity maps the source schema and data type to the sink.

DB2 DATABASE TYPE	INTERIM SERVICE DATA TYPE
BigInt	Int64
Binary	Byte[]
Blob	Byte[]
Char	String
Clob	String
Date	Datetime
DB2DynArray	String
DbClob	String
Decimal	Decimal

DB2 DATABASE TYPE	INTERIM SERVICE DATA TYPE
DecimalFloat	Decimal
Double	Double
Float	Double
Graphic	String
Integer	Int32
LongVarBinary	Byte[]
LongVarChar	String
LongVarGraphic	String
Numeric	Decimal
Real	Single
SmallInt	Int16
Time	TimeSpan
Timestamp	DateTime
VarBinary	Byte[]
VarChar	String
VarGraphic	String
Xml	Byte[]

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Delimited text format in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 9 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Follow this article when you want to **parse the delimited text files or write the data into delimited text format**.

Delimited text format is supported for the following connectors:

- Amazon S3
- Amazon S3 Compatible Storage
- Azure Blob
- Azure Data Lake Storage Gen1
- Azure Data Lake Storage Gen2
- Azure Files
- File System
- FTP
- Google Cloud Storage
- HDFS
- HTTP
- Oracle Cloud Storage
- SFTP

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the delimited text dataset.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **DelimitedText**.	Yes
location	Location settings of the file(s). Each file-based connector has its own location type and supported properties under `location`.	Yes
columnDelimiter	The character(s) used to separate columns in a file. The default value is **comma** `,`. When the column delimiter is defined as empty string, which means no delimiter, the whole line is taken as a single column. Currently, column delimiter as empty string is only supported for mapping data flow but not Copy activity.	No

PROPERTY	DESCRIPTION	REQUIRED
rowDelimiter	For Copy activity, the single character or "\r\n" used to separate rows in a file. The default value is any of the following values **on read: ["\r\n", "\r", "\n"]**; **on write: "\r\n"**. "\r\n" is only supported in copy command. For Mapping data flow, the single or two characters used to separate rows in a file. The default value is any of the following values **on read: ["\r\n", "\r", "\n"]**; **on write: "\n"**. When the row delimiter is set to no delimiter (empty string), the column delimiter must be set as no delimiter (empty string) as well, which means to treat the entire content as a single value. Currently, row delimiter as empty string is only supported for mapping data flow but not Copy activity.	No
quoteChar	The single character to quote column values if it contains column delimiter. The default value is **double quotes** `"`. When `quoteChar` is defined as empty string, it means there is no quote char and column value is not quoted, and `escapeChar` is used to escape the column delimiter and itself.	No
escapeChar	The single character to escape quotes inside a quoted value. The default value is **backslash** `\`. When `escapeChar` is defined as empty string, the `quoteChar` must be set as empty string as well, in which case make sure all column values don't contain delimiters.	No
firstRowAsHeader	Specifies whether to treat/make the first row as a header line with names of columns. Allowed values are **true** and **false** (default). When first row as header is false, note UI data preview and lookup activity output auto generate column names as Prop_{n} (starting from 0), copy activity requires explicit mapping from source to sink and locates columns by ordinal (starting from 1), and mapping data flow lists and locates columns with name as Column_{n} (starting from 1).	No
nullValue	Specifies the string representation of null value. The default value is **empty string**.	No

PROPERTY	DESCRIPTION	REQUIRED
encodingName	The encoding type used to read/write test files. Allowed values are as follows: "UTF-8","UTF-8 without BOM", "UTF-16", "UTF-16BE", "UTF-32", "UTF-32BE", "US-ASCII", "UTF-7", "BIG5", "EUC-JP", "EUC-KR", "GB2312", "GB18030", "JOHAB", "SHIFT-JIS", "CP875", "CP866", "IBM00858", "IBM037", "IBM273", "IBM437", "IBM500", "IBM737", "IBM775", "IBM850", "IBM852", "IBM855", "IBM857", "IBM860", "IBM861", "IBM863", "IBM864", "IBM865", "IBM869", "IBM870", "IBM01140", "IBM01141", "IBM01142", "IBM01143", "IBM01144", "IBM01145", "IBM01146", "IBM01147", "IBM01148", "IBM01149", "ISO-2022-JP", "ISO-2022-KR", "ISO-8859-1", "ISO-8859-2", "ISO-8859-3", "ISO-8859-4", "ISO-8859-5", "ISO-8859-6", "ISO-8859-7", "ISO-8859-8", "ISO-8859-9", "ISO-8859-13", "ISO-8859-15", "WINDOWS-874", "WINDOWS-1250", "WINDOWS-1251", "WINDOWS-1252", "WINDOWS-1253", "WINDOWS-1254", "WINDOWS-1255", "WINDOWS-1256", "WINDOWS-1257", "WINDOWS-1258". Note mapping data flow doesn't support UTF-7 encoding.	No
compressionCodec	The compression codec used to read/write text files. Allowed values are **bzip2**, **gzip**, **deflate**, **ZipDeflate**, **TarGzip**, **Tar**, **snappy**, or **lz4**. Default is not compressed. **Note** currently Copy activity doesn't support "snappy" & "lz4", and mapping data flow doesn't support "ZipDeflate", "TarGzip" and "Tar". **Note** when using copy activity to decompress **ZipDeflate/TarGzip/Tar** file(s) and write to file-based sink data store, by default files are extracted to the folder: `<path specified in dataset>/<folder named as source compressed file>/`, use `preserveZipFileNameAsFolder` / `preserveCompressionFileNameAsFolder` on copy activity source to control whether to preserve the name of the compressed file(s) as folder structure.	No

PROPERTY	DESCRIPTION	REQUIRED
compressionLevel	The compression ratio. Allowed values are **Optimal** or **Fastest**. - **Fastest:** The compression operation should complete as quickly as possible, even if the resulting file is not optimally compressed. - **Optimal**: The compression operation should be optimally compressed, even if the operation takes a longer time to complete. For more information, see Compression Level topic.	No

Below is an example of delimited text dataset on Azure Blob Storage:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<Azure Blob Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "container": "containername",
                "folderPath": "folder/subfolder",
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "escapeChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the delimited text source and sink.

Delimited text as source

The following properties are supported in the copy activity ***source*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **DelimitedTextSource**.	Yes
formatSettings	A group of properties. Refer to **Delimited text read settings** table below.	No

PROPERTY	DESCRIPTION	REQUIRED
storeSettings	A group of properties on how to read data from a data store. Each file-based connector has its own supported read settings under `storeSettings`.	No

Supported **delimited text read settings** under `formatSettings`:

PROPERTY	DESCRIPTION	REQUIRED
type	The type of formatSettings must be set to **DelimitedTextReadSettings**.	Yes
skipLineCount	Indicates the number of **non-empty** rows to skip when reading data from input files. If both skipLineCount and firstRowAsHeader are specified, the lines are skipped first and then the header information is read from the input file.	No
compressionProperties	A group of properties on how to decompress data for a given compression codec.	No
preserveZipFileNameAsFolder (*under* `compressionProperties` -> `type` *as* `ZipDeflateReadSettings`)	Applies when input dataset is configured with **ZipDeflate** compression. Indicates whether to preserve the source zip file name as folder structure during copy. - When set to **true (default)**, the service writes unzipped files to `<path specified in dataset>/<folder named as source zip file>/`. - When set to **false**, the service writes unzipped files directly to `<path specified in dataset>`. Make sure you don't have duplicated file names in different source zip files to avoid racing or unexpected behavior.	No

PROPERTY	DESCRIPTION	REQUIRED
preserveCompressionFileNameAsFolder (*under* `compressionProperties` -> `type` *as* `TarGZipReadSettings` *or* `TarReadSettings`)	Applies when input dataset is configured with **TarGzip/Tar** compression. Indicates whether to preserve the source compressed file name as folder structure during copy. - When set to **true (default)**, the service writes decompressed files to `<path specified in dataset>/<folder named as source compressed file>/`. - When set to **false**, the service writes decompressed files directly to `<path specified in dataset>`. Make sure you don't have duplicated file names in different source files to avoid racing or unexpected behavior.	No

```
"activities": [
    {
        "name": "CopyFromDelimitedText",
        "type": "Copy",
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "storeSettings": {
                    "type": "AzureBlobStorageReadSettings",
                    "recursive": true
                },
                "formatSettings": {
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 3,
                    "compressionProperties": {
                        "type": "ZipDeflateReadSettings",
                        "preserveZipFileNameAsFolder": false
                    }
                }
            },
            ...
        }
        ...
    }
]
```

Delimited text as sink

The following properties are supported in the copy activity ***sink*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **DelimitedTextSink**.	Yes
formatSettings	A group of properties. Refer to **Delimited text write settings** table below.	No

PROPERTY	DESCRIPTION	REQUIRED
storeSettings	A group of properties on how to write data to a data store. Each file-based connector has its own supported write settings under `storeSettings`.	No

Supported **delimited text write settings** under `formatSettings`:

PROPERTY	DESCRIPTION	REQUIRED
type	The type of formatSettings must be set to **DelimitedTextWriteSettings**.	Yes
fileExtension	The file extension used to name the output files, for example, `.csv`, `.txt`. It must be specified when the `fileName` is not specified in the output DelimitedText dataset. When file name is configured in the output dataset, it will be used as the sink file name and the file extension setting will be ignored.	Yes when file name is not specified in output dataset
maxRowsPerFile	When writing data into a folder, you can choose to write to multiple files and specify the max rows per file.	No
fileNamePrefix	Applicable when `maxRowsPerFile` is configured. Specify the file name prefix when writing data to multiple files, resulted in this pattern: `<fileNamePrefix>_00000.<fileExtension>`. If not specified, file name prefix will be auto generated. This property does not apply when source is file-based store or partition-option-enabled data store.	No

Mapping data flow properties

In mapping data flows, you can read and write to delimited text format in the following data stores: Azure Blob Storage, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2 and SFTP, and you can read delimited text format in Amazon S3.

Source properties

The below table lists the properties supported by a delimited text source. You can edit these properties in the **Source options** tab.

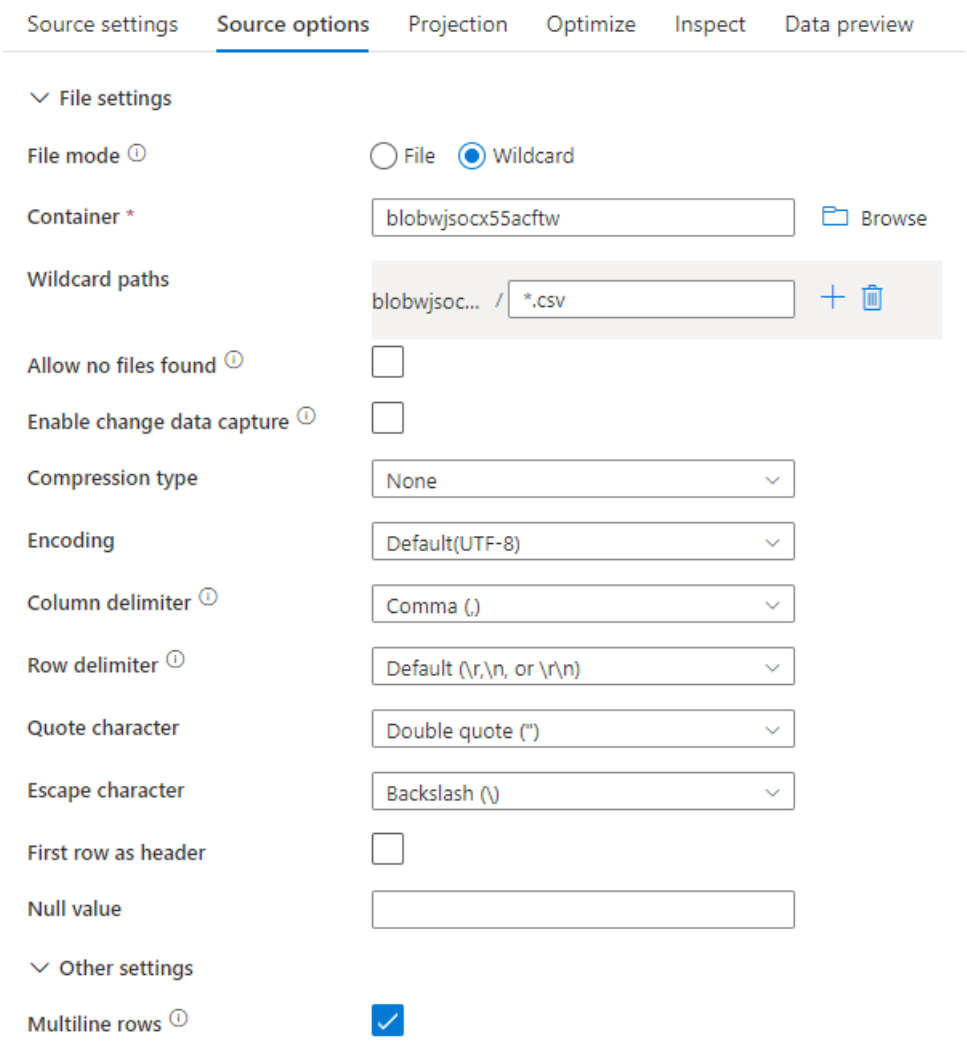
NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Wild card paths	All files matching the wildcard path will be processed. Overrides the folder and file path set in the dataset.	no	String[]	wildcardPaths
Partition root path	For file data that is partitioned, you can enter a partition root path in order to read partitioned folders as columns	no	String	partitionRootPath
List of files	Whether your source is pointing to a text file that lists files to process	no	`true` or `false`	fileList
Multiline rows	Does the source file contain rows that span multiple lines. Multiline values must be in quotes.	no `true` or `false`	multiLineRow	
Column to store file name	Create a new column with the source file name and path	no	String	rowUrlColumn
After completion	Delete or move the files after processing. File path starts from the container root	no	Delete: `true` or `false` Move: `['<from>', '<to>']`	purgeFiles moveFiles
Filter by last modified	Choose to filter files based upon when they were last altered	no	Timestamp	modifiedAfter modifiedBefore
Allow no files found	If true, an error is not thrown if no files are found	no	`true` or `false`	ignoreNoFilesFound

> **NOTE**
> Data flow sources support for list of files is limited to 1024 entries in your file. To include more files, use wildcards in your file list.

Source example

The below image is an example of a delimited text source configuration in mapping data flows.



The associated data flow script is:

```
source(
    allowSchemaDrift: true,
    validateSchema: false,
    multiLineRow: true,
    wildcardPaths:['*.csv']) ~> CSVSource
```

> **NOTE**
> Data flow sources support a limited set of Linux globbing that is support by Hadoop file systems

Sink properties

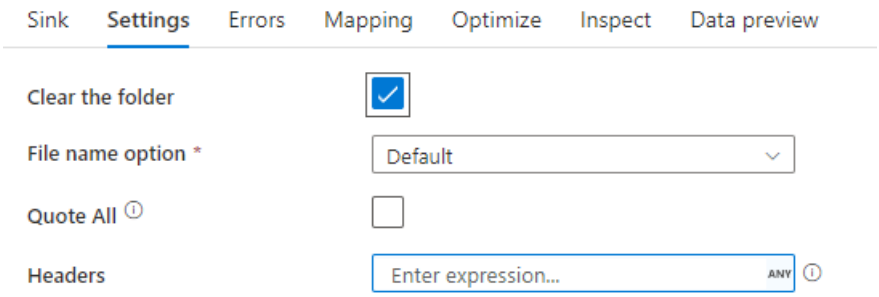
The below table lists the properties supported by a delimited text sink. You can edit these properties in the **Settings** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Clear the folder	If the destination folder is cleared prior to write	no	`true` or `false`	truncate

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
File name option	The naming format of the data written. By default, one file per partition in format `part-#####-tid-<guid>`	no	Pattern: String Per partition: String[] Name file as column data: String Output to single file: `['<fileName>']` Name folder as column data: String	filePattern partitionFileNames rowUrlColumn partitionFileNames rowFolderUrlColumn
Quote all	Enclose all values in quotes	no	`true` or `false`	quoteAll
Header	Add customer headers to output files	no	`[<string array>]`	header

Sink example

The below image is an example of a delimited text sink configuration in mapping data flows.



The associated data flow script is:

```
CSVSource sink(allowSchemaDrift: true,
    validateSchema: false,
    truncate: true,
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> CSVSink
```

Related connectors and formats

Here are some common connectors and formats related to the delimited text format:

- Azure Blob Storage (connector-azure-blob-storage.md)
- Binary format (format-binary.md)
- Dataverse(connector-dynamics-crm-office-365.md)
- Delta format(format-delta.md)
- Excel format(format-excel.md)
- File System(connector-file-system.md)
- FTP(connector-ftp.md)
- HTTP(connector-http.md)
- JSON format(format-json.md)
- Parquet format(format-parquet.md)

Next steps

- Copy activity overview
- Mapping data flow
- Lookup activity
- GetMetadata activity

Copy data from Drill using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Drill. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Drill connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell

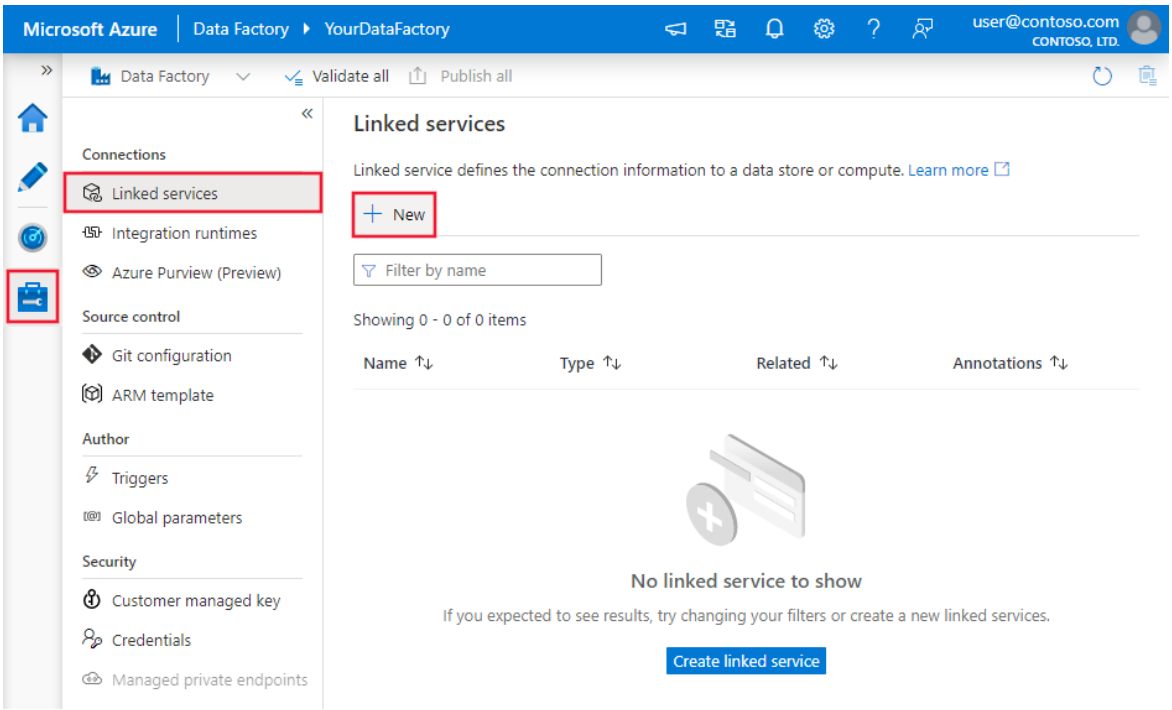
- The REST API
- The Azure Resource Manager template

Create a linked service to Drill using UI

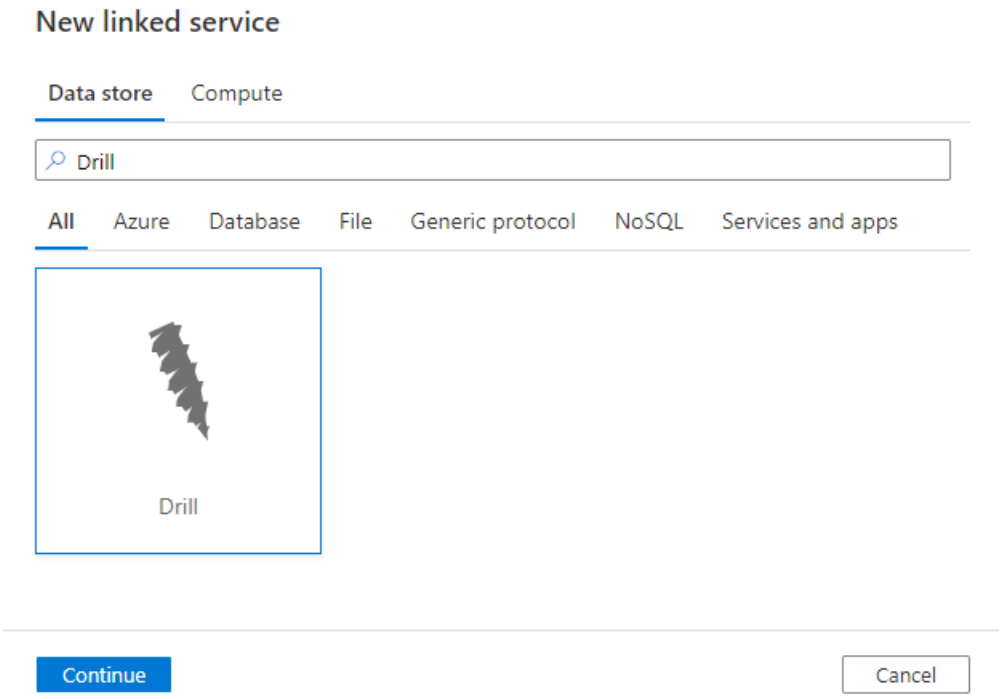
Use the following steps to create a linked service to Drill in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

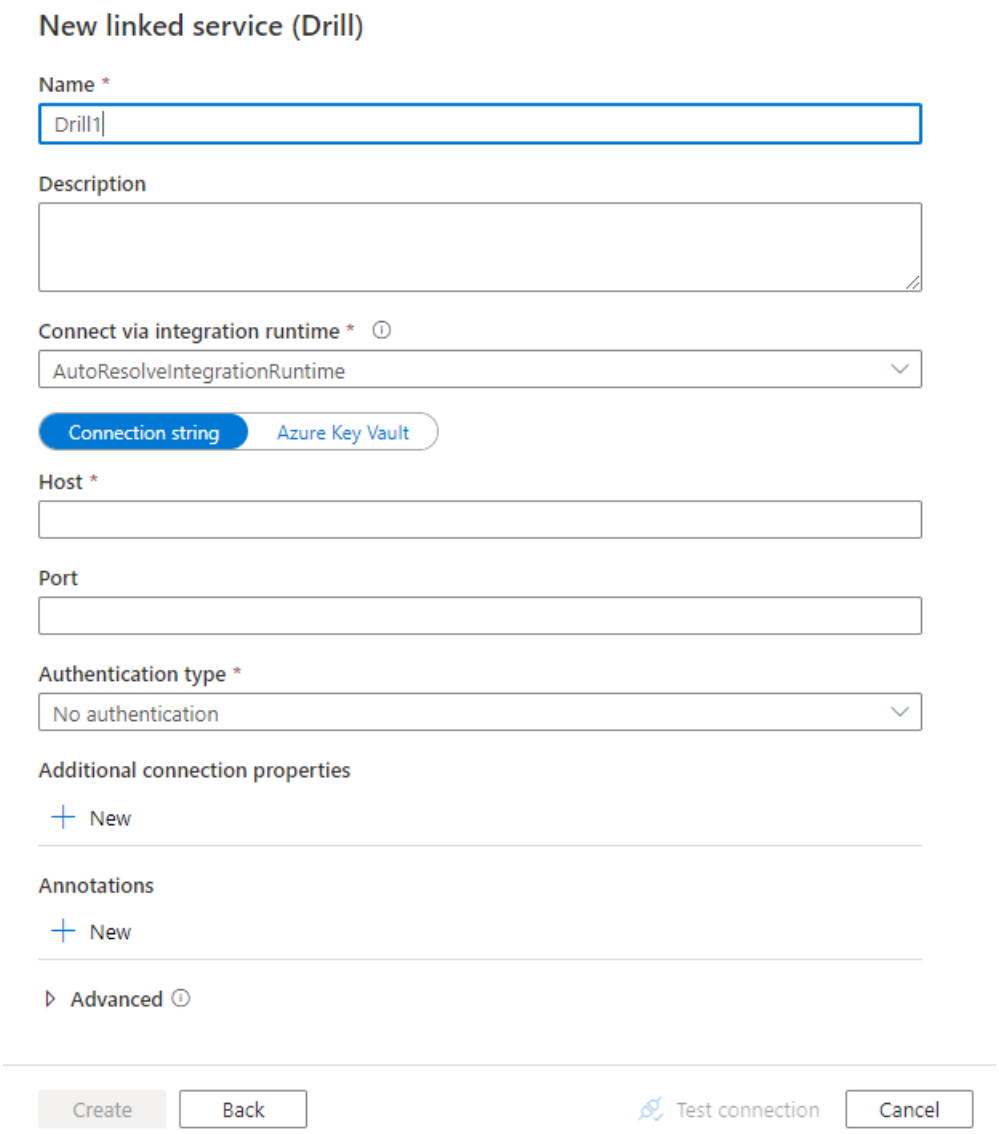
   - Azure Data Factory
   - Azure Synapse



2. Search for Drill and select the Drill connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Drill connector.

Linked service properties

The following properties are supported for Drill linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Drill**	Yes
connectionString	An ODBC connection string to connect to Drill. You can also put password in Azure Key Vault and pull the `pwd` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "DrillLinkedService",
    "properties": {
        "type": "Drill",
        "typeProperties": {
            "connectionString": "ConnectionType=Direct;Host=<host>;Port=<port>;AuthenticationType=Plain;UID=
<user name>;PWD=<password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "DrillLinkedService",
    "properties": {
        "type": "Drill",
        "typeProperties": {
            "connectionString": "ConnectionType=Direct;Host=<host>;Port=<port>;AuthenticationType=Plain;UID=
<user name>;",
            "pwd": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Drill dataset.

To copy data from Drill, set the type property of the dataset to **DrillTable**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **DrillTable**	Yes
schema	Name of the schema.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table with schema. This property is supported for backward compatibility. Use `schema` and `table` for new workload.	No (if "query" in activity source is specified)

Example

```
{
    "name": "DrillDataset",
    "properties": {
        "type": "DrillTable",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Drill linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Drill source.

DrillSource as source

To copy data from Drill, set the source type in the copy activity to **DrillSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **DrillSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromDrill",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Drill input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DrillSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy and transform data in Dynamics 365 (Microsoft Dataverse) or Dynamics CRM using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 17 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use a copy activity in Azure Data Factory or Synapse pipelines to copy data from and to Dynamics 365 (Microsoft Dataverse) or Dynamics CRM, and use a data flow to transform data in Dynamics 365 (Microsoft Dataverse) or Dynamics CRM. To learn more, read the Azure Data Factory and the Azure Synapse Analytics introduction articles.

Supported capabilities

This connector is supported for the following activities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Mapping data flow (source/sink)	①
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that a copy activity supports as sources and sinks, see the Supported data stores table.

> **NOTE**
> Effective November 2020, Common Data Service has been renamed to Microsoft Dataverse. This article is updated to reflect the latest terminology.

This Dynamics connector supports Dynamics versions 7 through 9 for both online and on-premises. More specifically:

- Version 7 maps to Dynamics CRM 2015.
- Version 8 maps to Dynamics CRM 2016 and the early version of Dynamics 365.
- Version 9 maps to the later version of Dynamics 365.

Refer to the following table of supported authentication types and configurations for Dynamics versions and products.

DYNAMICS VERSIONS	AUTHENTICATION TYPES	LINKED SERVICE SAMPLES
Dataverse Dynamics 365 online Dynamics CRM online	Azure Active Directory (Azure AD) service principal Office 365	Dynamics online and Azure AD service-principal or Office 365 authentication
Dynamics 365 on-premises with internet-facing deployment (IFD) Dynamics CRM 2016 on-premises with IFD Dynamics CRM 2015 on-premises with IFD	IFD	Dynamics on-premises with IFD and IFD authentication

NOTE

With the deprecation of regional Discovery Service, the service has upgraded to leverage global Discovery Service while using Office 365 Authentication.

IMPORTANT

If your tenant and user is configured in Azure Active Directory for conditional access and/or Multi-Factor Authentication is required, you will not be able to use Office 365 Authentication type. For those situations, you must use an Azure Active Directory (Azure AD) service principal authentication.

For Dynamics 365 specifically, the following application types are supported:

- Dynamics 365 for Sales
- Dynamics 365 for Customer Service
- Dynamics 365 for Field Service
- Dynamics 365 for Project Service Automation
- Dynamics 365 for Marketing This connector doesn't support other application types like Finance, Operations, and Talent.

TIP

To copy data from Dynamics 365 Finance and Operations, you can use the Dynamics AX connector.

This Dynamics connector is built on top of Dynamics XRM tooling.

Prerequisites

To use this connector with Azure AD service-principal authentication, you must set up server-to-server (S2S) authentication in Dataverse or Dynamics. First register the application user (Service Principal) in Azure Active Directory. You can find out how to do this here. During application registration you will need to create that user in Dataverse or Dynamics and grant permissions. Those permissions can either be granted directly or indirectly by adding the application user to a team which has been granted permissions in Dataverse or Dynamics. You can find more information on how to set up an application user to authenticate with Dataverse here.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

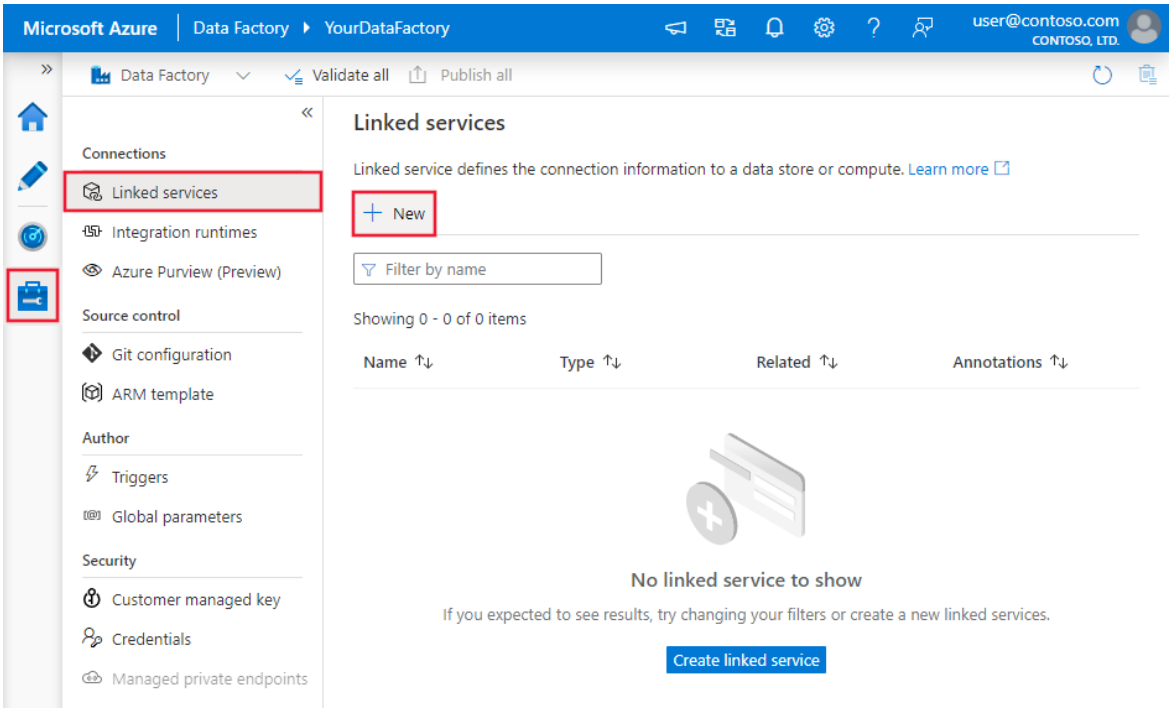
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Dynamics 365 (Microsoft Dataverse) or Dynamics CRM using UI

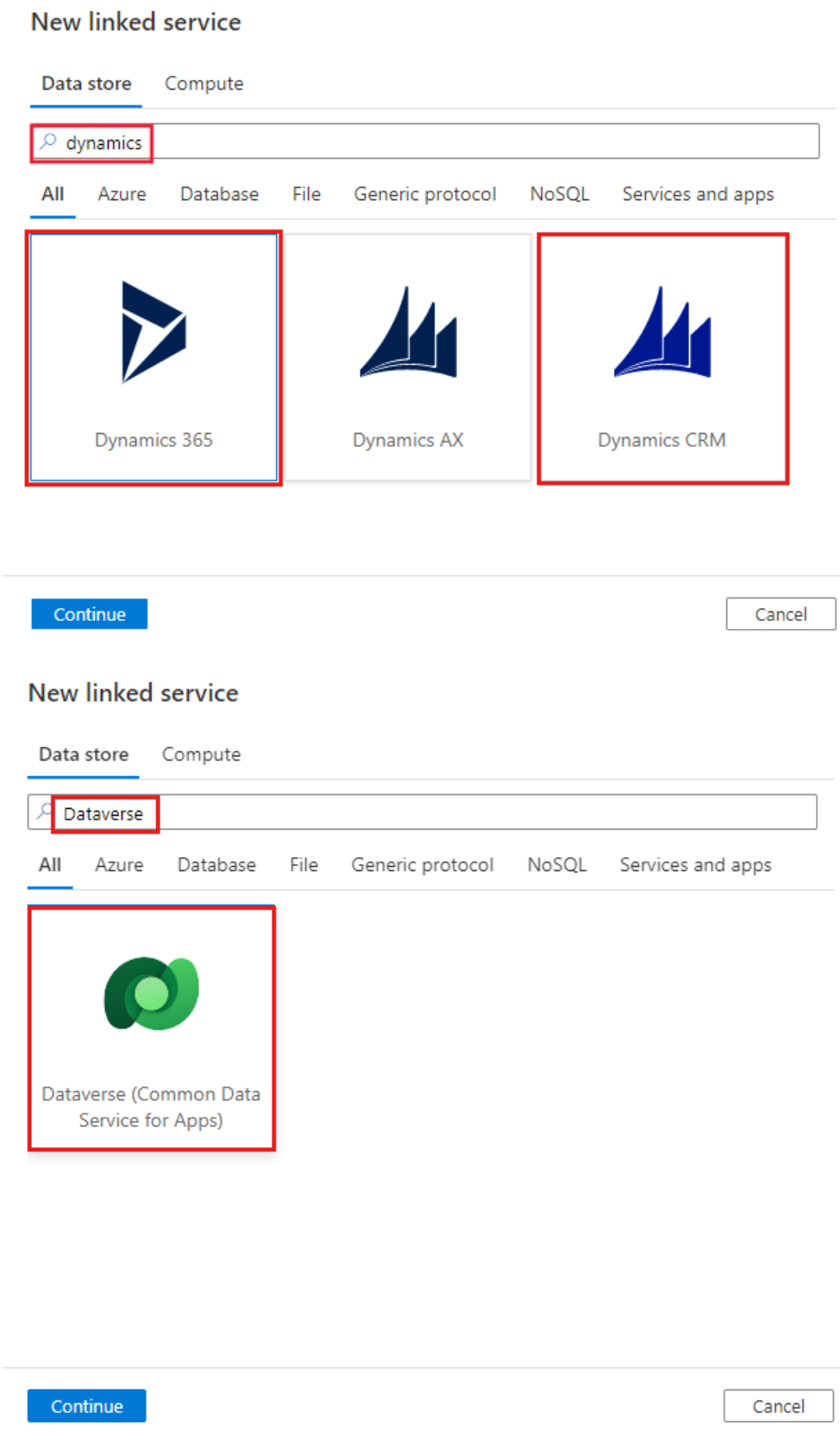
Use the following steps to create a linked service to Dynamics 365 in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

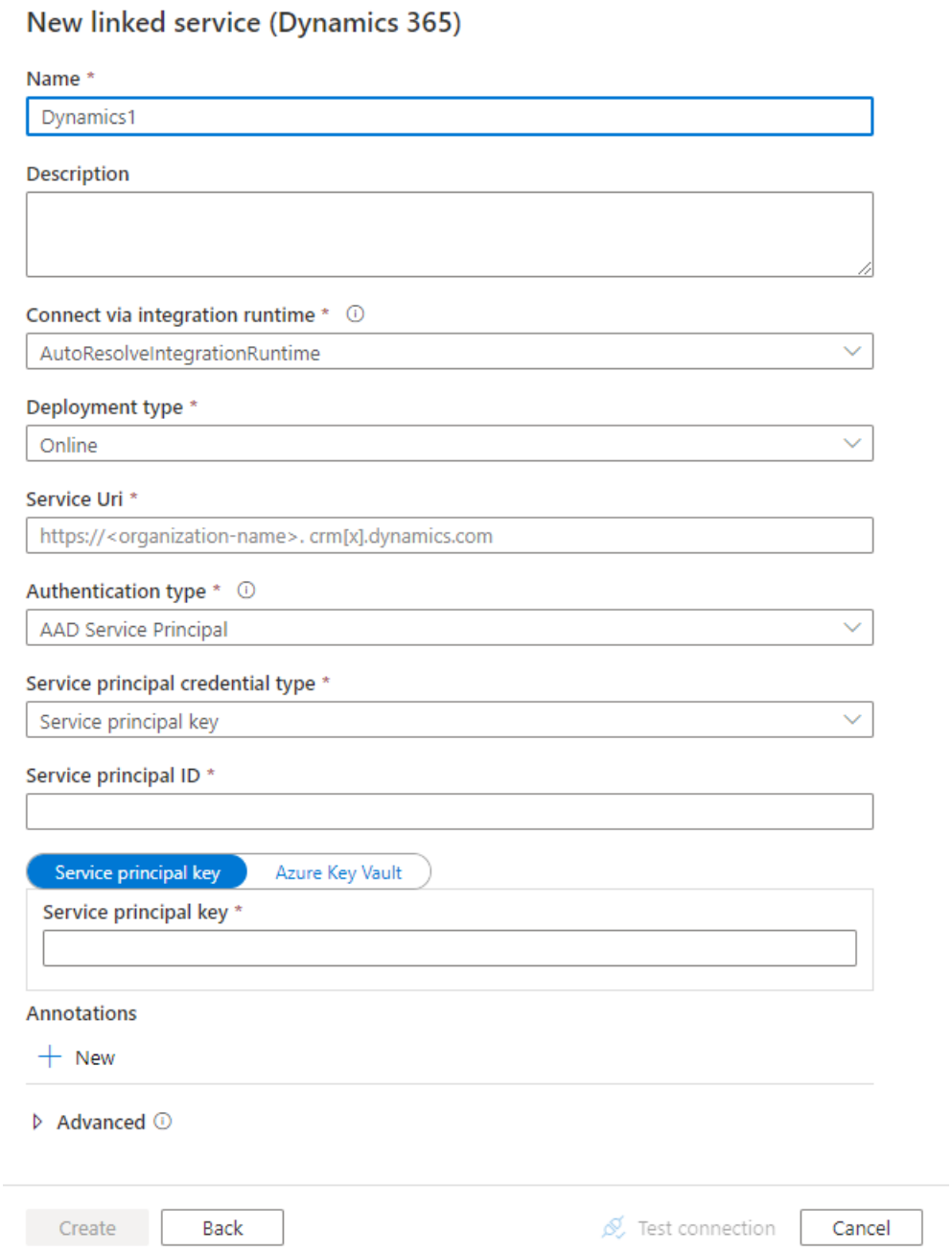
   - Azure Data Factory
   - Azure Synapse



2. Search for Dynamics or Dataverse and select the Dynamics 365 (Microsoft Dataverse) or Dynamics CRM connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to Dynamics.

Linked service properties

The following properties are supported for the Dynamics linked service.

Dynamics 365 and Dynamics CRM online

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to "Dynamics", "DynamicsCrm", or "CommonDataServiceForApps".	Yes
deploymentType	The deployment type of the Dynamics instance. The value must be "Online" for Dynamics online.	Yes

PROPERTY	DESCRIPTION	REQUIRED
serviceUri	The service URL of your Dynamics instance, the same one you access from browser. An example is "https://<organization-name>.crm[x].dynamics.com".	Yes
authenticationType	The authentication type to connect to a Dynamics server. Valid values are "AADServicePrincipal", "Office365" and "ManagedIdentity".	Yes
servicePrincipalId	The client ID of the Azure AD application.	Yes when authentication is "AADServicePrincipal"
servicePrincipalCredentialType	The credential type to use for service-principal authentication. Valid values are "ServicePrincipalKey" and "ServicePrincipalCert".	Yes when authentication is "AADServicePrincipal"
servicePrincipalCredential	The service-principal credential. When you use "ServicePrincipalKey" as the credential type, `servicePrincipalCredential` can be a string that the service encrypts upon linked service deployment. Or it can be a reference to a secret in Azure Key Vault. When you use "ServicePrincipalCert" as the credential, `servicePrincipalCredential` must be a reference to a certificate in Azure Key Vault, and ensure the certificate content type is **PKCS #12**.	Yes when authentication is "AADServicePrincipal"
username	The username to connect to Dynamics.	Yes when authentication is "Office365"
password	The password for the user account you specified as the username. Mark this field with "SecureString" to store it securely, or reference a secret stored in Azure Key Vault.	Yes when authentication is "Office365"
credentials	Specify the user-assigned managed identity as the credential object. Create one or multiple user-assigned managed identities, assign them to your data factory and create credentials for each user-assigned managed identity.	Yes when authentication is "ManagedIdentity"
connectVia	The integration runtime to be used to connect to the data store. If no value is specified, the property uses the default Azure integration runtime.	No

NOTE

The Dynamics connector formerly used the optional **organizationName** property to identify your Dynamics CRM or Dynamics 365 online instance. While that property still works, we suggest you specify the new **serviceUri** property instead to gain better performance for instance discovery.

Example: Dynamics online using Azure AD service-principal and key authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "AADServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalCredentialType": "ServicePrincipalKey",
            "servicePrincipalCredential": "<service principal key>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Dynamics online using Azure AD service-principal and certificate authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "AADServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalCredentialType": "ServicePrincipalCert",
            "servicePrincipalCredential": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<AKV reference>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<certificate name in AKV>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Dynamics online using Office 365 authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "Office365",
            "username": "test@contoso.onmicrosoft.com",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Dynamics online using user-assigned managed identity authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "ManagedIdentity",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dynamics 365 and Dynamics CRM on-premises with IFD

Additional properties that compare to Dynamics online are **hostName** and **port**.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to "Dynamics", "DynamicsCrm", or "CommonDataServiceForApps".	Yes.
deploymentType	The deployment type of the Dynamics instance. The value must be "OnPremisesWithIfd" for Dynamics on-premises with IFD.	Yes.
hostName	The host name of the on-premises Dynamics server.	Yes.

PROPERTY	DESCRIPTION	REQUIRED
port	The port of the on-premises Dynamics server.	No. The default value is 443.
organizationName	The organization name of the Dynamics instance.	Yes.
authenticationType	The authentication type to connect to the Dynamics server. Specify "Ifd" for Dynamics on-premises with IFD.	Yes.
username	The username to connect to Dynamics.	Yes.
password	The password for the user account you specified for the username. You can mark this field with "SecureString" to store it securely. Or you can store a password in Key Vault and let the copy activity pull from there when it does data copy. Learn more from Store credentials in Key Vault.	Yes.
connectVia	The integration runtime to be used to connect to the data store. If no value is specified, the property uses the default Azure integration runtime.	No

Example: Dynamics on-premises with IFD using IFD authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "description": "Dynamics on-premises with IFD linked service using IFD authentication",
        "typeProperties": {
            "deploymentType": "OnPremisesWithIFD",
            "hostName": "contosodynamicsserver.contoso.com",
            "port": 443,
            "organizationName": "admsDynamicsTest",
            "authenticationType": "Ifd",
            "username": "test@contoso.onmicrosoft.com",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by Dynamics dataset.

To copy data from and to Dynamics, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to "DynamicsEntity", "DynamicsCrmEntity", or "CommonDataServiceForAppsEntity".	Yes
entityName	The logical name of the entity to retrieve.	No for source if the activity source is specified as "query" and yes for sink

Example

```
{
    "name": "DynamicsDataset",
    "properties": {
        "type": "DynamicsEntity",
        "schema": [],
        "typeProperties": {
            "entityName": "account"
        },
        "linkedServiceName": {
            "referenceName": "<Dynamics linked service name>",
            "type": "linkedservicereference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Dynamics source and sink types.

Dynamics as a source type

To copy data from Dynamics, the copy activity **source** section supports the following properties:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to "DynamicsSource", "DynamicsCrmSource", or "CommonDataServiceForAppsSource".	Yes
query	FetchXML is a proprietary query language that is used in Dynamics online and on-premises. See the following example. To learn more, see Build queries with FetchXML.	No if `entityName` in the dataset is specified

> **NOTE**
> The PK column will always be copied out even if the column projection you configure in the FetchXML query doesn't contain it.

IMPORTANT

- When you copy data from Dynamics, explicit column mapping from Dynamics to sink is optional. But we highly recommend the mapping to ensure a deterministic copy result.
- When the service imports a schema in the authoring UI, it infers the schema. It does so by sampling the top rows from the Dynamics query result to initialize the source column list. In that case, columns with no values in the top rows are omitted. The same behavior also applies to data preview and copy executions if there is no explicit mapping. You can review and add more columns into the mapping, which are honored during copy runtime.

Example

```
"activities":[
    {
        "name": "CopyFromDynamics",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Dynamics input dataset>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DynamicsSource",
                "query": "<FetchXML Query>"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Sample FetchXML query

```
<fetch>
  <entity name="account">
    <attribute name="accountid" />
    <attribute name="name" />
    <attribute name="marketingonly" />
    <attribute name="modifiedon" />
    <order attribute="modifiedon" descending="false" />
    <filter type="and">
      <condition attribute ="modifiedon" operator="between">
        <value>2017-03-10 18:40:00z</value>
        <value>2017-03-12 20:40:00z</value>
      </condition>
    </filter>
  </entity>
</fetch>
```

Dynamics as a sink type

To copy data to Dynamics, the copy activity **sink** section supports the following properties:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to "DynamicsSink", "DynamicsCrmSink", or "CommonDataServiceForAppsSink".	Yes.
writeBehavior	The write behavior of the operation. The value must be "Upsert".	Yes
alternateKeyName	The alternate key name defined on your entity to do an upsert.	No.
writeBatchSize	The row count of data written to Dynamics in each batch.	No. The default value is 10.
ignoreNullValues	Whether to ignore null values from input data other than key fields during a write operation. Valid values are **TRUE** and **FALSE**: • **TRUE**: Leave the data in the destination object unchanged when you do an upsert or update operation. Insert a defined default value when you do an insert operation. • **FALSE**: Update the data in the destination object to a null value when you do an upsert or update operation. Insert a null value when you do an insert operation.	No. The default value is **FALSE**.
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

NOTE

The default value for both the sink **writeBatchSize** and the copy activity **parallelCopies** for the Dynamics sink is 10. Therefore, 100 records are concurrently submitted by default to Dynamics.

For Dynamics 365 online, there's a limit of two concurrent batch calls per organization. If that limit is exceeded, a "Server Busy" exception is thrown before the first request is ever run. Keep **writeBatchSize** at 10 or less to avoid such throttling of concurrent calls.

The optimal combination of **writeBatchSize** and **parallelCopies** depends on the schema of your entity. Schema elements include the number of columns, row size, and number of plug-ins, workflows, or workflow activities hooked up to those calls. The default setting of **writeBatchSize** (10) × **parallelCopies** (10) is the recommendation according to the Dynamics service. This value works for most Dynamics entities, although it might not give the best performance. You can tune the performance by adjusting the combination in your copy activity settings.

Example

```
"activities":[
    {
        "name": "CopyToDynamics",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Dynamics output dataset>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "DynamicsSink",
                "writeBehavior": "Upsert",
                "writeBatchSize": 10,
                "ignoreNullValues": true
            }
        }
    }
]
```

Retrieving data from views

To retrieve data from Dynamics views, you need to get the saved query of the view, and use the query to get the data.

There are two entities which store different types of view: "saved query" stores system view and "user query" stores user view. To get the information of the views, refer to the following FetchXML query and replace the "TARGETENTITY" with `savedquery` or `userquery`. Each entity type has more available attributes that you can add to the query based on your need. Learn more about savedquery entity and userquery entity.

```
<fetch top="5000" >
  <entity name="<TARGETENTITY>">
    <attribute name="name" />
    <attribute name="fetchxml" />
    <attribute name="returnedtypecode" />
    <attribute name="querytype" />
  </entity>
</fetch>
```

You can also add filters to filter the views. For example, add the following filter to get a view named "My Active Accounts" in account entity.

```
<filter type="and" >
    <condition attribute="returnedtypecode" operator="eq" value="1" />
    <condition attribute="name" operator="eq" value="My Active Accounts" />
</filter>
```

Data type mapping for Dynamics

When you copy data from Dynamics, the following table shows mappings from Dynamics data types to interim data types within the service. To learn how a copy activity maps to a source schema and a data type maps to a sink, see Schema and data type mappings.

Configure the corresponding interim data type in a dataset structure that is based on your source Dynamics data type by using the following mapping table:

DYNAMICS DATA TYPE	SERVICE INTERIM DATA TYPE	SUPPORTED AS SOURCE	SUPPORTED AS SINK
AttributeTypeCode.BigInt	Long	✓	✓
AttributeTypeCode.Boolean	Boolean	✓	✓
AttributeType.Customer	GUID	✓	✓ (See guidance)
AttributeType.DateTime	Datetime	✓	✓
AttributeType.Decimal	Decimal	✓	✓
AttributeType.Double	Double	✓	✓
AttributeType.EntityName	String	✓	✓
AttributeType.Integer	Int32	✓	✓
AttributeType.Lookup	GUID	✓	✓ (See guidance)
AttributeType.ManagedProperty	Boolean	✓	
AttributeType.Memo	String	✓	✓
AttributeType.Money	Decimal	✓	✓
AttributeType.Owner	GUID	✓	✓ (See guidance)
AttributeType.Picklist	Int32	✓	✓
AttributeType.Uniqueidentifier	GUID	✓	✓
AttributeType.String	String	✓	✓
AttributeType.State	Int32	✓	✓
AttributeType.Status	Int32	✓	✓

NOTE

The Dynamics data types **AttributeType.CalendarRules**, **AttributeType.MultiSelectPicklist**, and **AttributeType.PartyList** aren't supported.

Writing data to a lookup field

To write data into a lookup field with multiple targets like Customer and Owner, follow this guidance and example:

1. Make your source contains both the field value and the corresponding target entity name.
   - If all records map to the same target entity, ensure one of the following conditions:
     - Your source data has a column that stores the target entity name.
     - You've added an additional column in the copy activity source to define the target entity.
   - If different records map to different target entities, make sure your source data has a column that stores the corresponding target entity name.
2. Map both the value and entity-reference columns from source to sink. The entity-reference column must be mapped to a virtual column with the special naming pattern `{lookup_field_name}@EntityReference`. The column doesn't actually exist in Dynamics. It's used to indicate this column is the metadata column of the given multitarget lookup field.

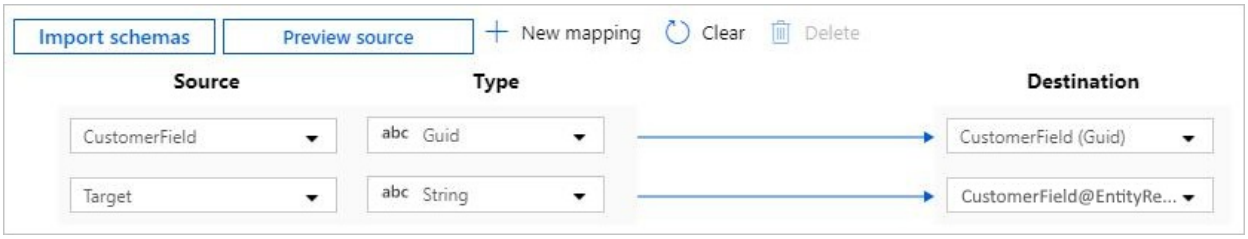
For example, assume the source has these two columns:

- **CustomerField** column of type **GUID**, which is the primary key value of the target entity in Dynamics.
- **Target** column of type **String**, which is the logical name of the target entity.

Also assume you want to copy such data to the sink Dynamics entity field **CustomerField** of type **Customer**.

In copy-activity column mapping, map the two columns as follows:

- **CustomerField** to **CustomerField**. This mapping is the normal field mapping.
- **Target** to **CustomerField@EntityReference**. The sink column is a virtual column representing the entity reference. Input such field names in a mapping, as they won't show up by importing schemas.



If all of your source records map to the same target entity and your source data doesn't contain the target entity name, here is a shortcut: in the copy activity source, add an additional column. Name the new column by using the pattern `{lookup_field_name}@EntityReference`, set the value to the target entity name, then proceed with column mapping as usual. If your source and sink column names are identical, you can also skip explicit column mapping because copy activity by default maps columns by name.

Mapping data flow properties

When transforming data in mapping data flow, you can read from and write to tables in Dynamics. For more information, see the source transformation and sink transformation in mapping data flows. You can choose to use a Dynamics dataset or an inline dataset as source and sink type.

Source transformation

The below table lists the properties supported by Dynamics. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Entity name	The logical name of the entity to retrieve.	Yes when use inline dataset	-	*(for inline dataset only)* entity
Query	FetchXML is a proprietary query language that is used in Dynamics online and on-premises. See the following example. To learn more, see Build queries with FetchXML.	No	String	query

> **NOTE**
> If you select **Query** as input type, the column type from tables can not be retrieved. It will be treated as string by default.

Dynamics source script example

When you use Dynamics dataset as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 query: '<fetch mapping='logical' count='3 paging-cookie=''><entity name='new_dataflow_crud_test'><attribute
name='new_name'/><attribute name='new_releasedate'/></entity></fetch>') ~> DynamicsSource
```

If you use inline dataset, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 store: 'dynamics',
 format: 'dynamicsformat',
 entity: 'Entity1',
 query: '<fetch mapping='logical' count='3 paging-cookie=''><entity name='new_dataflow_crud_test'><attribute
name='new_name'/><attribute name='new_releasedate'/></entity></fetch>') ~> DynamicsSource
```

Sink transformation

The below table lists the properties supported by Dynamics sink. You can edit these properties in the **Sink options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Alternate key name	The alternate key name defined on your entity to do an update, upsert or delete.	No	-	alternateKeyName

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Update method	Specify what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an Alter row transformation is required to tag rows for those actions.	Yes	`true` or `false`	insertable updateable upsertable deletable
Entity name	The logical name of the entity to write.	Yes when use inline dataset	-	*(for inline dataset only)* entity

Dynamics sink script example

When you use Dynamics dataset as sink type, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    deletable:true,
    insertable:true,
    updateable:true,
    upsertable:true,
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> DynamicsSink
```

If you use inline dataset, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    store: 'dynamics',
    format: 'dynamicsformat',
    entity: 'Entity1',
    deletable: true,
    insertable: true,
    updateable: true,
    upsertable: true,
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> DynamicsSink
```

Lookup activity properties

To learn details about the properties, see Lookup activity.

Next steps

For a list of supported data stores the copy activity as sources and sinks, see Supported data stores.

Copy data from Dynamics AX using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from Dynamics AX source. The article builds on Copy Activity, which presents a general overview of Copy Activity.

Supported capabilities

This Dynamics AX connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that supports as sources and sinks, see Supported data stores.

Specifically, this Dynamics AX connector supports copying data from Dynamics AX using **OData protocol** with **Service Principal authentication**.

> **TIP**
> You can also use this connector to copy data from **Dynamics 365 Finance and Operations**. Refer to Dynamics 365's OData support and Authentication method.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

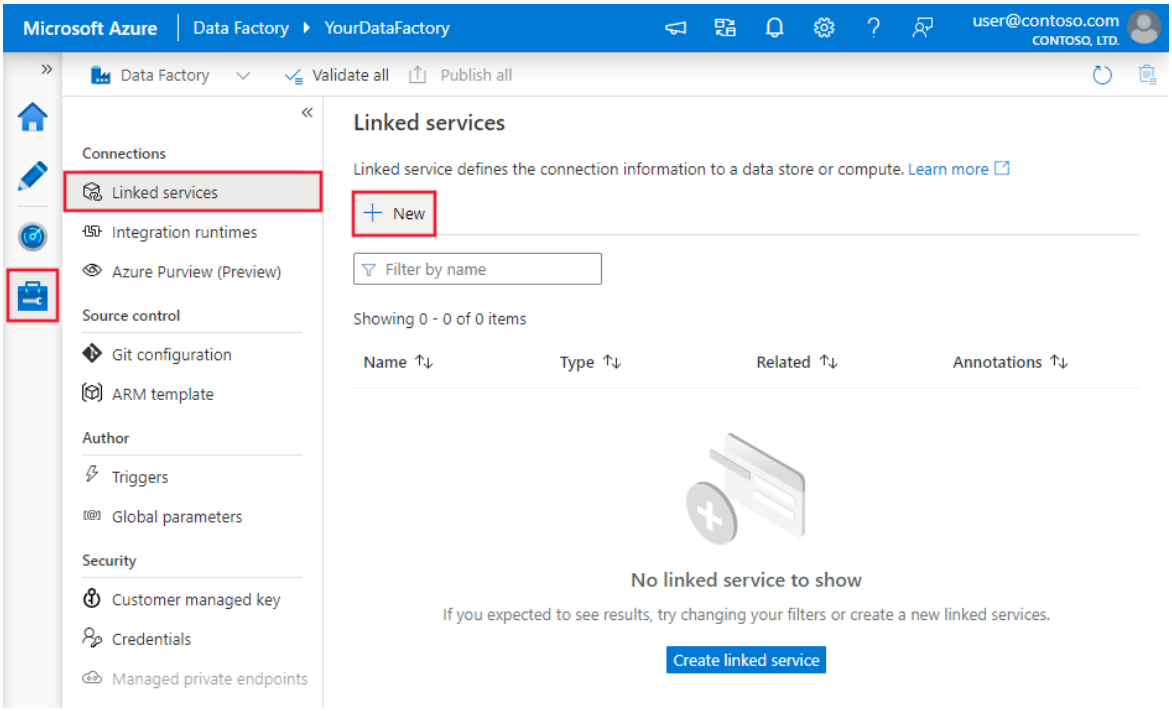
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Dynamics AX using UI

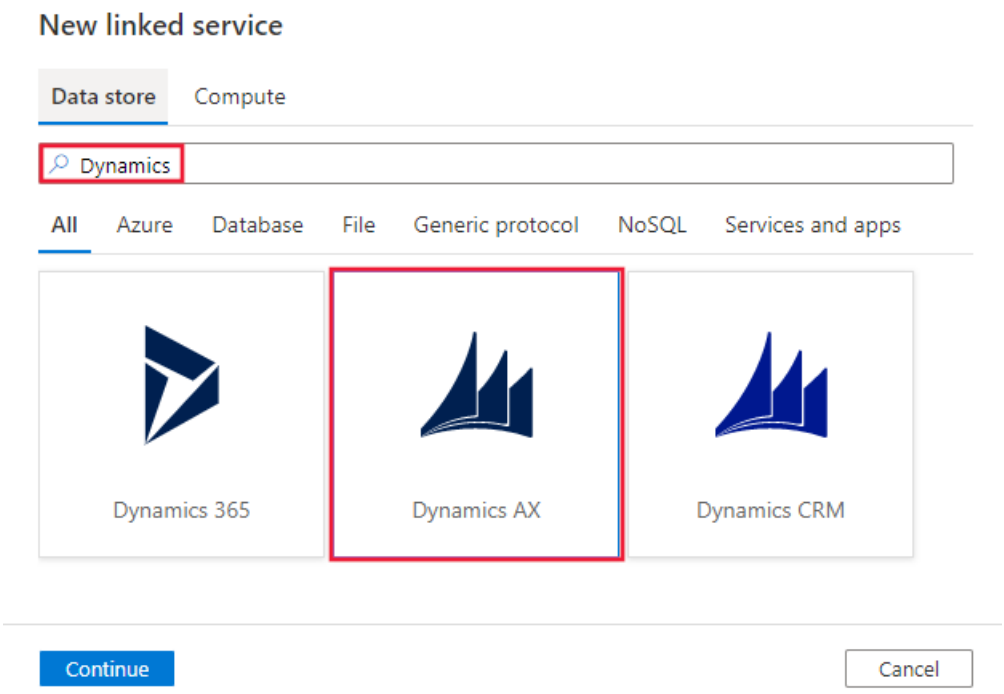
Use the following steps to create a linked service to Dynamics AX in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

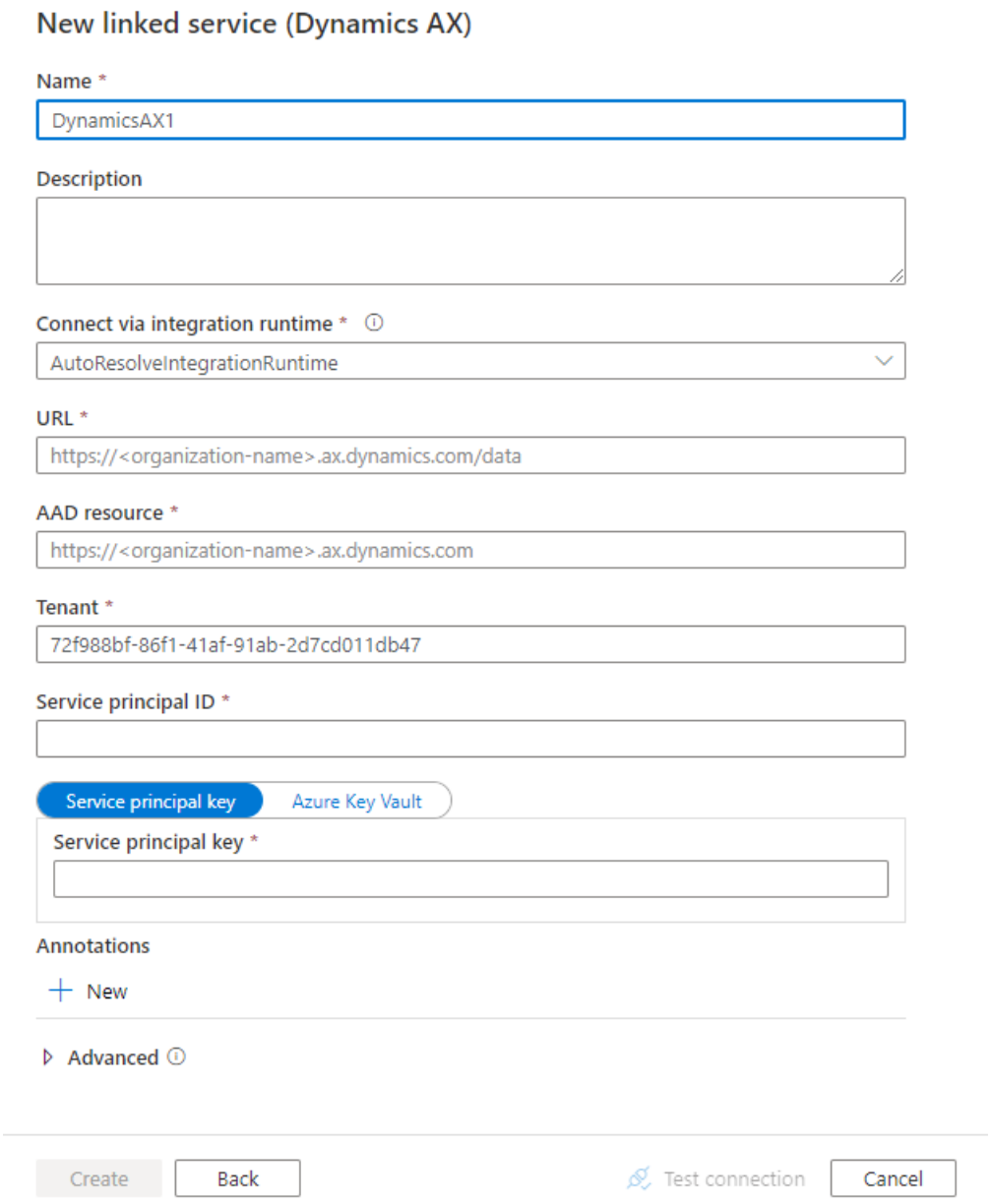
- Azure Data Factory
- Azure Synapse



2. Search for Dynamics and select the Dynamics AX connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties you can use to define Data Factory entities that are specific to Dynamics AX connector.

Prerequisites

To use service principal authentication, follow these steps:

1. Register an application entity in Azure Active Directory (Azure AD) by following Register your application with an Azure AD tenant. Make note of the following values, which you use to define the linked service:
   - Application ID
   - Application key
   - Tenant ID
2. Go to Dynamics AX, and grant this service principal proper permission to access your Dynamics AX.

Linked service properties

The following properties are supported for Dynamics AX linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **DynamicsAX**.	Yes
url	The Dynamics AX (or Dynamics 365 Finance and Operations) instance OData endpoint.	Yes
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key. Mark this field as a **SecureString** to store it securely, or reference a secret stored in Azure Key Vault.	Yes
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. Retrieve it by hovering the mouse in the top-right corner of the Azure portal.	Yes
aadResourceId	Specify the AAD resource you are requesting for authorization. For example, if your Dynamics URL is `https://sampledynamics.sandbox.operations.dynamics.com/data/`, the corresponding AAD resource is usually `https://sampledynamics.sandbox.operations.dynamics.com`.	Yes
connectVia	The Integration Runtime to use to connect to the data store. You can choose Azure Integration Runtime or a self-hosted Integration Runtime (if your data store is located in a private network). If not specified, the default Azure Integration Runtime is used.	No

Example

```
{
    "name": "DynamicsAXLinkedService",
    "properties": {
        "type": "DynamicsAX",
        "typeProperties": {
            "url": "<Dynamics AX instance OData endpoint>",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>",
            "aadResourceId": "<AAD resource, e.g. https://sampledynamics.sandbox.operations.dynamics.com>"
        }
    },
    "connectVia": {
        "referenceName": "<name of Integration Runtime>",
        "type": "IntegrationRuntimeReference"
    }
}
```

Dataset properties

This section provides a list of properties that the Dynamics AX dataset supports.

For a full list of sections and properties that are available for defining datasets, see Datasets and linked services.

To copy data from Dynamics AX, set the **type** property of the dataset to **DynamicsAXResource**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the dataset must be set to **DynamicsAXResource**.	Yes
path	The path to the Dynamics AX OData entity.	Yes

Example

```
{
    "name": "DynamicsAXResourceDataset",
    "properties": {
        "type": "DynamicsAXResource",
        "typeProperties": {
            "path": "<entity path e.g. dd04tentitySet>"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Dynamics AX linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy Activity properties

This section provides a list of properties that the Dynamics AX source supports.

For a full list of sections and properties that are available for defining activities, see Pipelines.

Dynamics AX as source

To copy data from Dynamics AX, set the **source** type in Copy Activity to **DynamicsAXSource**. The following properties are supported in the Copy Activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the Copy Activity source must be set to **DynamicsAXSource**.	Yes
query	OData query options for filtering data. Example: `"?$select=Name,Description&$top=5"`. **Note**: The connector copies data from the combined URL: `[URL specified in linked service]/[path specified in dataset][query specified in copy activity source]`. For more information, see OData URL components.	No
httpRequestTimeout	The timeout (the **TimeSpan** value) for the HTTP request to get a response. This value is the timeout to get a response, not the timeout to read response data. If not specified, the default value is **00:30:00** (30 minutes).	No

Example

```
"activities":[
    {
        "name": "CopyFromDynamicsAX",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Dynamics AX input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DynamicsAXSource",
                "query": "$top=10"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores that Copy Activity supports as sources and sinks, see Supported data stores and formats.

Copy and transform data in Dynamics 365 (Microsoft Dataverse) or Dynamics CRM using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 17 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use a copy activity in Azure Data Factory or Synapse pipelines to copy data from and to Dynamics 365 (Microsoft Dataverse) or Dynamics CRM, and use a data flow to transform data in Dynamics 365 (Microsoft Dataverse) or Dynamics CRM. To learn more, read the Azure Data Factory and the Azure Synapse Analytics introduction articles.

Supported capabilities

This connector is supported for the following activities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Mapping data flow (source/sink)	①
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that a copy activity supports as sources and sinks, see the Supported data stores table.

> **NOTE**
> Effective November 2020, Common Data Service has been renamed to Microsoft Dataverse. This article is updated to reflect the latest terminology.

This Dynamics connector supports Dynamics versions 7 through 9 for both online and on-premises. More specifically:

- Version 7 maps to Dynamics CRM 2015.
- Version 8 maps to Dynamics CRM 2016 and the early version of Dynamics 365.
- Version 9 maps to the later version of Dynamics 365.

Refer to the following table of supported authentication types and configurations for Dynamics versions and products.

DYNAMICS VERSIONS	AUTHENTICATION TYPES	LINKED SERVICE SAMPLES
Dataverse Dynamics 365 online Dynamics CRM online	Azure Active Directory (Azure AD) service principal Office 365	Dynamics online and Azure AD service-principal or Office 365 authentication
Dynamics 365 on-premises with internet-facing deployment (IFD) Dynamics CRM 2016 on-premises with IFD Dynamics CRM 2015 on-premises with IFD	IFD	Dynamics on-premises with IFD and IFD authentication

NOTE

With the deprecation of regional Discovery Service, the service has upgraded to leverage global Discovery Service while using Office 365 Authentication.

IMPORTANT

If your tenant and user is configured in Azure Active Directory for conditional access and/or Multi-Factor Authentication is required, you will not be able to use Office 365 Authentication type. For those situations, you must use an Azure Active Directory (Azure AD) service principal authentication.

For Dynamics 365 specifically, the following application types are supported:

- Dynamics 365 for Sales
- Dynamics 365 for Customer Service
- Dynamics 365 for Field Service
- Dynamics 365 for Project Service Automation
- Dynamics 365 for Marketing This connector doesn't support other application types like Finance, Operations, and Talent.

TIP

To copy data from Dynamics 365 Finance and Operations, you can use the Dynamics AX connector.

This Dynamics connector is built on top of Dynamics XRM tooling.

Prerequisites

To use this connector with Azure AD service-principal authentication, you must set up server-to-server (S2S) authentication in Dataverse or Dynamics. First register the application user (Service Principal) in Azure Active Directory. You can find out how to do this here. During application registration you will need to create that user in Dataverse or Dynamics and grant permissions. Those permissions can either be granted directly or indirectly by adding the application user to a team which has been granted permissions in Dataverse or Dynamics. You can find more information on how to set up an application user to authenticate with Dataverse here.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

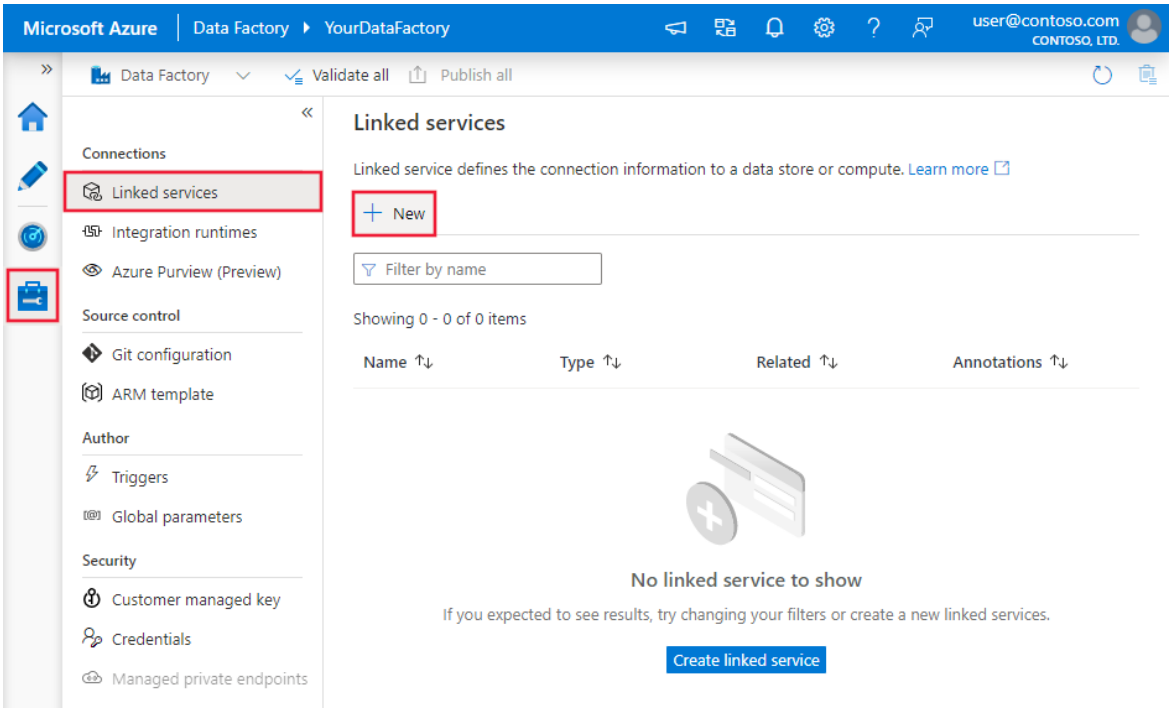
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Dynamics 365 (Microsoft Dataverse) or Dynamics CRM using UI

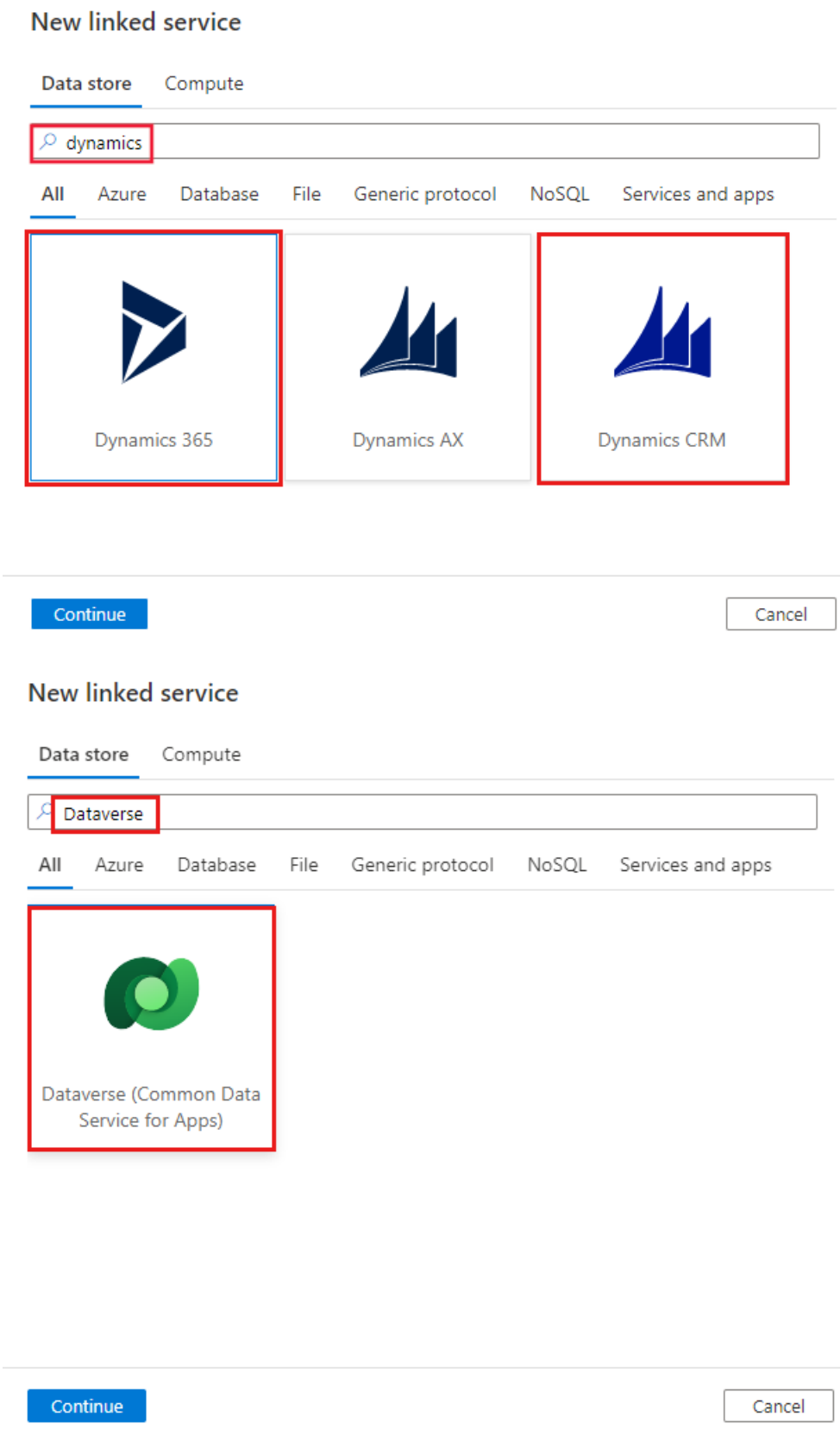
Use the following steps to create a linked service to Dynamics 365 in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

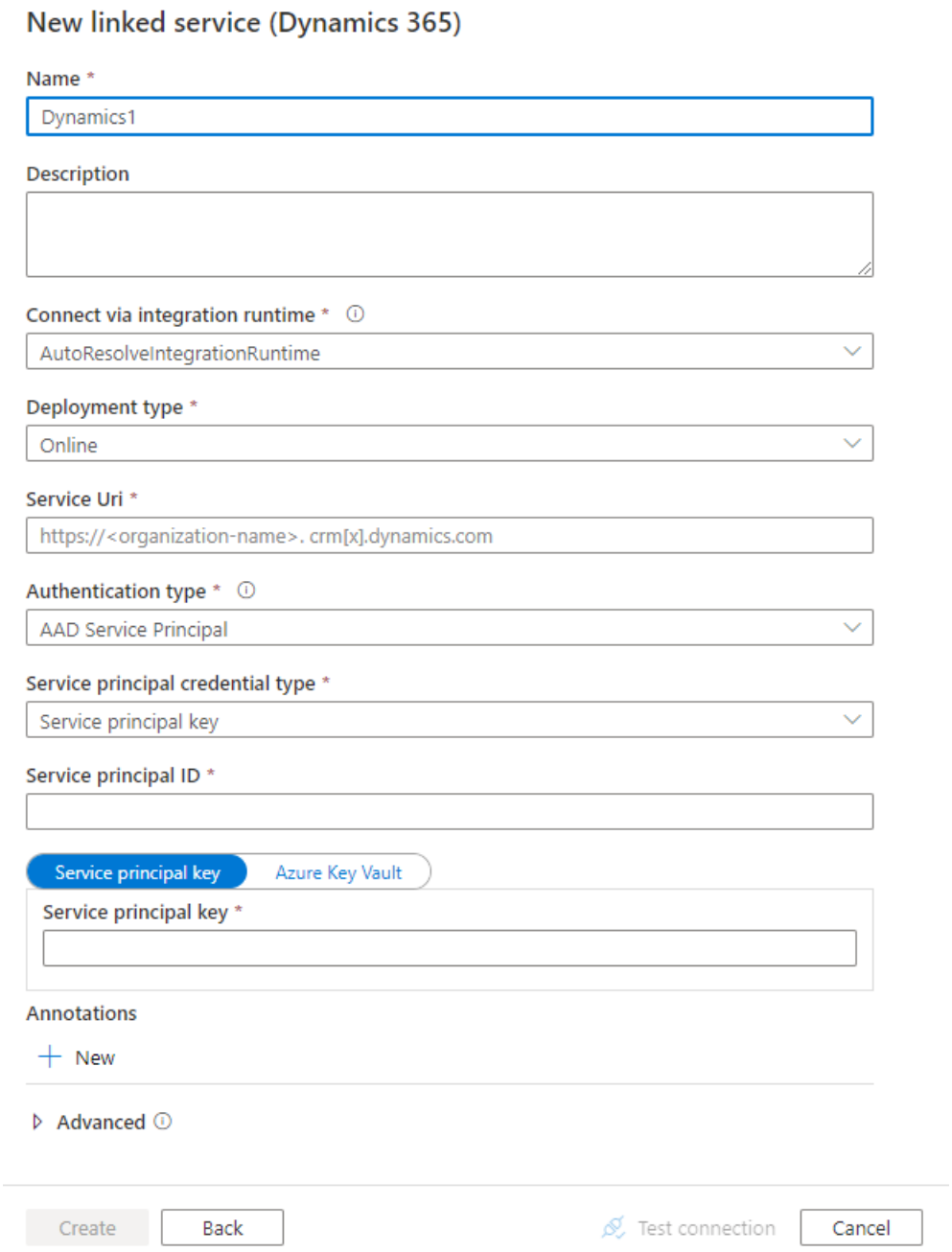
   - Azure Data Factory
   - Azure Synapse



2. Search for Dynamics or Dataverse and select the Dynamics 365 (Microsoft Dataverse) or Dynamics CRM connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to Dynamics.

Linked service properties

The following properties are supported for the Dynamics linked service.

Dynamics 365 and Dynamics CRM online

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to "Dynamics", "DynamicsCrm", or "CommonDataServiceForApps".	Yes
deploymentType	The deployment type of the Dynamics instance. The value must be "Online" for Dynamics online.	Yes

PROPERTY	DESCRIPTION	REQUIRED
serviceUri	The service URL of your Dynamics instance, the same one you access from browser. An example is "https://<organization-name>.crm[x].dynamics.com".	Yes
authenticationType	The authentication type to connect to a Dynamics server. Valid values are "AADServicePrincipal", "Office365" and "ManagedIdentity".	Yes
servicePrincipalId	The client ID of the Azure AD application.	Yes when authentication is "AADServicePrincipal"
servicePrincipalCredentialType	The credential type to use for service-principal authentication. Valid values are "ServicePrincipalKey" and "ServicePrincipalCert".	Yes when authentication is "AADServicePrincipal"
servicePrincipalCredential	The service-principal credential. When you use "ServicePrincipalKey" as the credential type, `servicePrincipalCredential` can be a string that the service encrypts upon linked service deployment. Or it can be a reference to a secret in Azure Key Vault. When you use "ServicePrincipalCert" as the credential, `servicePrincipalCredential` must be a reference to a certificate in Azure Key Vault, and ensure the certificate content type is **PKCS #12**.	Yes when authentication is "AADServicePrincipal"
username	The username to connect to Dynamics.	Yes when authentication is "Office365"
password	The password for the user account you specified as the username. Mark this field with "SecureString" to store it securely, or reference a secret stored in Azure Key Vault.	Yes when authentication is "Office365"
credentials	Specify the user-assigned managed identity as the credential object. Create one or multiple user-assigned managed identities, assign them to your data factory and create credentials for each user-assigned managed identity.	Yes when authentication is "ManagedIdentity"
connectVia	The integration runtime to be used to connect to the data store. If no value is specified, the property uses the default Azure integration runtime.	No

NOTE

The Dynamics connector formerly used the optional **organizationName** property to identify your Dynamics CRM or Dynamics 365 online instance. While that property still works, we suggest you specify the new **serviceUri** property instead to gain better performance for instance discovery.

Example: Dynamics online using Azure AD service-principal and key authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "AADServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalCredentialType": "ServicePrincipalKey",
            "servicePrincipalCredential": "<service principal key>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Dynamics online using Azure AD service-principal and certificate authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "AADServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalCredentialType": "ServicePrincipalCert",
            "servicePrincipalCredential": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<AKV reference>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<certificate name in AKV>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Dynamics online using Office 365 authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "Office365",
            "username": "test@contoso.onmicrosoft.com",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Dynamics online using user-assigned managed identity authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "typeProperties": {
            "deploymentType": "Online",
            "serviceUri": "https://<organization-name>.crm[x].dynamics.com",
            "authenticationType": "ManagedIdentity",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dynamics 365 and Dynamics CRM on-premises with IFD

Additional properties that compare to Dynamics online are **hostName** and **port**.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to "Dynamics", "DynamicsCrm", or "CommonDataServiceForApps".	Yes.
deploymentType	The deployment type of the Dynamics instance. The value must be "OnPremisesWithIfd" for Dynamics on-premises with IFD.	Yes.
hostName	The host name of the on-premises Dynamics server.	Yes.

PROPERTY	DESCRIPTION	REQUIRED
port	The port of the on-premises Dynamics server.	No. The default value is 443.
organizationName	The organization name of the Dynamics instance.	Yes.
authenticationType	The authentication type to connect to the Dynamics server. Specify "Ifd" for Dynamics on-premises with IFD.	Yes.
username	The username to connect to Dynamics.	Yes.
password	The password for the user account you specified for the username. You can mark this field with "SecureString" to store it securely. Or you can store a password in Key Vault and let the copy activity pull from there when it does data copy. Learn more from Store credentials in Key Vault.	Yes.
connectVia	The integration runtime to be used to connect to the data store. If no value is specified, the property uses the default Azure integration runtime.	No

Example: Dynamics on-premises with IFD using IFD authentication

```
{
    "name": "DynamicsLinkedService",
    "properties": {
        "type": "Dynamics",
        "description": "Dynamics on-premises with IFD linked service using IFD authentication",
        "typeProperties": {
            "deploymentType": "OnPremisesWithIFD",
            "hostName": "contosodynamicsserver.contoso.com",
            "port": 443,
            "organizationName": "admsDynamicsTest",
            "authenticationType": "Ifd",
            "username": "test@contoso.onmicrosoft.com",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by Dynamics dataset.

To copy data from and to Dynamics, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to "DynamicsEntity", "DynamicsCrmEntity", or "CommonDataServiceForAppsEntity".	Yes
entityName	The logical name of the entity to retrieve.	No for source if the activity source is specified as "query" and yes for sink

Example

```
{
    "name": "DynamicsDataset",
    "properties": {
        "type": "DynamicsEntity",
        "schema": [],
        "typeProperties": {
            "entityName": "account"
        },
        "linkedServiceName": {
            "referenceName": "<Dynamics linked service name>",
            "type": "linkedservicereference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Dynamics source and sink types.

Dynamics as a source type

To copy data from Dynamics, the copy activity **source** section supports the following properties:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to "DynamicsSource", "DynamicsCrmSource", or "CommonDataServiceForAppsSource".	Yes
query	FetchXML is a proprietary query language that is used in Dynamics online and on-premises. See the following example. To learn more, see Build queries with FetchXML.	No if `entityName` in the dataset is specified

> **NOTE**
> The PK column will always be copied out even if the column projection you configure in the FetchXML query doesn't contain it.

IMPORTANT

- When you copy data from Dynamics, explicit column mapping from Dynamics to sink is optional. But we highly recommend the mapping to ensure a deterministic copy result.
- When the service imports a schema in the authoring UI, it infers the schema. It does so by sampling the top rows from the Dynamics query result to initialize the source column list. In that case, columns with no values in the top rows are omitted. The same behavior also applies to data preview and copy executions if there is no explicit mapping. You can review and add more columns into the mapping, which are honored during copy runtime.

Example

```
"activities":[
    {
        "name": "CopyFromDynamics",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Dynamics input dataset>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DynamicsSource",
                "query": "<FetchXML Query>"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Sample FetchXML query

```
<fetch>
  <entity name="account">
    <attribute name="accountid" />
    <attribute name="name" />
    <attribute name="marketingonly" />
    <attribute name="modifiedon" />
    <order attribute="modifiedon" descending="false" />
    <filter type="and">
      <condition attribute ="modifiedon" operator="between">
        <value>2017-03-10 18:40:00z</value>
        <value>2017-03-12 20:40:00z</value>
      </condition>
    </filter>
  </entity>
</fetch>
```

Dynamics as a sink type

To copy data to Dynamics, the copy activity **sink** section supports the following properties:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to "DynamicsSink", "DynamicsCrmSink", or "CommonDataServiceForAppsSink".	Yes.
writeBehavior	The write behavior of the operation. The value must be "Upsert".	Yes
alternateKeyName	The alternate key name defined on your entity to do an upsert.	No.
writeBatchSize	The row count of data written to Dynamics in each batch.	No. The default value is 10.
ignoreNullValues	Whether to ignore null values from input data other than key fields during a write operation. Valid values are **TRUE** and **FALSE**: • **TRUE**: Leave the data in the destination object unchanged when you do an upsert or update operation. Insert a defined default value when you do an insert operation. • **FALSE**: Update the data in the destination object to a null value when you do an upsert or update operation. Insert a null value when you do an insert operation.	No. The default value is **FALSE**.
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

NOTE

The default value for both the sink **writeBatchSize** and the copy activity **parallelCopies** for the Dynamics sink is 10. Therefore, 100 records are concurrently submitted by default to Dynamics.

For Dynamics 365 online, there's a limit of two concurrent batch calls per organization. If that limit is exceeded, a "Server Busy" exception is thrown before the first request is ever run. Keep **writeBatchSize** at 10 or less to avoid such throttling of concurrent calls.

The optimal combination of **writeBatchSize** and **parallelCopies** depends on the schema of your entity. Schema elements include the number of columns, row size, and number of plug-ins, workflows, or workflow activities hooked up to those calls. The default setting of **writeBatchSize** (10) × **parallelCopies** (10) is the recommendation according to the Dynamics service. This value works for most Dynamics entities, although it might not give the best performance. You can tune the performance by adjusting the combination in your copy activity settings.

Example

```
"activities":[
    {
        "name": "CopyToDynamics",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Dynamics output dataset>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "DynamicsSink",
                "writeBehavior": "Upsert",
                "writeBatchSize": 10,
                "ignoreNullValues": true
            }
        }
    }
]
```

Retrieving data from views

To retrieve data from Dynamics views, you need to get the saved query of the view, and use the query to get the data.

There are two entities which store different types of view: "saved query" stores system view and "user query" stores user view. To get the information of the views, refer to the following FetchXML query and replace the "TARGETENTITY" with `savedquery` or `userquery`. Each entity type has more available attributes that you can add to the query based on your need. Learn more about savedquery entity and userquery entity.

```
<fetch top="5000" >
  <entity name="<TARGETENTITY>">
    <attribute name="name" />
    <attribute name="fetchxml" />
    <attribute name="returnedtypecode" />
    <attribute name="querytype" />
  </entity>
</fetch>
```

You can also add filters to filter the views. For example, add the following filter to get a view named "My Active Accounts" in account entity.

```
<filter type="and" >
    <condition attribute="returnedtypecode" operator="eq" value="1" />
    <condition attribute="name" operator="eq" value="My Active Accounts" />
</filter>
```

Data type mapping for Dynamics

When you copy data from Dynamics, the following table shows mappings from Dynamics data types to interim data types within the service. To learn how a copy activity maps to a source schema and a data type maps to a sink, see Schema and data type mappings.

Configure the corresponding interim data type in a dataset structure that is based on your source Dynamics data type by using the following mapping table:

DYNAMICS DATA TYPE	SERVICE INTERIM DATA TYPE	SUPPORTED AS SOURCE	SUPPORTED AS SINK
AttributeTypeCode.BigInt	Long	✓	✓
AttributeTypeCode.Boolean	Boolean	✓	✓
AttributeType.Customer	GUID	✓	✓ (See guidance)
AttributeType.DateTime	Datetime	✓	✓
AttributeType.Decimal	Decimal	✓	✓
AttributeType.Double	Double	✓	✓
AttributeType.EntityName	String	✓	✓
AttributeType.Integer	Int32	✓	✓
AttributeType.Lookup	GUID	✓	✓ (See guidance)
AttributeType.ManagedProperty	Boolean	✓	
AttributeType.Memo	String	✓	✓
AttributeType.Money	Decimal	✓	✓
AttributeType.Owner	GUID	✓	✓ (See guidance)
AttributeType.Picklist	Int32	✓	✓
AttributeType.Uniqueidentifier	GUID	✓	✓
AttributeType.String	String	✓	✓
AttributeType.State	Int32	✓	✓
AttributeType.Status	Int32	✓	✓

NOTE

The Dynamics data types **AttributeType.CalendarRules**, **AttributeType.MultiSelectPicklist**, and **AttributeType.PartyList** aren't supported.

Writing data to a lookup field

To write data into a lookup field with multiple targets like Customer and Owner, follow this guidance and example:

1. Make your source contains both the field value and the corresponding target entity name.
   - If all records map to the same target entity, ensure one of the following conditions:
     - Your source data has a column that stores the target entity name.
     - You've added an additional column in the copy activity source to define the target entity.
   - If different records map to different target entities, make sure your source data has a column that stores the corresponding target entity name.
2. Map both the value and entity-reference columns from source to sink. The entity-reference column must be mapped to a virtual column with the special naming pattern `{lookup_field_name}@EntityReference`. The column doesn't actually exist in Dynamics. It's used to indicate this column is the metadata column of the given multitarget lookup field.

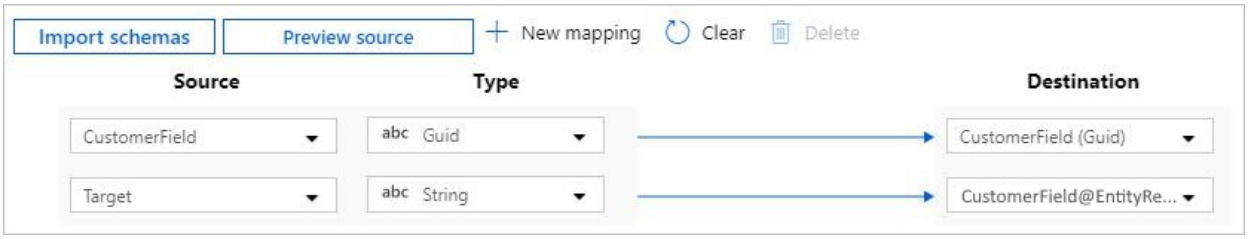
For example, assume the source has these two columns:

- **CustomerField** column of type **GUID**, which is the primary key value of the target entity in Dynamics.
- **Target** column of type **String**, which is the logical name of the target entity.

Also assume you want to copy such data to the sink Dynamics entity field **CustomerField** of type **Customer**.

In copy-activity column mapping, map the two columns as follows:

- **CustomerField** to **CustomerField**. This mapping is the normal field mapping.
- **Target** to **CustomerField@EntityReference**. The sink column is a virtual column representing the entity reference. Input such field names in a mapping, as they won't show up by importing schemas.



If all of your source records map to the same target entity and your source data doesn't contain the target entity name, here is a shortcut: in the copy activity source, add an additional column. Name the new column by using the pattern `{lookup_field_name}@EntityReference`, set the value to the target entity name, then proceed with column mapping as usual. If your source and sink column names are identical, you can also skip explicit column mapping because copy activity by default maps columns by name.

Mapping data flow properties

When transforming data in mapping data flow, you can read from and write to tables in Dynamics. For more information, see the source transformation and sink transformation in mapping data flows. You can choose to use a Dynamics dataset or an inline dataset as source and sink type.

Source transformation

The below table lists the properties supported by Dynamics. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Entity name	The logical name of the entity to retrieve.	Yes when use inline dataset	-	*(for inline dataset only)* entity
Query	FetchXML is a proprietary query language that is used in Dynamics online and on-premises. See the following example. To learn more, see Build queries with FetchXML.	No	String	query

> **NOTE**
> If you select **Query** as input type, the column type from tables can not be retrieved. It will be treated as string by default.

Dynamics source script example

When you use Dynamics dataset as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 query: '<fetch mapping='logical' count='3 paging-cookie=''><entity name='new_dataflow_crud_test'><attribute
name='new_name'/><attribute name='new_releasedate'/></entity></fetch>') ~> DynamicsSource
```

If you use inline dataset, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 store: 'dynamics',
 format: 'dynamicsformat',
 entity: 'Entity1',
 query: '<fetch mapping='logical' count='3 paging-cookie=''><entity name='new_dataflow_crud_test'><attribute
name='new_name'/><attribute name='new_releasedate'/></entity></fetch>') ~> DynamicsSource
```

Sink transformation

The below table lists the properties supported by Dynamics sink. You can edit these properties in the **Sink options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Alternate key name	The alternate key name defined on your entity to do an update, upsert or delete.	No	-	alternateKeyName

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Update method	Specify what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an Alter row transformation is required to tag rows for those actions.	Yes	`true` or `false`	insertable updateable upsertable deletable
Entity name	The logical name of the entity to write.	Yes when use inline dataset	-	*(for inline dataset only)* entity

Dynamics sink script example

When you use Dynamics dataset as sink type, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    deletable:true,
    insertable:true,
    updateable:true,
    upsertable:true,
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> DynamicsSink
```

If you use inline dataset, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    store: 'dynamics',
    format: 'dynamicsformat',
    entity: 'Entity1',
    deletable: true,
    insertable: true,
    updateable: true,
    upsertable: true,
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> DynamicsSink
```

Lookup activity properties

To learn details about the properties, see Lookup activity.

Next steps

For a list of supported data stores the copy activity as sources and sinks, see Supported data stores.

Excel file format in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Follow this article when you want to **parse the Excel files**. The service supports both ".xls" and ".xlsx".

Excel format is supported for the following connectors: Amazon S3, Amazon S3 Compatible Storage, Azure Blob, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure Files, File System, FTP, Google Cloud Storage, HDFS, HTTP, Oracle Cloud Storage and SFTP. It is supported as source but not sink.

> **NOTE**
> ".xls" format is not supported while using HTTP.

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the Excel dataset.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **Excel**.	Yes
location	Location settings of the file(s). Each file-based connector has its own location type and supported properties under `location`.	Yes
sheetName	The Excel worksheet name to read data.	Specify `sheetName` or `sheetIndex`
sheetIndex	The Excel worksheet index to read data, starting from 0.	Specify `sheetName` or `sheetIndex`
range	The cell range in the given worksheet to locate the selective data, e.g.: - Not specified: reads the whole worksheet as a table from the first non-empty row and column - `A3`: reads a table starting from the given cell, dynamically detects all the rows below and all the columns to the right - `A3:H5`: reads this fixed range as a table - `A3:A3`: reads this single cell	No

PROPERTY	DESCRIPTION	REQUIRED
firstRowAsHeader	Specifies whether to treat the first row in the given worksheet/range as a header line with names of columns. Allowed values are **true** and **false** (default).	No
nullValue	Specifies the string representation of null value. The default value is **empty string**.	No
compression	Group of properties to configure file compression. Configure this section when you want to do compression/decompression during activity execution.	No
type (*under* `compression`)	The compression codec used to read/write JSON files. Allowed values are **bzip2**, **gzip**, **deflate**, **ZipDeflate**, **TarGzip**, **Tar**, **snappy**, or **lz4**. Default is not compressed. **Note** currently Copy activity doesn't support "snappy" & "lz4", and mapping data flow doesn't support "ZipDeflate", "TarGzip" and "Tar". **Note** when using copy activity to decompress **ZipDeflate** file(s) and write to file-based sink data store, files are extracted to the folder: `<path specified in dataset>/<folder named as source zip file>/` .	No.
level (*under* `compression`)	The compression ratio. Allowed values are **Optimal** or **Fastest**. - **Fastest:** The compression operation should complete as quickly as possible, even if the resulting file is not optimally compressed. - **Optimal**: The compression operation should be optimally compressed, even if the operation takes a longer time to complete. For more information, see Compression Level topic.	No

Below is an example of Excel dataset on Azure Blob Storage:

```
{
    "name": "ExcelDataset",
    "properties": {
        "type": "Excel",
        "linkedServiceName": {
            "referenceName": "<Azure Blob Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "container": "containername",
                "folderPath": "folder/subfolder",
            },
            "sheetName": "MyWorksheet",
            "range": "A3:H5",
            "firstRowAsHeader": true
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the Excel source.

Excel as source

The following properties are supported in the copy activity ***source*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **ExcelSource**.	Yes
storeSettings	A group of properties on how to read data from a data store. Each file-based connector has its own supported read settings under `storeSettings`.	No

```
"activities": [
    {
        "name": "CopyFromExcel",
        "type": "Copy",
        "typeProperties": {
            "source": {
                "type": "ExcelSource",
                "storeSettings": {
                    "type": "AzureBlobStorageReadSettings",
                    "recursive": true
                }
            },
            ...
        }
        ...
    }
]
```

Mapping data flow properties

In mapping data flows, you can read Excel format in the following data stores: Azure Blob Storage, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Amazon S3 and SFTP. You can point to Excel files either using Excel dataset or using an inline dataset.

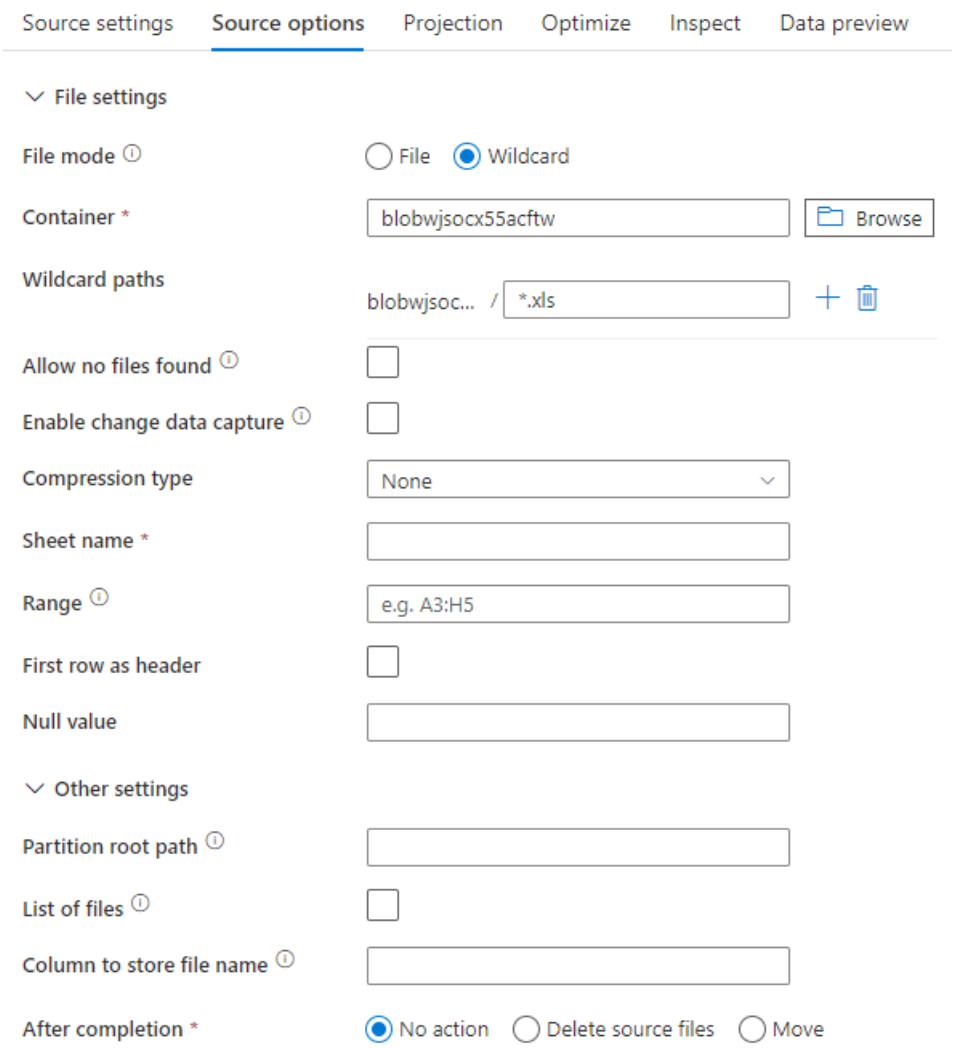
Source properties

The below table lists the properties supported by an Excel source. You can edit these properties in the **Source options** tab. When using inline dataset, you will see additional file settings, which are the same as the properties described in dataset properties section.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Wild card paths	All files matching the wildcard path will be processed. Overrides the folder and file path set in the dataset.	no	String[]	wildcardPaths
Partition root path	For file data that is partitioned, you can enter a partition root path in order to read partitioned folders as columns	no	String	partitionRootPath
List of files	Whether your source is pointing to a text file that lists files to process	no	`true` or `false`	fileList
Column to store file name	Create a new column with the source file name and path	no	String	rowUrlColumn
After completion	Delete or move the files after processing. File path starts from the container root	no	Delete: `true` or `false` Move: `['<from>', '<to>']`	purgeFiles moveFiles
Filter by last modified	Choose to filter files based upon when they were last altered	no	Timestamp	modifiedAfter modifiedBefore
Allow no files found	If true, an error is not thrown if no files are found	no	`true` or `false`	ignoreNoFilesFound

Source example

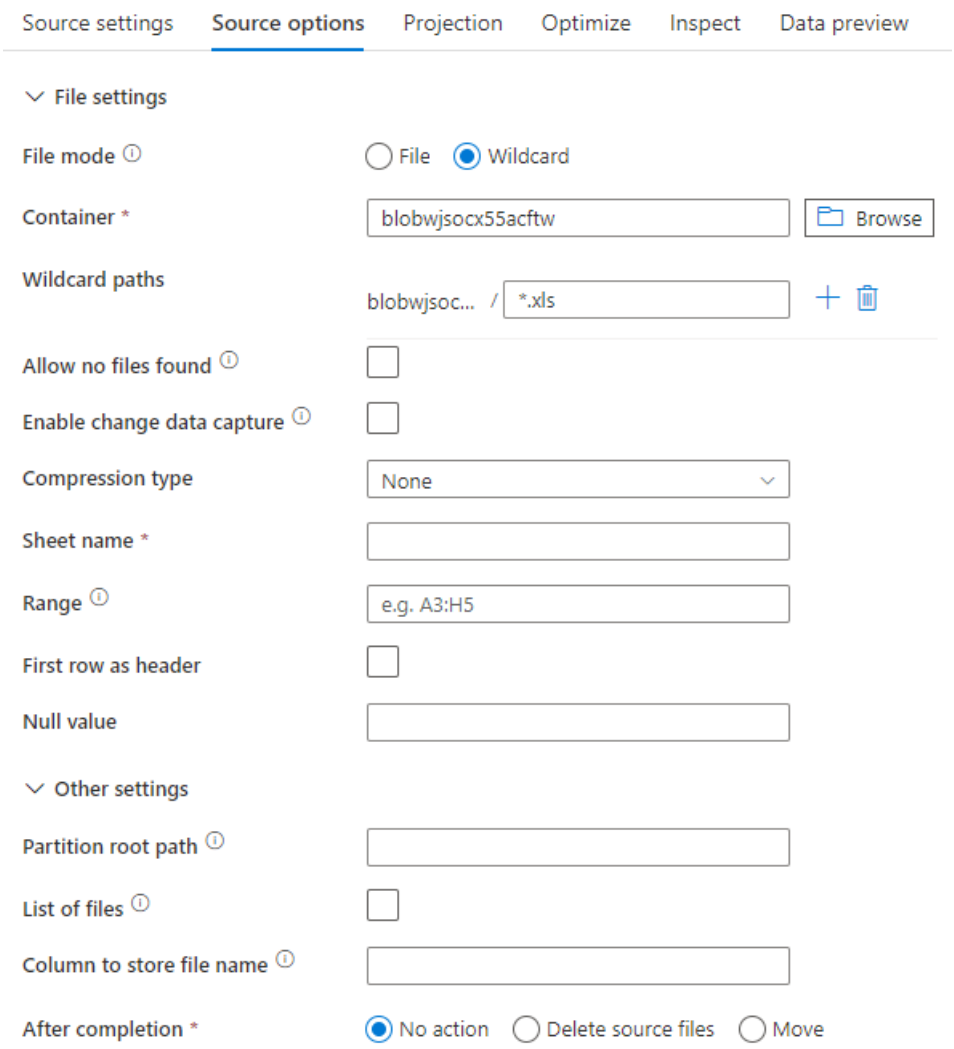
The below image is an example of an Excel source configuration in mapping data flows using dataset mode.



The associated data flow script is:

```
source(allowSchemaDrift: true,
    validateSchema: false,
    wildcardPaths:['*.xls']) ~> ExcelSource
```

If you use inline dataset, you see the following source options in mapping data flow.



The associated data flow script is:

```
source(allowSchemaDrift: true,
    validateSchema: false,
    format: 'excel',
    fileSystem: 'container',
    folderPath: 'path',
    fileName: 'sample.xls',
    sheetName: 'worksheet',
    firstRowAsHeader: true) ~> ExcelSourceInlineDataset
```

Handling very large Excel files

The Excel connector does not support streaming read for the Copy activity and must load the entire file into memory before data can be read. To import schema, preview data, or refresh an Excel dataset, the data must be returned before the http request timeout (100s). For large Excel files, these operations may not finish within that timeframe, causing a timeout error. If you want to move large Excel files (>100MB) into another data store, you can use one of following options to work around this limitation:

- Use the self-hosted integration runtime (SHIR), then use the Copy activity to move the large Excel file into another data store with the SHIR.
- Split the large Excel file into several smaller ones, then use the Copy activity to move the folder containing the files.
- Use a dataflow activity to move the large Excel file into another data store. Dataflow supports streaming read for Excel and can move/transfer large files quickly.
- Manually convert the large Excel file to CSV format, then use a Copy activity to move the file.

Next steps

- Copy activity overview
- Lookup activity
- GetMetadata activity

Copy data to or from a file system by using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 18 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to copy data to and from file system. To learn more read the introductory article for Azure Data Factory or Azure Synapse Analytics.

Supported capabilities

This file system connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Lookup activity	① ②
GetMetadata activity	① ②
Delete activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

Specifically, this file system connector supports:

- Copying files from/to local machine or network file share. To use a Linux file share, install Samba on your Linux server.
- Copying files using **Windows** authentication.
- Copying files as-is or parsing/generating files with the supported file formats and compression codecs.

> **NOTE**
> Mapped network drive is not supported when loading data from a network file share. Use the actual path instead e.g. `\\server\share`.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data

access strategies.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

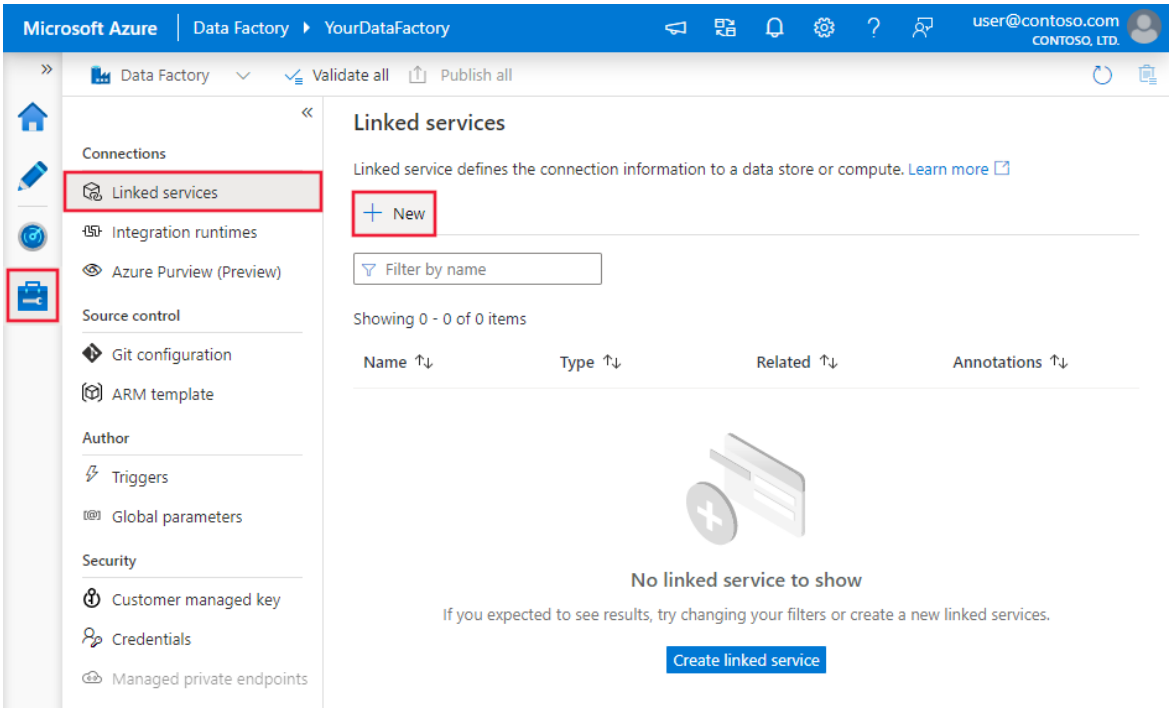
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a file system linked service using UI

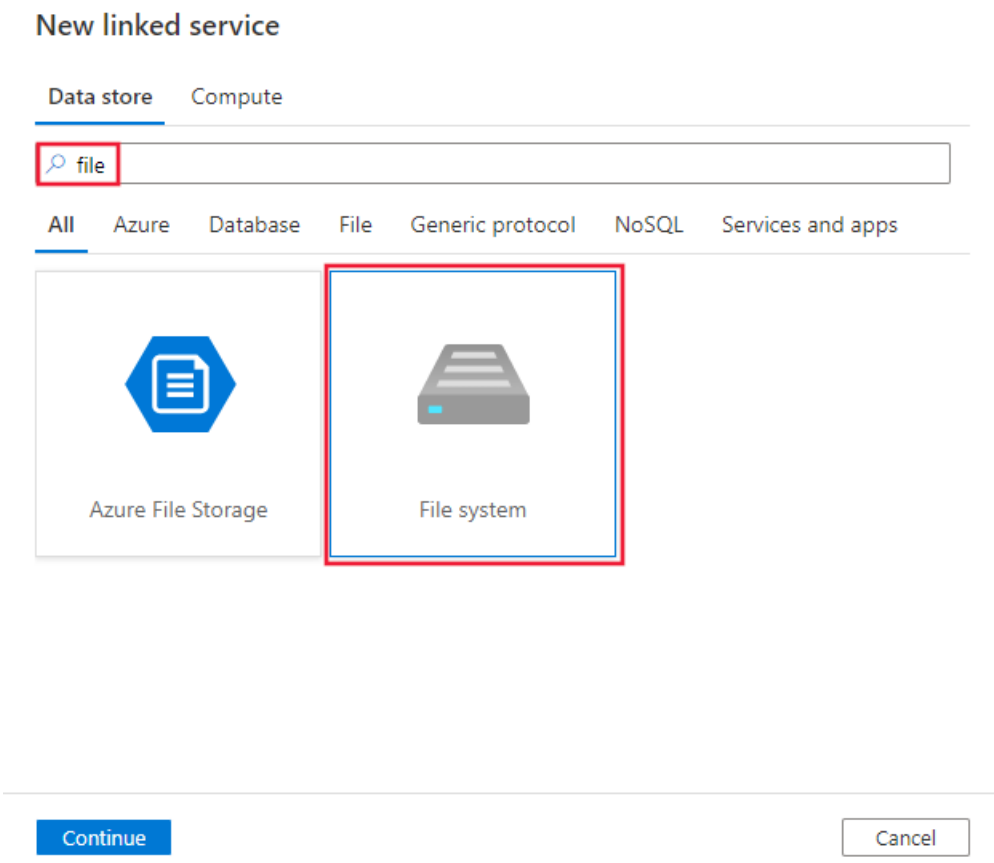
Use the following steps to create a file system linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

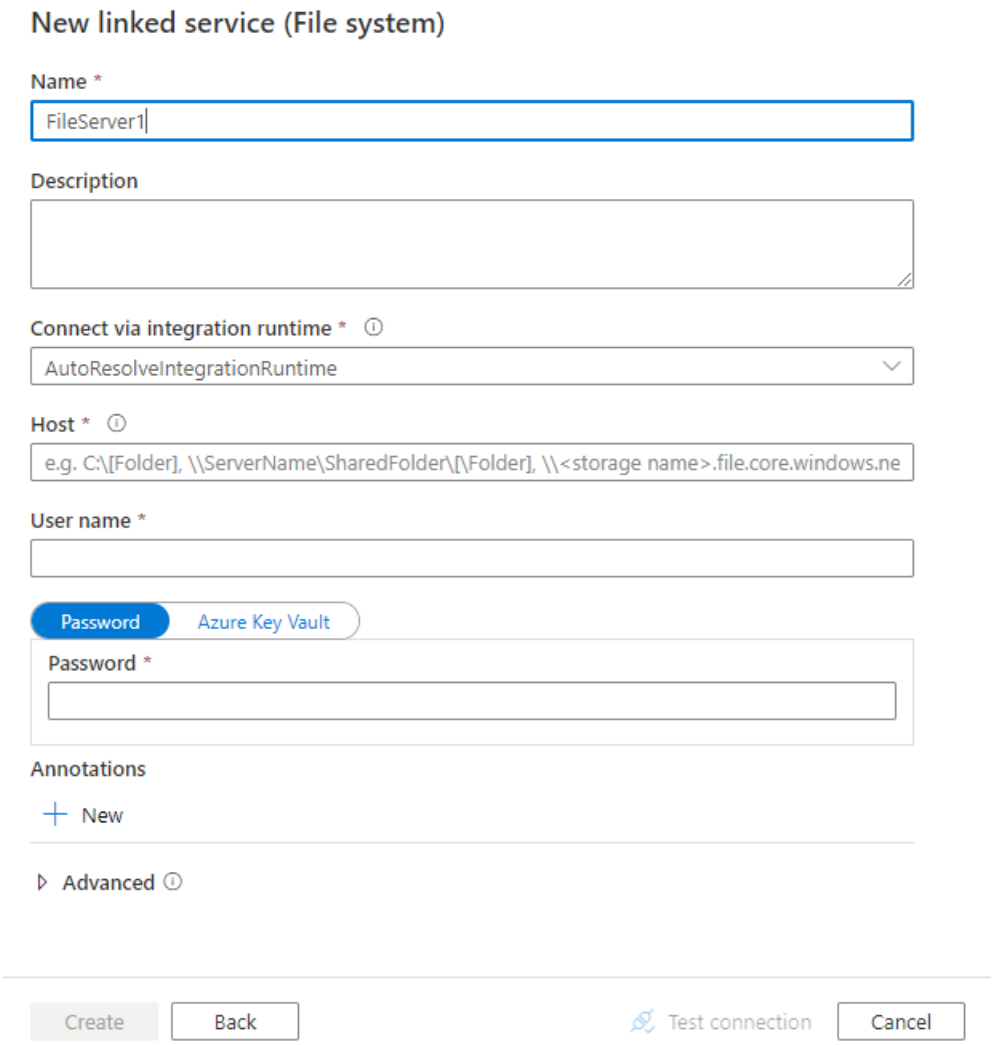
   - Azure Data Factory
   - Azure Synapse



2. Search for file and select the File System connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory and Synapse pipeline entities specific to file system.

Linked service properties

The following properties are supported for file system linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **FileServer**.	Yes
host	Specifies the root path of the folder that you want to copy. Use the escape character "" for special characters in the string. See Sample linked service and dataset definitions for examples.	Yes
userId	Specify the ID of the user who has access to the server.	Yes
password	Specify the password for the user (userId). Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Sample linked service and dataset definitions

SCENARIO	"HOST" IN LINKED SERVICE DEFINITION	"FOLDERPATH" IN DATASET DEFINITION
Local folder on Integration Runtime machine: Examples: D:* or D:\folder\subfolder*	In JSON: `D:\\` On UI: `D:\`	In JSON: `.\\` or `folder\\subfolder` On UI: `.\` or `folder\subfolder`
Remote shared folder: Examples: \\myserver\share* or \\myserver\share\folder\subfolder*	In JSON: `\\\\myserver\\share` On UI: `\\myserver\share`	In JSON: `.\\` or `folder\\subfolder` On UI: `.\` or `folder\subfolder`

> **NOTE**
>
> When authoring via UI, you don't need to input double backslash (`\\`) to escape like you do via JSON, specify single backslash.

Example:

```
{
    "name": "FileLinkedService",
    "properties": {
        "type": "FileServer",
        "typeProperties": {
            "host": "<host>",
            "userId": "<domain>\\<user>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for file system under `location` settings in format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `location` in dataset must be set to **FileServerLocation**.	Yes
folderPath	The path to folder. If you want to use wildcard to filter folder, skip this setting and specify in activity source settings. Note that you will need to setup the file share location in your Windows or Linux environment to expose the folder for sharing.	No
fileName	The file name under the given folderPath. If you want to use wildcard to filter files, skip this setting and specify in activity source settings.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<File system linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "FileServerLocation",
                "folderPath": "root/folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by file system source and sink.

File system as source

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for file system under `storeSettings` settings in format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `storeSettings` must be set to **FileServerReadSettings**.	Yes
Locate the files to copy:		
OPTION 1: static path	Copy from the given folder/file path specified in the dataset. If you want to copy all files from a folder, additionally specify `wildcardFileName` as `*`.	

PROPERTY	DESCRIPTION	REQUIRED
OPTION 2: server side filter - fileFilter	File server side native filter, which provides better performance than OPTION 3 wildcard filter. Use `*` to match zero or more characters and `?` to match zero or single character. Learn more about the syntax and notes from the **Remarks** under this section.	No
OPTION 3: client side filter - wildcardFolderPath	The folder path with wildcard characters to filter source folders. Such filter happens within the service, which enumerate the folders/files under the given path then apply the wildcard filter. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual folder name has wildcard or this escape char inside. See more examples in Folder and file filter examples.	No
OPTION 3: client side filter - wildcardFileName	The file name with wildcard characters under the given folderPath/wildcardFolderPath to filter source files. Such filter happens within the service, which enumerates the files under the given path then apply the wildcard filter. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual file name has wildcard or this escape char inside. See more examples in Folder and file filter examples.	Yes
OPTION 3: a list of files - fileListPath	Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset. When using this option, do not specify file name in dataset. See more examples in File list examples.	No
Additional settings:		

PROPERTY	DESCRIPTION	REQUIRED
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when recursive is set to true and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**. This property doesn't apply when you configure `fileListPath`.	No
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No
modifiedDatetimeStart	Files filter based on the attribute: Last Modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z". The properties can be NULL, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal with the datetime value will be selected. When `modifiedDatetimeEnd` has datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value will be selected. This property doesn't apply when you configure `fileListPath`.	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromFileSystem",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "FileServerReadSettings",
                    "recursive": true,
                    "wildcardFolderPath": "myfolder*A",
                    "wildcardFileName": "*.csv"
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

File system as sink

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- JSON format
- ORC format
- Parquet format

The following properties are supported for file system under `storeSettings` settings in format-based copy sink:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `storeSettings` must be set to **FileServerWriteSettings**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
copyBehavior	Defines the copy behavior when the source is files from a file-based data store. Allowed values are: - **PreserveHierarchy (default)**: Preserves the file hierarchy in the target folder. The relative path of source file to source folder is identical to the relative path of target file to target folder. - **FlattenHierarchy**: All files from the source folder are in the first level of the target folder. The target files have autogenerated names. - **MergeFiles**: Merges all files from the source folder to one file. If the file name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToFileSystem",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Parquet output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "ParquetSink",
                "storeSettings":{
                    "type": "FileServerWriteSettings",
                    "copyBehavior": "PreserveHierarchy"
                }
            }
        }
    }
]
```

Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`Folder*`	(empty, use default)	false	FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	(empty, use default)	true	FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv
`Folder*`	`*.csv`	false	FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	`*.csv`	true	FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This section describes the resulting behavior of using file list path in copy activity source.

Assuming you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	PIPELINE CONFIGURATION
root FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In dataset:** - Folder path: `root/FolderA` **In copy activity source:** - File list path: `root/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line with the relative path to the path configured in the dataset.

recursive and copyBehavior examples

This section describes the resulting behavior of the Copy operation for different combinations of recursive and copyBehavior values.

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET
true	preserveHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder Folder1 is created with the same structure as the source: Folder1 File1 File2 Subfolder1 File3 File4 File5.
true	flattenHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the following structure: Folder1 autogenerated name for File1 autogenerated name for File2 autogenerated name for File3 autogenerated name for File4 autogenerated name for File5
true	mergeFiles	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the following structure: Folder1 File1 + File2 + File3 + File4 + File 5 contents are merged into one file with autogenerated file name

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET

false	preserveHierarchy	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target folder Folder1 is created with the following structure
			
			Folder1
			  File1
			  File2
			
			Subfolder1 with File3, File4, and File5 are not picked up.

false	flattenHierarchy	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target folder Folder1 is created with the following structure
			
			Folder1
			  autogenerated name for File1
			  autogenerated name for File2
			
			Subfolder1 with File3, File4, and File5 are not picked up.

false	mergeFiles	Folder1
		  File1
		  File2
		  Subfolder1
		  File3
		  File4
		  File5
			The target folder Folder1 is created with the following structure
			
			Folder1
			  File1 + File2 contents are merged into one file with autogenerated file name. autogenerated name for File1
			
			Subfolder1 with File3, File4, and File5 are not picked up.

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity

Delete activity properties

To learn details about the properties, check Delete activity

Legacy models

> **NOTE**
>
> The following models are still supported as-is for backward compatibility. You are suggested to use the new model mentioned in above sections going forward, and the authoring UI has switched to generating the new model.

Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **FileShare**	Yes
folderPath	Path to the folder. Wildcard filter is supported, allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual folder name has wildcard or this escape char inside. Examples: rootfolder/subfolder/, see more examples in Sample linked service and dataset definitions and Folder and file filter examples.	No
fileName	**Name or wildcard filter** for the file(s) under the specified "folderPath". If you don't specify a value for this property, the dataset points to all files in the folder. For filter, allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). - Example 1: `"fileName": "*.csv"` - Example 2: `"fileName": "???20180427.txt"` Use `^` to escape if your actual file name has wildcard or this escape char inside. When fileName isn't specified for an output dataset and **preserveHierarchy** isn't specified in the activity sink, the copy activity automatically generates the file name with the following pattern: "*Data.[activity run ID GUID].[GUID if FlattenHierarchy].[format if configured].[compression if configured]*", for example "Data.0a405f8a-93ff-4c6f-b3be-f69616f1df7a.txt.gz"; if you copy from tabular source using table name instead of query, the name pattern is "*[table name].[format].[compression if configured]*", for example "MyTable.csv".	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	Files filter based on the attribute: Last Modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z". Be aware the overall performance of data movement will be impacted by enabling this setting when you want to do file filter from huge amounts of files. The properties can be NULL, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal with the datetime value will be selected. When `modifiedDatetimeEnd` has datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value will be selected.	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeEnd	Files filter based on the attribute: Last Modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z". Be aware the overall performance of data movement will be impacted by enabling this setting when you want to do file filter from huge amounts of files. The properties can be NULL, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has datetime value but `modifiedDatetimeEnd` is NULL, it means the files whose last modified attribute is greater than or equal with the datetime value will be selected. When `modifiedDatetimeEnd` has datetime value but `modifiedDatetimeStart` is NULL, it means the files whose last modified attribute is less than the datetime value will be selected.	No
format	If you want to **copy files as-is** between file-based stores (binary copy), skip the format section in both input and output dataset definitions. If you want to parse or generate files with a specific format, the following file format types are supported: **TextFormat**, **JsonFormat**, **AvroFormat**, **OrcFormat**, **ParquetFormat**. Set the **type** property under format to one of these values. For more information, see Text Format, Json Format, Avro Format, Orc Format, and Parquet Format sections.	No (only for binary copy scenario)
compression	Specify the type and level of compression for the data. For more information, see Supported file formats and compression codecs. Supported types are: **GZip**, **Deflate**, **BZip2**, and **ZipDeflate**. Supported levels are: **Optimal** and **Fastest**.	No

TIP

To copy all files under a folder, specify **folderPath** only.
To copy a single file with a given name, specify **folderPath** with folder part and **fileName** with file name.
To copy a subset of files under a folder, specify **folderPath** with folder part and **fileName** with wildcard filter.

NOTE

If you were using "fileFilter" property for file filter, it is still supported as-is, while you are suggested to use the new filter capability added to "fileName" going forward.

Example:

```
{
    "name": "FileSystemDataset",
    "properties": {
        "type": "FileShare",
        "linkedServiceName":{
            "referenceName": "<file system linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "folderPath": "folder/subfolder/",
            "fileName": "*",
            "modifiedDatetimeStart": "2018-12-01T05:00:00Z",
            "modifiedDatetimeEnd": "2018-12-01T06:00:00Z",
            "format": {
                "type": "TextFormat",
                "columnDelimiter": ",",
                "rowDelimiter": "\n"
            },
            "compression": {
                "type": "GZip",
                "level": "Optimal"
            }
        }
    }
}
```

Legacy copy activity source model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **FileSystemSource**	Yes
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note when recursive is set to true and sink is file-based store, empty folder/sub-folder will not be copied/created at sink. Allowed values are: **true** (default), **false**	No

PROPERTY	DESCRIPTION	REQUIRED
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromFileSystem",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<file system input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "FileSystemSource",
                "recursive": true
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Legacy copy activity sink model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to: **FileSystemSink**	Yes

PROPERTY	DESCRIPTION	REQUIRED
copyBehavior	Defines the copy behavior when the source is files from file-based data store. Allowed values are: - **PreserveHierarchy (default)**: preserves the file hierarchy in the target folder. The relative path of source file to source folder is identical to the relative path of target file to target folder. - **FlattenHierarchy**: all files from the source folder are in the first level of target folder. The target files have autogenerated name. - **MergeFiles**: merges all files from the source folder to one file. No record deduplication is performed during the merge. If the File Name is specified, the merged file name would be the specified name; otherwise, would be autogenerated file name.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToFileSystem",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<file system output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "FileSystemSink",
                "copyBehavior": "PreserveHierarchy"
            }
        }
    }
]
```

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from FTP server using Azure Data Factory or Synapse Analytics

8/19/2022 • 12 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to copy data from FTP server. To learn about more, read the introductory articles for Azure Data Factory and Synapse Analytics.

Supported capabilities

This FTP connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②
GetMetadata activity	① ②
Delete activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

Specifically, this FTP connector supports:

- Copying files using **Basic** or **Anonymous** authentication.
- Copying files as-is or parsing files with the supported file formats and compression codecs.

The FTP connector support FTP server running in passive mode. Active mode is not supported.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool

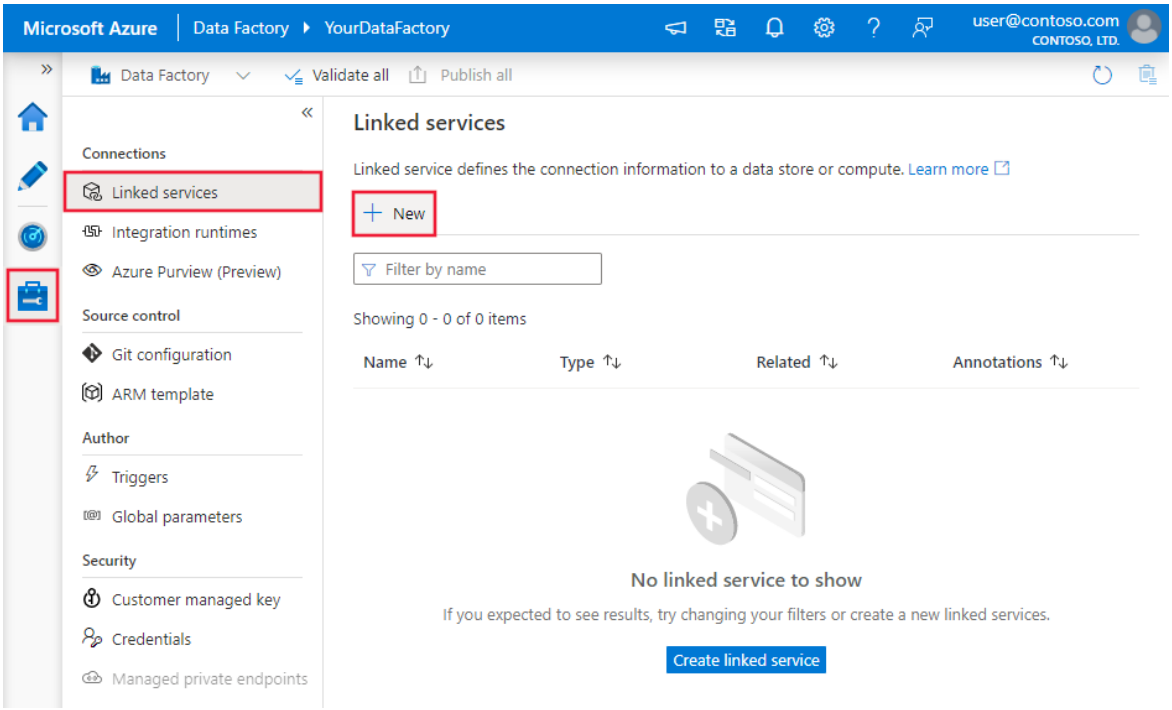
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to an FTP server using UI

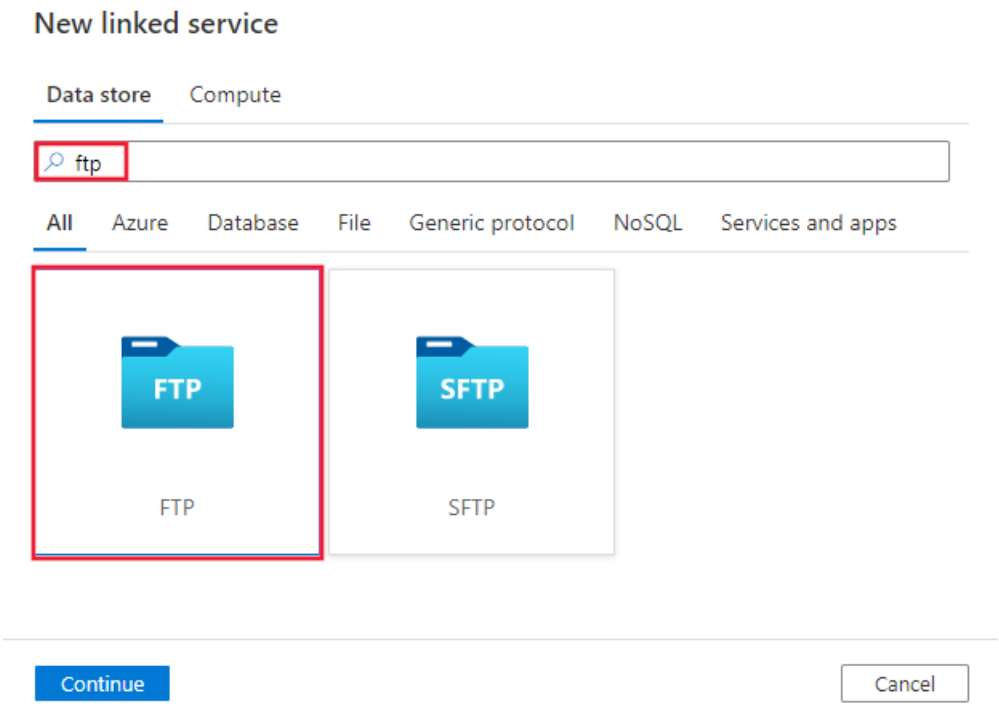
Use the following steps to create a linked service to an FTP server in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

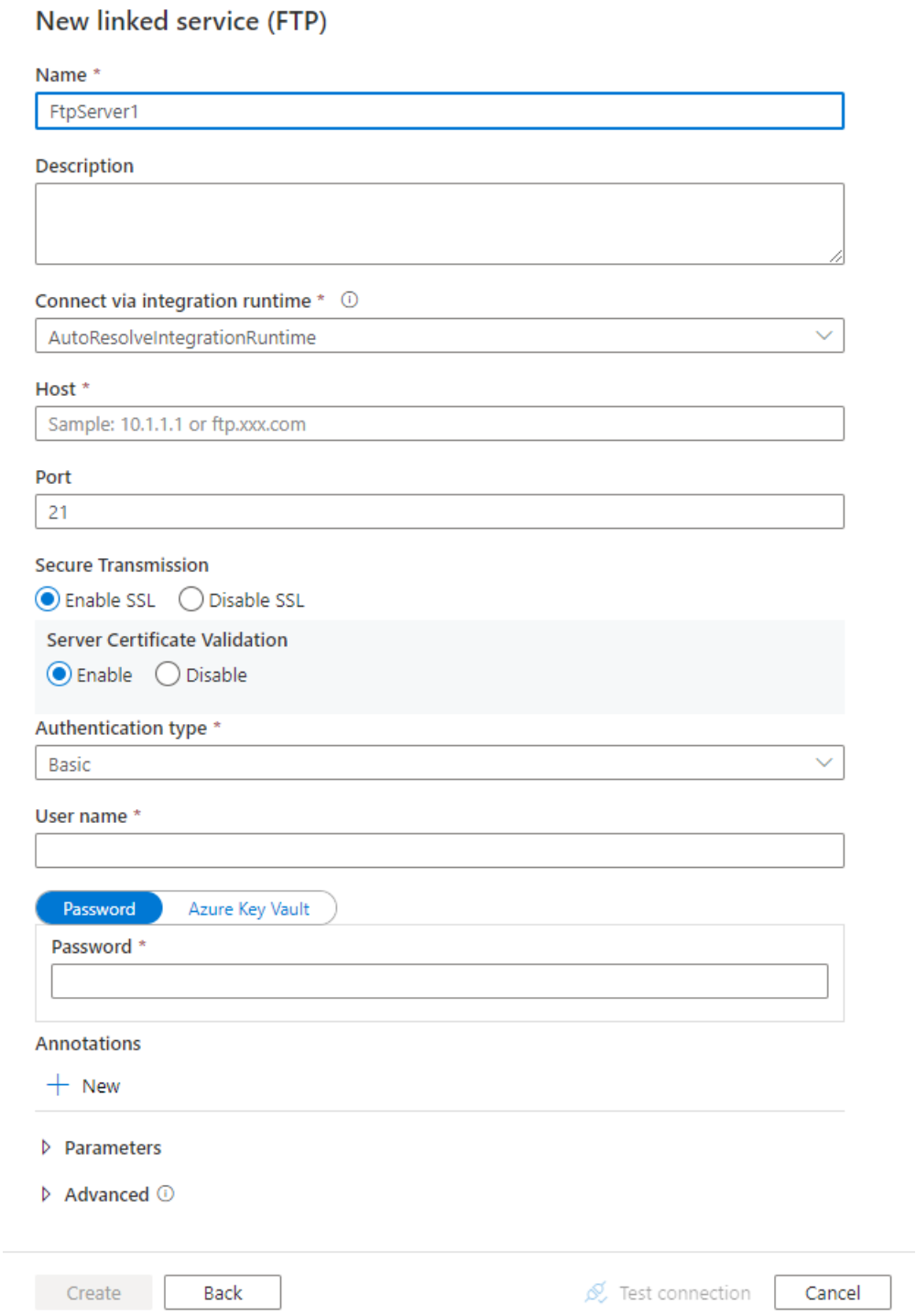
   - Azure Data Factory
   - Azure Synapse



2. Search for FTP and select the FTP connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to FTP.

Linked service properties

The following properties are supported for FTP linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **FtpServer**.	Yes
host	Specify the name or IP address of the FTP server.	Yes

PROPERTY	DESCRIPTION	REQUIRED
port	Specify the port on which the FTP server is listening. Allowed values are: integer, default value is **21**.	No
enableSsl	Specify whether to use FTP over an SSL/TLS channel. Allowed values are: **true** (default), **false**.	No
enableServerCertificateValidation	Specify whether to enable server TLS/SSL certificate validation when you are using FTP over SSL/TLS channel. Allowed values are: **true** (default), **false**.	No
authenticationType	Specify the authentication type. Allowed values are: **Basic**, **Anonymous**	Yes
userName	Specify the user who has access to the FTP server.	No
password	Specify the password for the user (userName). Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

NOTE

The FTP connector supports accessing FTP server with either no encryption or explicit SSL/TLS encryption; it doesn't support implicit SSL/TLS encryption.

Example 1: using Anonymous authentication

```
{
    "name": "FTPLinkedService",
    "properties": {
        "type": "FtpServer",
        "typeProperties": {
            "host": "<ftp server>",
            "port": 21,
            "enableSsl": true,
            "enableServerCertificateValidation": true,
            "authenticationType": "Anonymous"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 2: using Basic authentication

```
{
    "name": "FTPLinkedService",
    "properties": {
        "type": "FtpServer",
        "typeProperties": {
            "host": "<ftp server>",
            "port": 21,
            "enableSsl": true,
            "enableServerCertificateValidation": true,
            "authenticationType": "Basic",
            "userName": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for FTP under `location` settings in format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `location` in dataset must be set to **FtpServerLocation**.	Yes
folderPath	The path to folder. If you want to use wildcard to filter folder, skip this setting and specify in activity source settings.	No
fileName	The file name under the given folderPath. If you want to use wildcard to filter files, skip this setting and specify in activity source settings.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<FTP linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "FtpServerLocation",
                "folderPath": "root/folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by FTP source.

FTP as source

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for FTP under `storeSettings` settings in format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `storeSettings` must be set to **FtpReadSettings**.	Yes
Locate the files to copy:		
OPTION 1: static path	Copy from the given folder/file path specified in the dataset. If you want to copy all files from a folder, additionally specify `wildcardFileName` as `*`.	
OPTION 2: wildcard - wildcardFolderPath	The folder path with wildcard characters to filter source folders. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual folder name has wildcard or this escape char inside. See more examples in Folder and file filter examples.	No
OPTION 2: wildcard - wildcardFileName	The file name with wildcard characters under the given folderPath/wildcardFolderPath to filter source files. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual file name has wildcard or this escape char inside. See more examples in Folder and file filter examples.	Yes
OPTION 3: a list of files - fileListPath	Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset. When using this option, do not specify file name in dataset. See more examples in File list examples.	No
Additional settings:		
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when recursive is set to true and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**. This property doesn't apply when you configure `fileListPath`.	No

PROPERTY	DESCRIPTION	REQUIRED
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No
useBinaryTransfer	Specify whether to use the binary transfer mode. The values are true for binary mode (default), and false for ASCII.	No
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

PROPERTY	DESCRIPTION	REQUIRED
disableChunking	When copying data from FTP, the service tries to get the file length first, then divide the file into multiple parts and read them in parallel. Specify whether your FTP server supports getting file length or seeking to read from a certain offset. Allowed values are **false** (default), **true**.	No

Example:

```
"activities":[
    {
        "name": "CopyFromFTP",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "FtpReadSettings",
                    "recursive": true,
                    "wildcardFolderPath": "myfolder*A",
                    "wildcardFileName": "*.csv",
                    "disableChunking": false
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`Folder*`	(empty, use default)	false	FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	(empty, use default)	true	FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv
`Folder*`	`*.csv`	false	FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	`*.csv`	true	FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This section describes the resulting behavior of using file list path in copy activity source.

Assuming you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	CONFIGURATION
root FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In dataset:** - Folder path: `root/FolderA` **In copy activity source:** - File list path: `root/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line with the relative path to the path configured in the dataset.

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity

Delete activity properties

To learn details about the properties, check Delete activity

Legacy models

> **NOTE**
> The following models are still supported as-is for backward compatibility. You are suggested to use the new model mentioned in above sections going forward, and the authoring UI has switched to generating the new model.

Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **FileShare**	Yes
folderPath	Path to the folder. Wildcard filter is supported, allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual folder name has wildcard or this escape char inside. Examples: rootfolder/subfolder/, see more examples in Folder and file filter examples.	Yes

PROPERTY	DESCRIPTION	REQUIRED
fileName	**Name or wildcard filter** for the file(s) under the specified "folderPath". If you don't specify a value for this property, the dataset points to all files in the folder. For filter, allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). - Example 1: `"fileName": "*.csv"` - Example 2: `"fileName": "???20180427.txt"` Use `^` to escape if your actual file name has wildcard or this escape char inside.	No
format	If you want to **copy files as-is** between file-based stores (binary copy), skip the format section in both input and output dataset definitions. If you want to parse files with a specific format, the following file format types are supported: **TextFormat**, **JsonFormat**, **AvroFormat**, **OrcFormat**, **ParquetFormat**. Set the **type** property under format to one of these values. For more information, see Text Format, Json Format, Avro Format, Orc Format, and Parquet Format sections.	No (only for binary copy scenario)
compression	Specify the type and level of compression for the data. For more information, see Supported file formats and compression codecs. Supported types are: **GZip**, **Deflate**, **BZip2**, and **ZipDeflate**. Supported levels are: **Optimal** and **Fastest**.	No
useBinaryTransfer	Specify whether to use the binary transfer mode. The values are true for binary mode (default), and false for ASCII.	No

TIP

To copy all files under a folder, specify **folderPath** only.
To copy a single file with a given name, specify **folderPath** with folder part and **fileName** with file name.
To copy a subset of files under a folder, specify **folderPath** with folder part and **fileName** with wildcard filter.

NOTE

If you were using "fileFilter" property for file filter, it is still supported as-is, while you are suggested to use the new filter capability added to "fileName" going forward.

Example:

```
{
    "name": "FTPDataset",
    "properties": {
        "type": "FileShare",
        "linkedServiceName":{
            "referenceName": "<FTP linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "folderPath": "folder/subfolder/",
            "fileName": "myfile.csv.gz",
            "format": {
                "type": "TextFormat",
                "columnDelimiter": ",",
                "rowDelimiter": "\n"
            },
            "compression": {
                "type": "GZip",
                "level": "Optimal"
            }
        }
    }
}
```

Legacy copy activity source model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **FileSystemSource**	Yes
recursive	Indicates whether the data is read recursively from the sub folders or only from the specified folder. Note when recursive is set to true and sink is file-based store, empty folder/sub-folder will not be copied/created at sink. Allowed values are: **true** (default), **false**	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromFTP",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<FTP input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "FileSystemSource",
                "recursive": true
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Use GitHub to read Common Data Model entity references

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

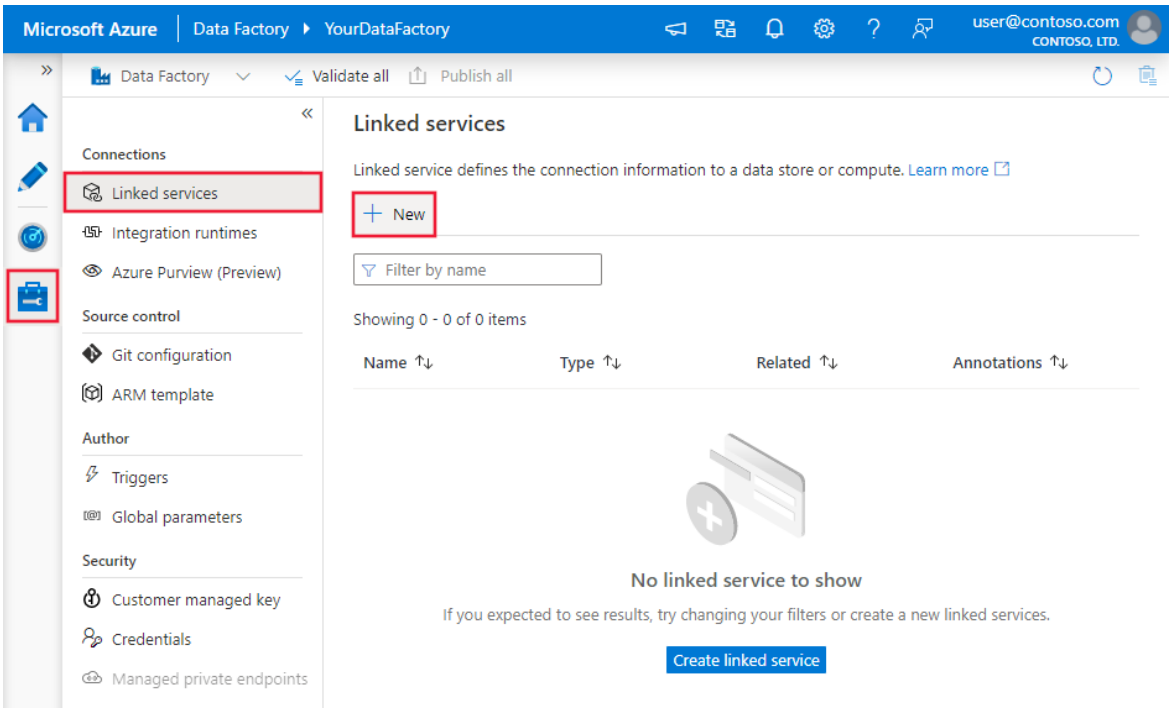
The GitHub connector in Azure Data Factory and Synapse Analytics pipelines is only used to receive the entity reference schema for the Common Data Model format in mapping data flow.

Create a linked service to GitHub using UI

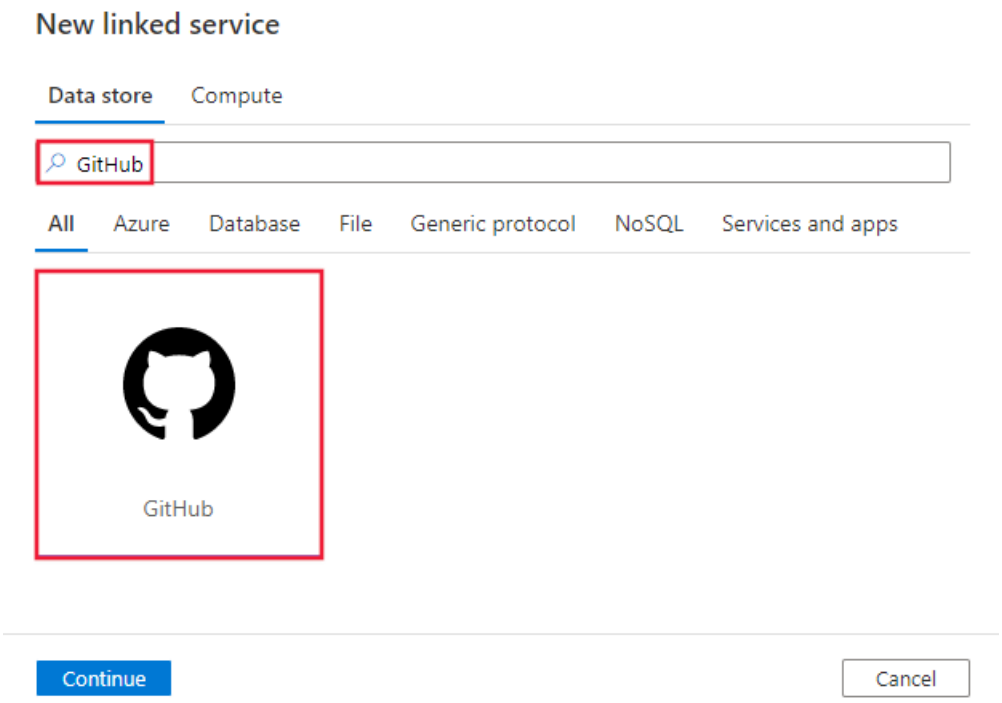
Use the following steps to create a linked service to GitHub in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

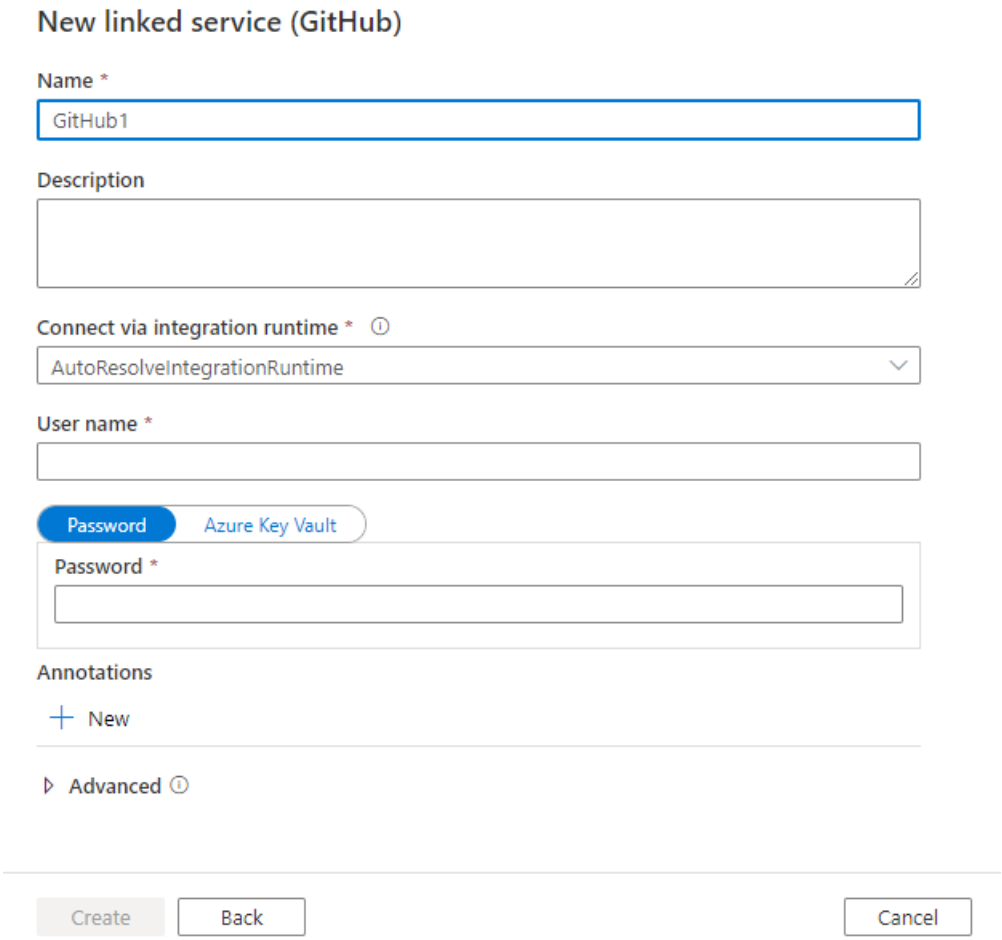
   - Azure Data Factory
   - Azure Synapse



2. Search for GitHub and select the GitHub connector.



3. Configure the service details, test the connection, and create the new linked service.



Linked service properties

The following properties are supported for the GitHub linked service.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **GitHub**.	yes
userName	GitHub username	yes
password	GitHub password	yes

Next steps

Create a source dataset in mapping data flow.

Copy data from Google AdWords using Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Google AdWords. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Google AdWords connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

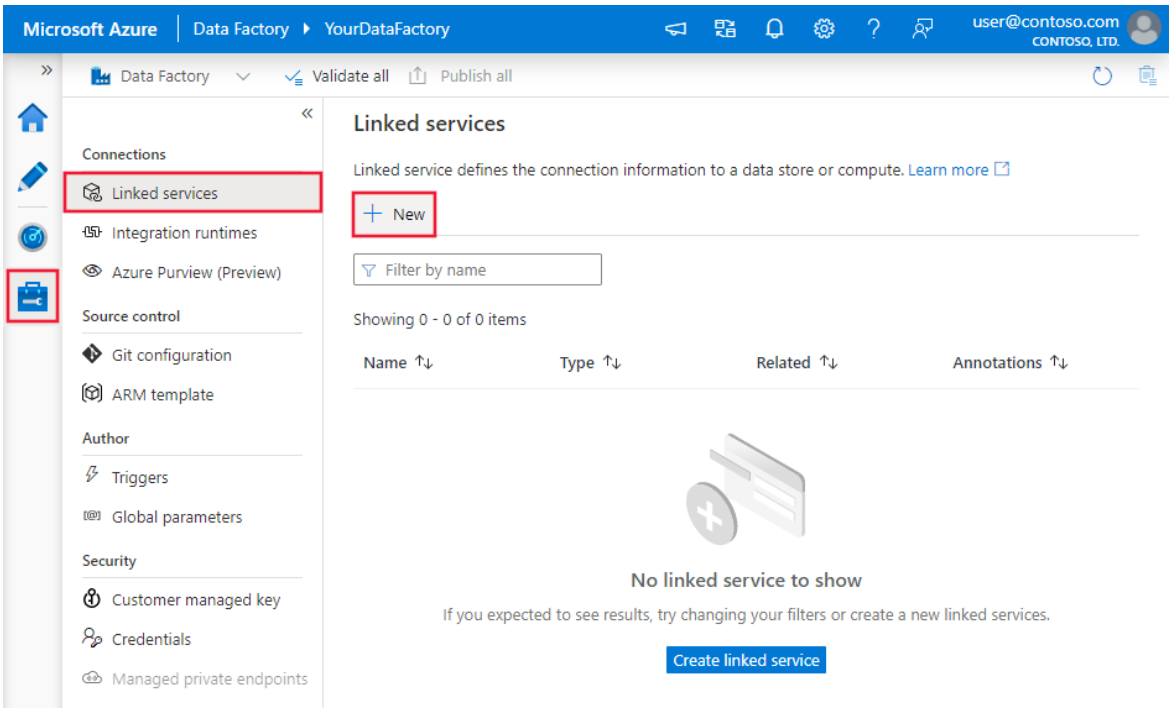
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Google AdWords using UI

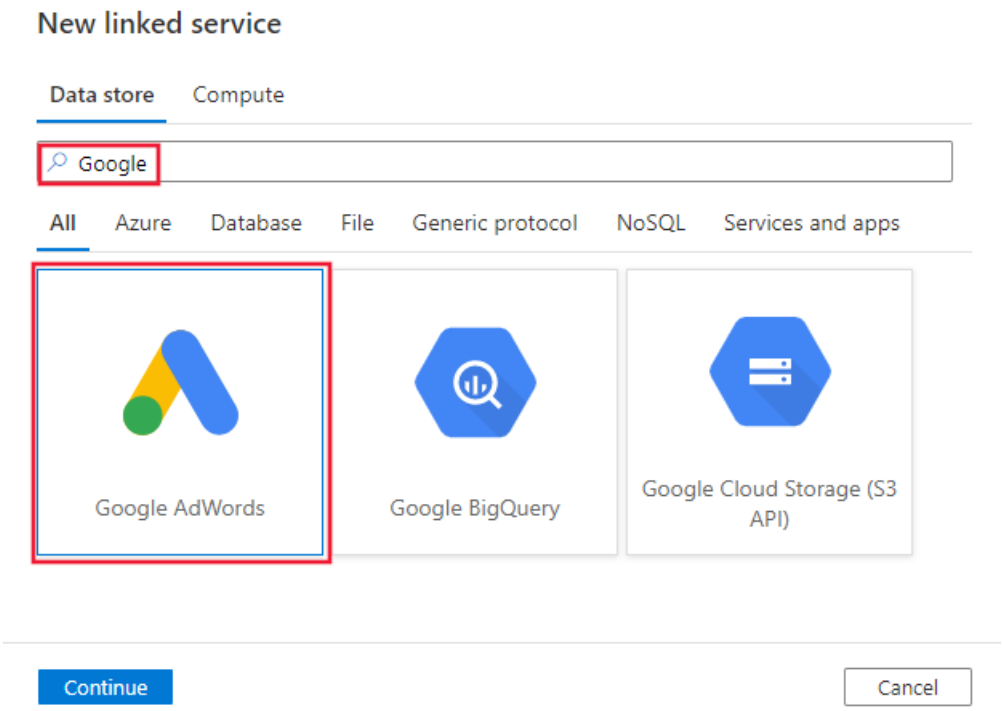
Use the following steps to create a linked service to Google AdWords in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

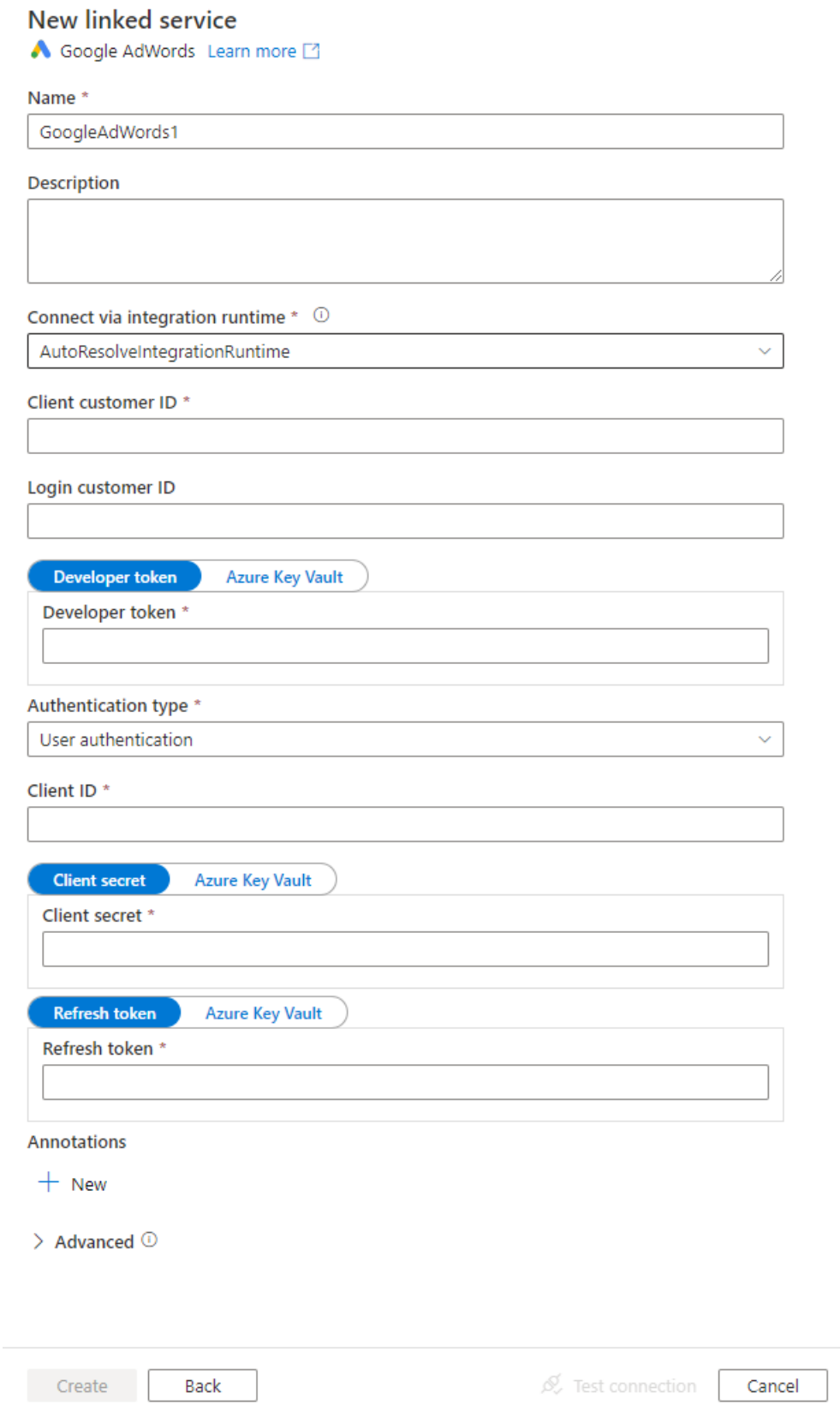
   - Azure Data Factory
   - Azure Synapse



2. Search for Google and select the Google AdWords connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Google AdWords connector.

Linked service properties

IMPORTANT

Due to the sunset of Google AdWords API by **April 27, 2022**, the service has upgraded to the new Google Ads API. Please refer this document for detailed migration steps and recommendations. Please make sure the migration to be done before **April 27, 2022**.

The following properties are supported for Google AdWords linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **GoogleAdWords**	Yes
connectionProperties	A group of properties that defines how to connect to Google AdWords.	Yes
Under `connectionProperties`***:***		
clientCustomerID	The Client customer ID of the AdWords account that you want to fetch report data for.	Yes
loginCustomerID	The customer ID of the Google AdWords manager account through which you want to fetch report data of specific customer.	No
developerToken	The developer token associated with the manager account that you use to grant access to the AdWords API. You can choose to mark this field as a SecureString to store it securely, or store password in Azure Key Vault and let the copy activity pull from there when performing data copy - learn more from Store credentials in Key Vault.	Yes
authenticationType	The OAuth 2.0 authentication mechanism used for authentication. ServiceAuthentication can only be used on self-hosted IR. Allowed values are: **ServiceAuthentication**, **UserAuthentication**	Yes
refreshToken	The refresh token obtained from Google for authorizing access to AdWords for UserAuthentication. You can choose to mark this field as a SecureString to store it securely, or store password in Azure Key Vault and let the copy activity pull from there when performing data copy - learn more from Store credentials in Key Vault.	No

PROPERTY	DESCRIPTION	REQUIRED
clientId	The client ID of the Google application used to acquire the refresh token. You can choose to mark this field as a SecureString to store it securely, or store password in Azure Key Vault and let the copy activity pull from there when performing data copy - learn more from Store credentials in Key Vault.	No
clientSecret	The client secret of the google application used to acquire the refresh token. You can choose to mark this field as a SecureString to store it securely, or store password in Azure Key Vault and let the copy activity pull from there when performing data copy - learn more from Store credentials in Key Vault.	No
email	The service account email ID that is used for ServiceAuthentication and can only be used on self-hosted IR.	No
keyFilePath	The full path to the `.p12` or `.json` key file that is used to authenticate the service account email address and can only be used on self-hosted IR.	No
trustedCertPath	The full path of the .pem file containing trusted CA certificates for verifying the server when connecting over TLS. This property can only be set when using TLS on self-hosted IR. The default value is the cacerts.pem file installed with the IR.	No
useSystemTrustStore	Specifies whether to use a CA certificate from the system trust store or from a specified PEM file. The default value is false.	No

Example:

```
{
    "name": "GoogleAdWordsLinkedService",
    "properties": {
        "type": "GoogleAdWords",
        "typeProperties": {
            "connectionProperties": {
                "clientCustomerID": "<clientCustomerID>",
                "loginCustomerID": "<loginCustomerID>",
                "developerToken": {
                    "type": "SecureString",
                    "value": "<developerToken>"
                },
                "authenticationType": "ServiceAuthentication",
                "refreshToken": {
                    "type": "SecureString",
                    "value": "<refreshToken>"
                },
                "clientId": {
                    "type": "SecureString",
                    "value": "<clientId>"
                },
                "clientSecret": {
                    "type": "SecureString",
                    "value": "<clientSecret>"
                },
                "email": "<email>",
                "keyFilePath": "<keyFilePath>",
                "trustedCertPath": "<trustedCertPath>",
                "useSystemTrustStore": true,
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Google AdWords dataset.

To copy data from Google AdWords, set the type property of the dataset to **GoogleAdWordsObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **GoogleAdWordsObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "GoogleAdWordsDataset",
    "properties": {
        "type": "GoogleAdWordsObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<GoogleAdWords linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Google AdWords source.

Google AdWords as source

To copy data from Google AdWords, set the source type in the copy activity to **GoogleAdWordsSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **GoogleAdWordsSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromGoogleAdWords",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<GoogleAdWords input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "GoogleAdWordsSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Google BigQuery using Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from Google BigQuery. It builds on the Copy Activity overview article that presents a general overview of the copy activity.

Supported capabilities

This Google BigQuery connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity. Therefore, you don't need to manually install a driver to use this connector.

> **NOTE**
> This Google BigQuery connector is built on top of the BigQuery APIs. Be aware that BigQuery limits the maximum rate of incoming requests and enforces appropriate quotas on a per-project basis, refer to Quotas & Limits - API requests. Make sure you do not trigger too many concurrent requests to the account.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

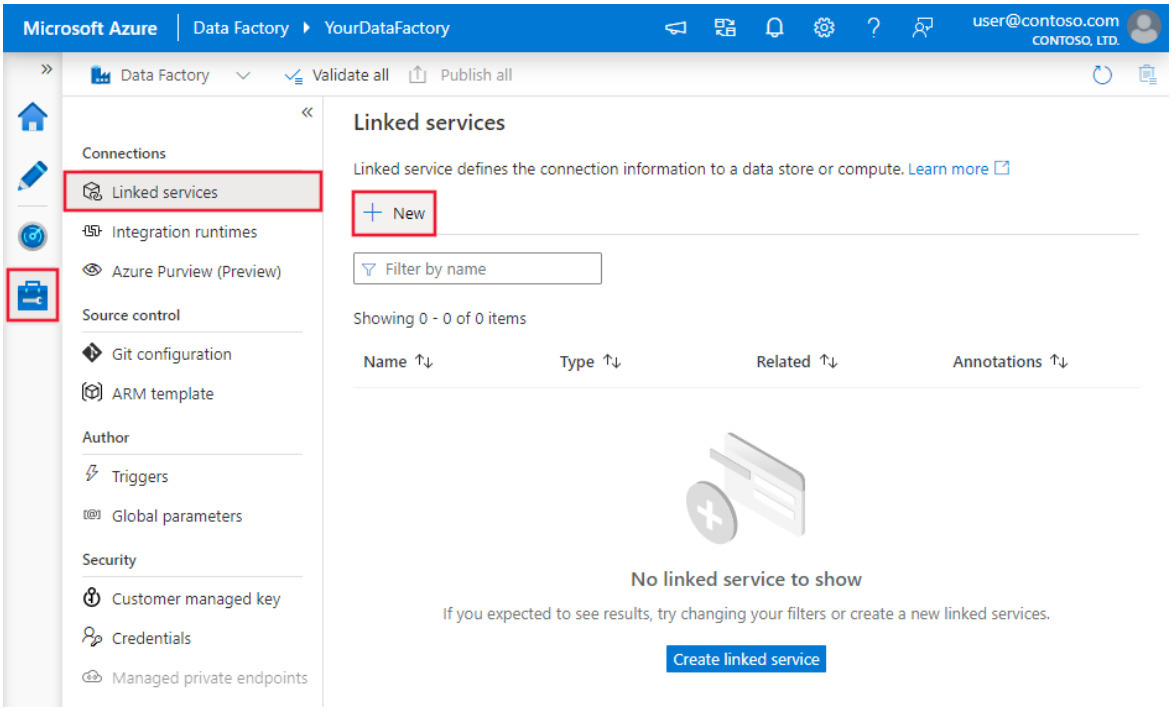
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Google BigQuery using UI

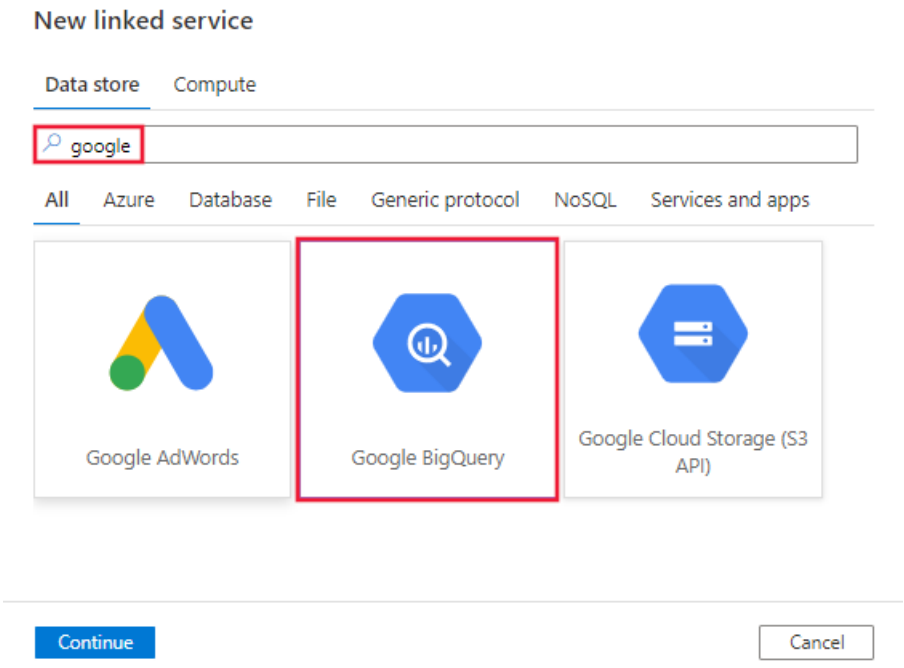
Use the following steps to create a linked service to Google BigQuery in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

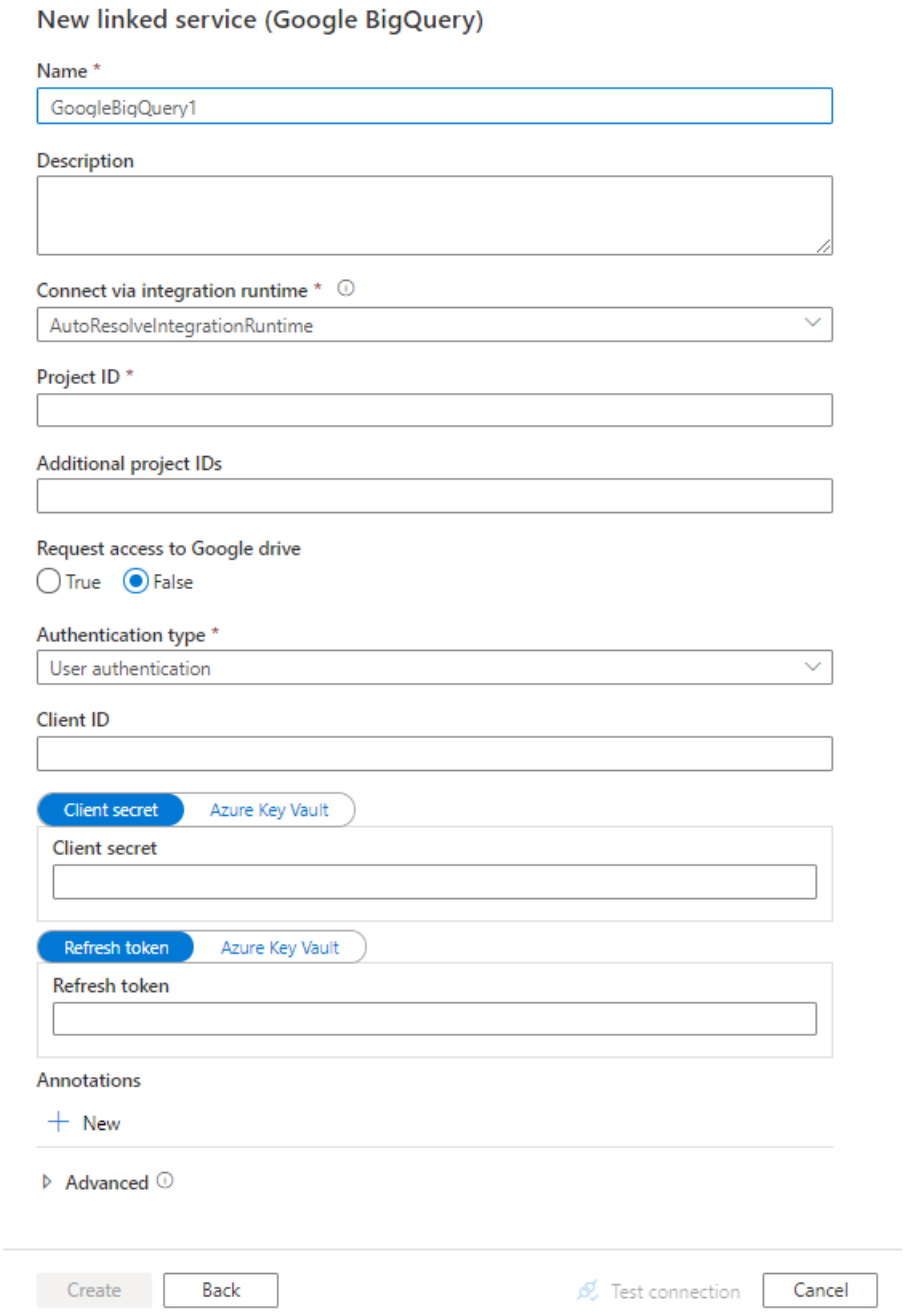
   - Azure Data Factory
   - Azure Synapse



2. Search for Google and select the Google BigQuery connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to the Google BigQuery connector.

Linked service properties

The following properties are supported for the Google BigQuery linked service.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **GoogleBigQuery**.	Yes
project	The project ID of the default BigQuery project to query against.	Yes

PROPERTY	DESCRIPTION	REQUIRED
additionalProjects	A comma-separated list of project IDs of public BigQuery projects to access.	No
requestGoogleDriveScope	Whether to request access to Google Drive. Allowing Google Drive access enables support for federated tables that combine BigQuery data with data from Google Drive. The default value is **false**.	No
authenticationType	The OAuth 2.0 authentication mechanism used for authentication. ServiceAuthentication can be used only on Self-hosted Integration Runtime. Allowed values are **UserAuthentication** and **ServiceAuthentication**. Refer to sections below this table on more properties and JSON samples for those authentication types respectively.	Yes

Using user authentication

Set "authenticationType" property to **UserAuthentication**, and specify the following properties along with generic properties described in the previous section:

PROPERTY	DESCRIPTION	REQUIRED
clientId	ID of the application used to generate the refresh token.	No
clientSecret	Secret of the application used to generate the refresh token. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
refreshToken	The refresh token obtained from Google used to authorize access to BigQuery. Learn how to get one from Obtaining OAuth 2.0 access tokens and this community blog. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No

The minimum scope required to obtain an OAuth 2.0 refresh token is `https://www.googleapis.com/auth/bigquery.readonly`. If you plan to run a query that might return large results, other scope might be required. For more information, refer to this article.

Example:

```
{
    "name": "GoogleBigQueryLinkedService",
    "properties": {
        "type": "GoogleBigQuery",
        "typeProperties": {
            "project" : "<project ID>",
            "additionalProjects" : "<additional project IDs>",
            "requestGoogleDriveScope" : true,
            "authenticationType" : "UserAuthentication",
            "clientId": "<id of the application used to generate the refresh token>",
            "clientSecret": {
                "type": "SecureString",
                "value":"<secret of the application used to generate the refresh token>"
            },
            "refreshToken": {
                "type": "SecureString",
                "value": "<refresh token>"
            }
        }
    }
}
```

Using service authentication

Set "authenticationType" property to **ServiceAuthentication**, and specify the following properties along with generic properties described in the previous section. This authentication type can be used only on Self-hosted Integration Runtime.

PROPERTY	DESCRIPTION	REQUIRED
email	The service account email ID that is used for ServiceAuthentication. It can be used only on Self-hosted Integration Runtime.	No
keyFilePath	The full path to the `.p12` or `.json` key file that is used to authenticate the service account email address.	No
trustedCertPath	The full path of the .pem file that contains trusted CA certificates used to verify the server when you connect over TLS. This property can be set only when you use TLS on Self-hosted Integration Runtime. The default value is the cacerts.pem file installed with the integration runtime.	No
useSystemTrustStore	Specifies whether to use a CA certificate from the system trust store or from a specified .pem file. The default value is **false**.	No

Example:

```
{
    "name": "GoogleBigQueryLinkedService",
    "properties": {
        "type": "GoogleBigQuery",
        "typeProperties": {
            "project" : "<project id>",
            "requestGoogleDriveScope" : true,
            "authenticationType" : "ServiceAuthentication",
            "email": "<email>",
            "keyFilePath": "<.p12 or .json key path on the IR machine>"
        },
        "connectVia": {
            "referenceName": "<name of Self-hosted Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the Google BigQuery dataset.

To copy data from Google BigQuery, set the type property of the dataset to **GoogleBigQueryObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **GoogleBigQueryObject**	Yes
dataset	Name of the Google BigQuery dataset.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table. This property is supported for backward compatibility. For new workload, use `dataset` and `table`.	No (if "query" in activity source is specified)

Example

```
{
    "name": "GoogleBigQueryDataset",
    "properties": {
        "type": "GoogleBigQueryObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<GoogleBigQuery linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the Google BigQuery source type.

GoogleBigQuerySource as a source type

To copy data from Google BigQuery, set the source type in the copy activity to **GoogleBigQuerySource**. The following properties are supported in the copy activity **source** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **GoogleBigQuerySource**.	Yes
query	Use the custom SQL query to read data. An example is `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromGoogleBigQuery",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<GoogleBigQuery input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "GoogleBigQuerySource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from Google Cloud Storage using Azure Data Factory or Synapse Analytics

8/19/2022 • 9 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to copy data from Google Cloud Storage (GCS). To learn more, read the introductory articles for Azure Data Factory and Synapse Analytics.

Supported capabilities

This Google Cloud Storage connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②
GetMetadata activity	① ②
Delete activity	① ②

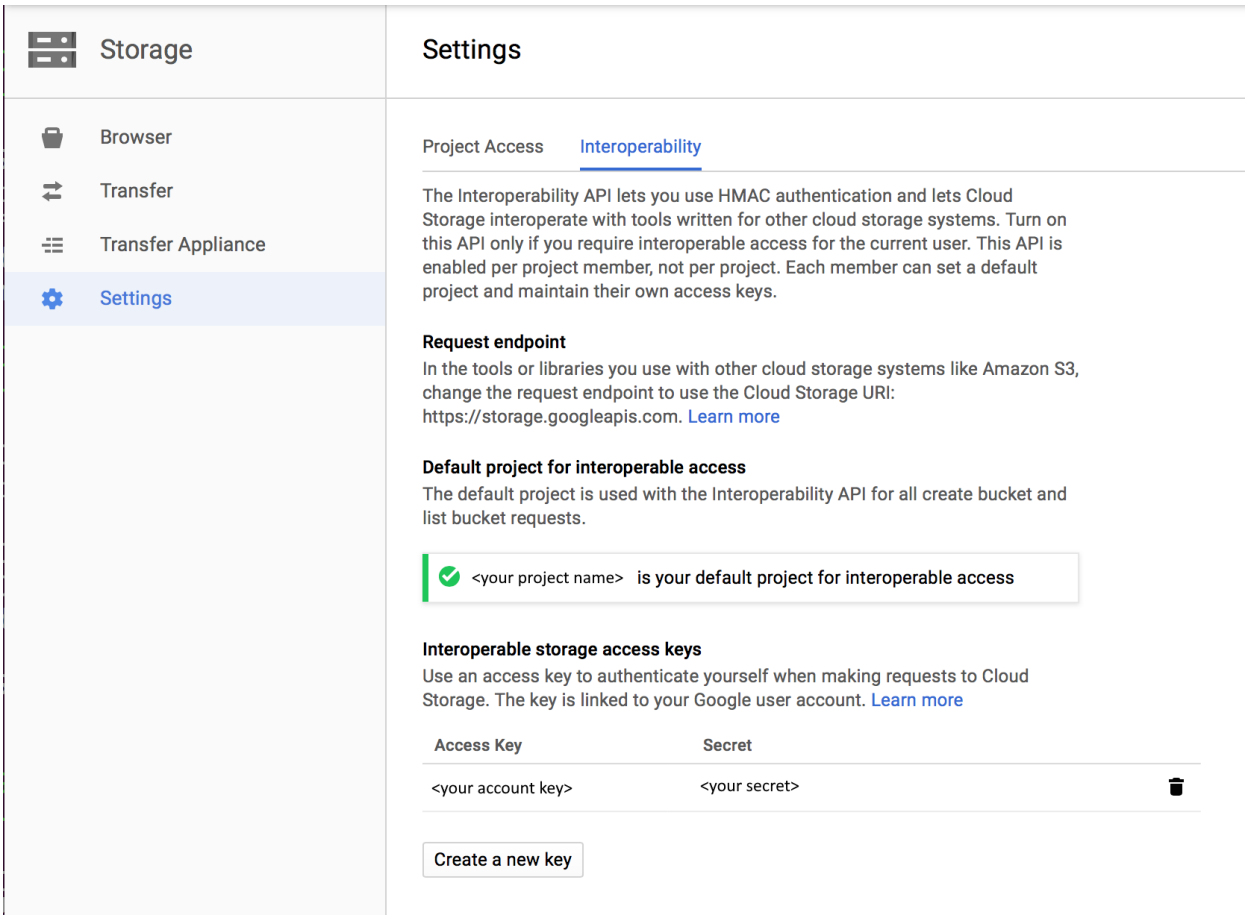
① Azure integration runtime ② Self-hosted integration runtime

Specifically, this Google Cloud Storage connector supports copying files as is or parsing files with the supported file formats and compression codecs. It takes advantage of GCS's S3-compatible interoperability.

Prerequisites

The following setup is required on your Google Cloud Storage account:

1. Enable interoperability for your Google Cloud Storage account
2. Set the default project that contains the data you want to copy from the target GCS bucket.
3. Create a service account and define the right levels of permissions by using Cloud IAM on GCP.
4. Generate the access keys for this service account.



Required permissions

To copy data from Google Cloud Storage, make sure you've been granted the following permissions for object operations: `storage.objects.get` and `storage.objects.list`.

If you use UI to author, additional `storage.buckets.list` permission is required for operations like testing connection to linked service and browsing from root. If you don't want to grant this permission, you can choose "Test connection to file path" or "Browse from specified path" options from the UI.

For the full list of Google Cloud Storage roles and associated permissions, see IAM roles for Cloud Storage on the Google Cloud site.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

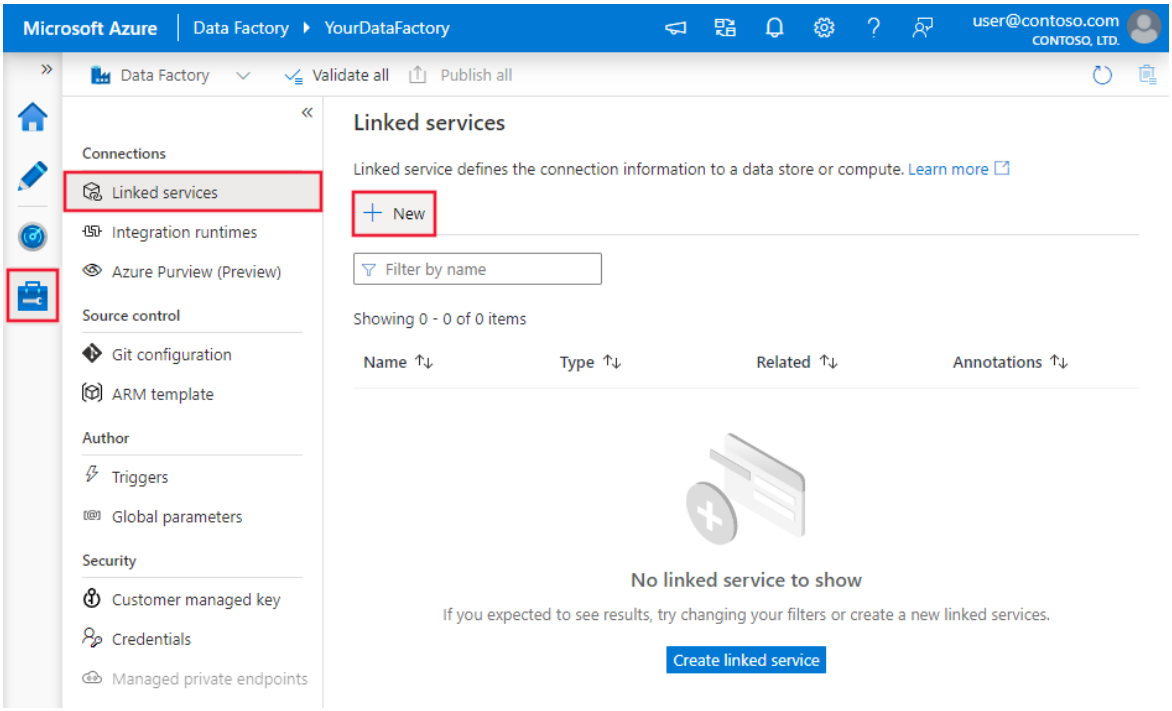
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Google Cloud Storage using UI

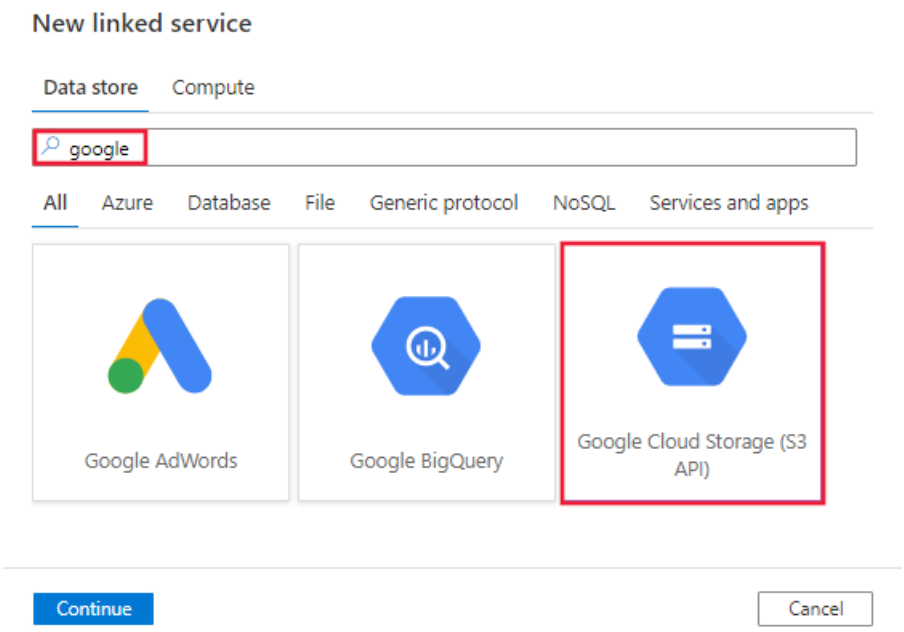
Use the following steps to create a linked service to Google Cloud Storage in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

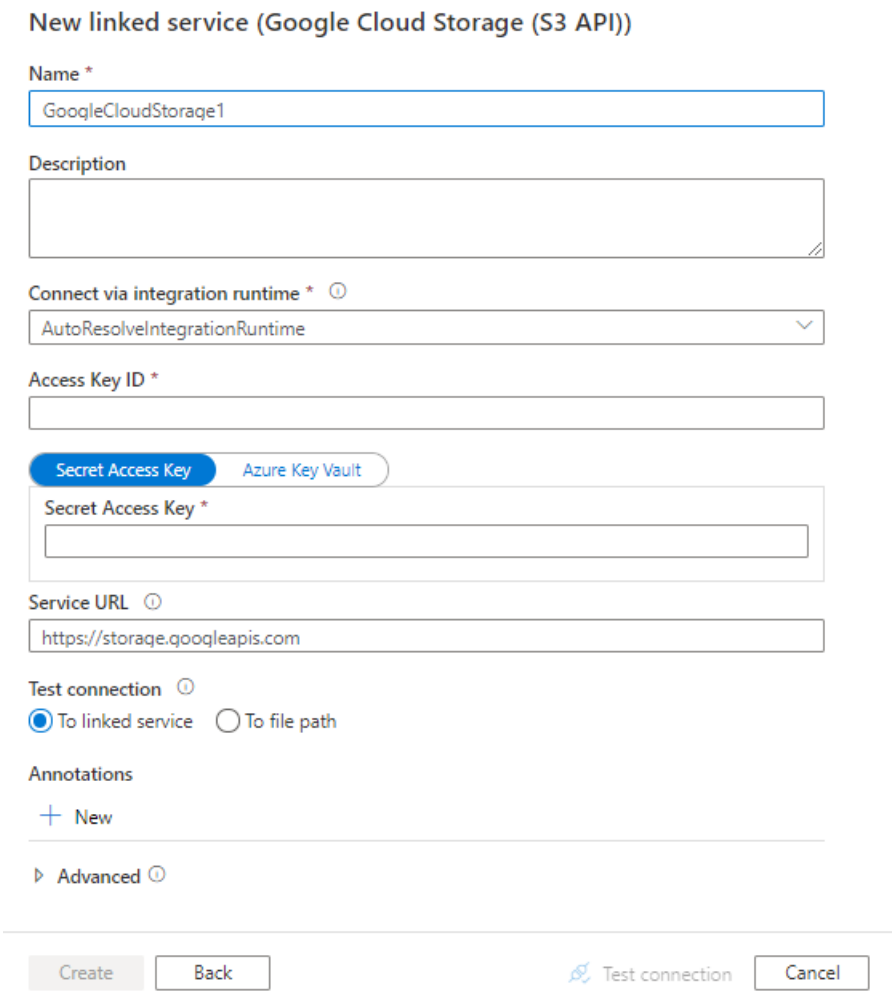
- Azure Data Factory
- Azure Synapse



2. Search for Google and select the Google Cloud Storage (S3 API) connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Google Cloud Storage.

Linked service properties

The following properties are supported for Google Cloud Storage linked services:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **GoogleCloudStorage**.	Yes
accessKeyId	ID of the secret access key. To find the access key and secret, see Prerequisites.	Yes
secretAccessKey	The secret access key itself. Mark this field as **SecureString** to store it securely, or reference a secret stored in Azure Key Vault.	Yes
serviceUrl	Specify the custom GCS endpoint as `https://storage.googleapis.com`.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

Here's an example:

```
{
    "name": "GoogleCloudStorageLinkedService",
    "properties": {
        "type": "GoogleCloudStorage",
        "typeProperties": {
            "accessKeyId": "<access key id>",
            "secretAccessKey": {
                "type": "SecureString",
                "value": "<secret access key>"
            },
            "serviceUrl": "https://storage.googleapis.com"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Google Cloud Storage under `location` settings in a format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property under `location` in the dataset must be set to **GoogleCloudStorageLocation**.	Yes
bucketName	The GCS bucket name.	Yes

PROPERTY	DESCRIPTION	REQUIRED
folderPath	The path to folder under the given bucket. If you want to use a wildcard to filter the folder, skip this setting and specify that in activity source settings.	No
fileName	The file name under the given bucket and folder path. If you want to use a wildcard to filter the files, skip this setting and specify that in activity source settings.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<Google Cloud Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "GoogleCloudStorageLocation",
                "bucketName": "bucketname",
                "folderPath": "folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties that the Google Cloud Storage source supports.

Google Cloud Storage as a source type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Google Cloud Storage under `storeSettings` settings in a format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property under `storeSettings` must be set to **GoogleCloudStorageReadSettings**.	Yes
Locate the files to copy:		
OPTION 1: static path	Copy from the given bucket or folder/file path specified in the dataset. If you want to copy all files from a bucket or folder, additionally specify `wildcardFileName` as `*`.	
OPTION 2: GCS prefix - prefix	Prefix for the GCS key name under the given bucket configured in the dataset to filter source GCS files. GCS keys whose names start with `bucket_in_dataset/this_prefix` are selected. It utilizes GCS's service-side filter, which provides better performance than a wildcard filter.	No
OPTION 3: wildcard - wildcardFolderPath	The folder path with wildcard characters under the given bucket configured in a dataset to filter source folders. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your folder name has a wildcard or this escape character inside. See more examples in Folder and file filter examples.	No
OPTION 3: wildcard - wildcardFileName	The file name with wildcard characters under the given bucket and folder path (or wildcard folder path) to filter source files. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your file name has a wildcard or this escape character inside. See more examples in Folder and file filter examples.	Yes
OPTION 3: a list of files - fileListPath	Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset. When you're using this option, do not specify the file name in the dataset. See more examples in File list examples.	No
Additional settings:		

PROPERTY	DESCRIPTION	REQUIRED
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when **recursive** is set to **true** and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**. This property doesn't apply when you configure `fileListPath`.	No
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No
modifiedDatetimeStart	Files are filtered based on the attribute: last modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z". The properties can be **NULL**, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is **NULL**, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is **NULL**, the files whose last modified attribute is less than the datetime value will be selected. This property doesn't apply when you configure `fileListPath`.	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromGoogleCloudStorage",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "GoogleCloudStorageReadSettings",
                    "recursive": true,
                    "wildcardFolderPath": "myfolder*A",
                    "wildcardFileName": "*.csv"
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

BUCKET	KEY	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
bucket	`Folder*/*`	false	bucket FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv

BUCKET	KEY	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
bucket	`Folder*/*`	true	bucket FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv
bucket	`Folder*/*.csv`	false	bucket FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
bucket	`Folder*/*.csv`	true	bucket FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This section describes the resulting behavior of using a file list path in the Copy activity source.

Assume that you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	CONFIGURATION
bucket FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In dataset:** - Bucket: `bucket` - Folder path: `FolderA` **In copy activity source:** - File list path: `bucket/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line, with the relative path to the path configured in the dataset.

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity.

Delete activity properties

To learn details about the properties, check Delete activity.

Legacy models

If you were using an Amazon S3 connector to copy data from Google Cloud Storage, it's still supported as is for backward compatibility. We suggest that you use the new model mentioned earlier. The authoring UI has switched to generating the new model.

Next steps

For a list of data stores that the Copy activity supports as sources and sinks, see Supported data stores.

Copy data from Greenplum using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Greenplum. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Greenplum connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK

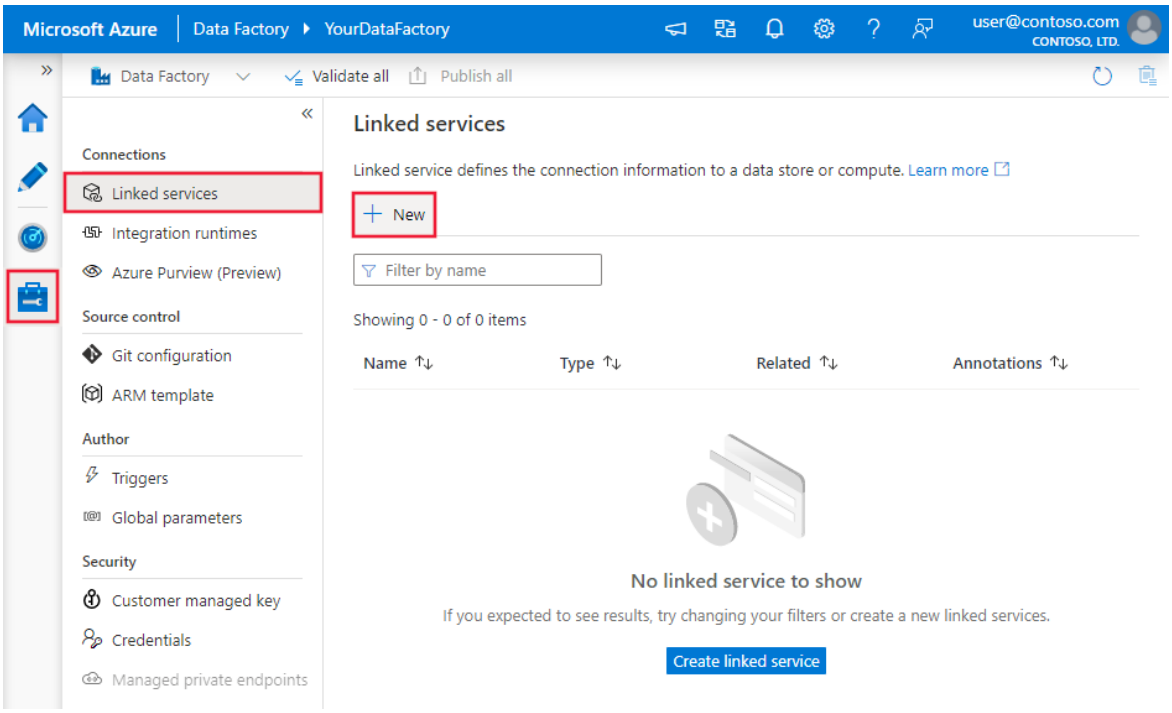
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Greenplum using UI

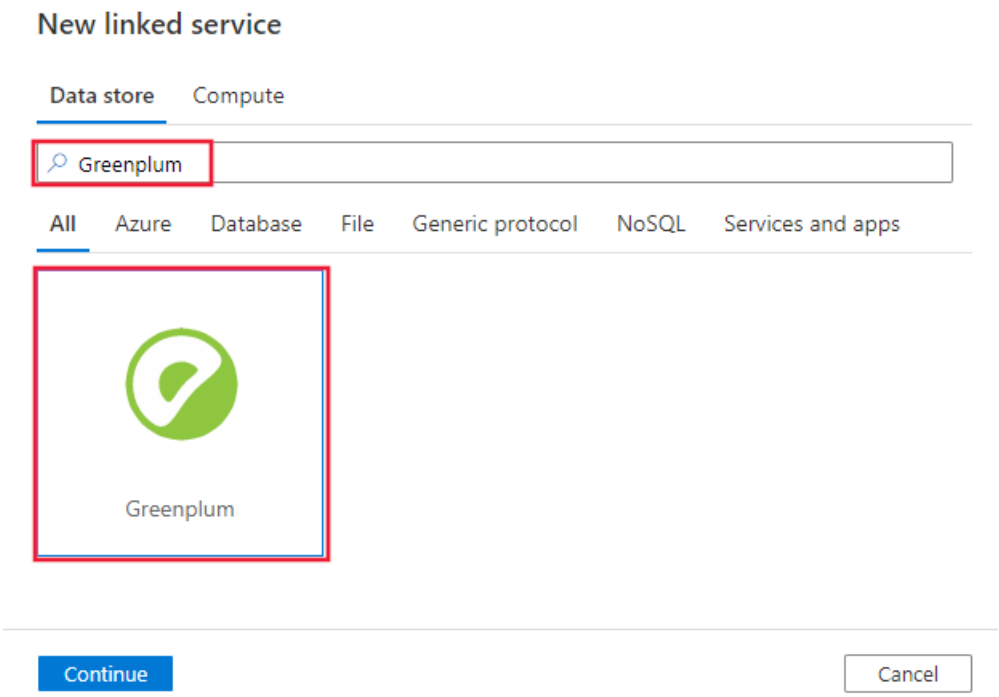
Use the following steps to create a linked service to Greenplum in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

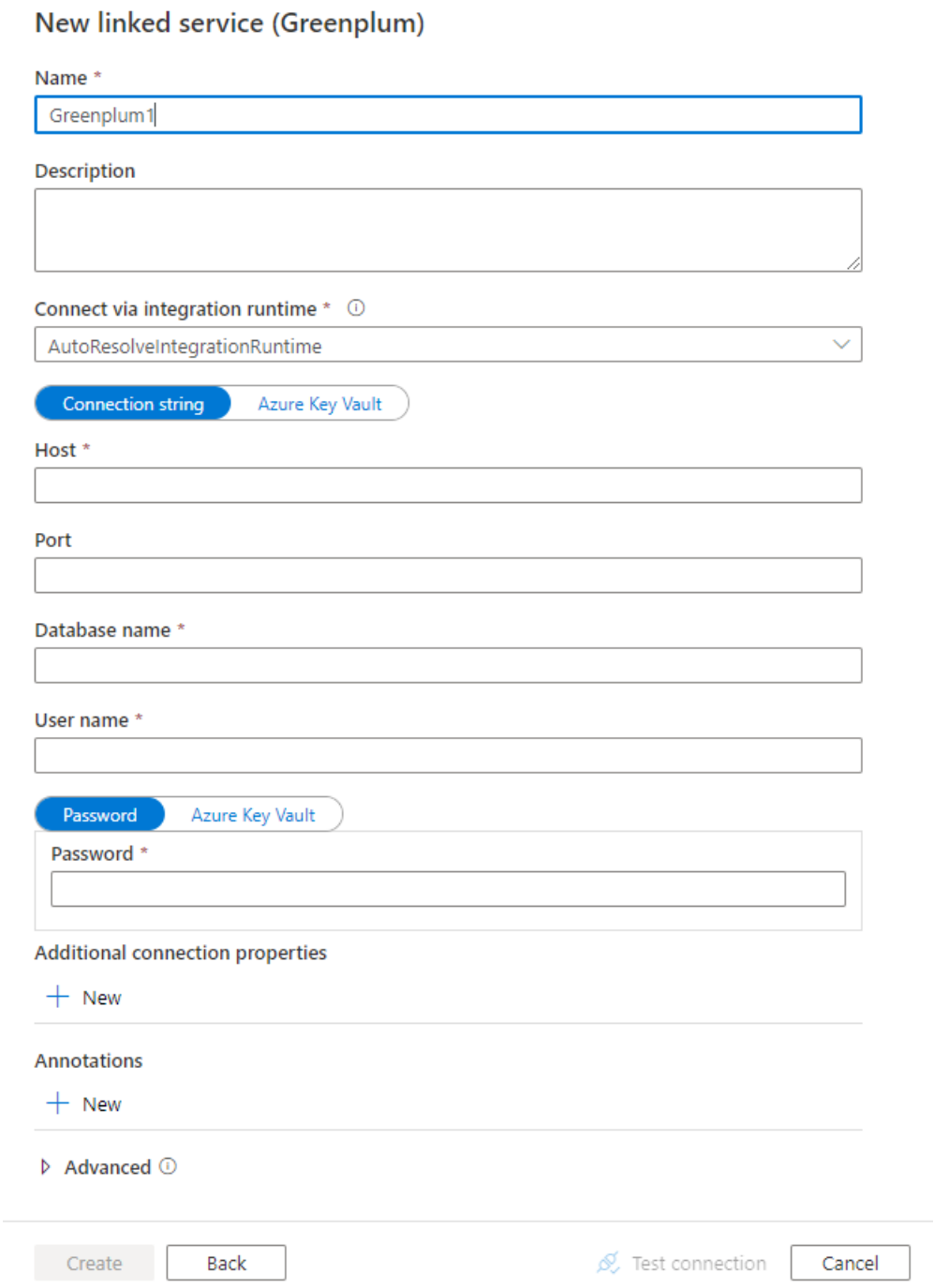
   - Azure Data Factory
   - Azure Synapse



2. Search for Greenplum and select the Greenplum connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Greenplum connector.

Linked service properties

The following properties are supported for Greenplum linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Greenplum**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	An ODBC connection string to connect to Greenplum. You can also put password in Azure Key Vault and pull the `pwd` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "GreenplumLinkedService",
    "properties": {
        "type": "Greenplum",
        "typeProperties": {
            "connectionString": "HOST=<server>;PORT=<port>;DB=<database>;UID=<user name>;PWD=<password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "GreenplumLinkedService",
    "properties": {
        "type": "Greenplum",
        "typeProperties": {
            "connectionString": "HOST=<server>;PORT=<port>;DB=<database>;UID=<user name>;",
            "pwd": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section

provides a list of properties supported by Greenplum dataset.

To copy data from Greenplum, set the type property of the dataset to **GreenplumTable**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **GreenplumTable**	Yes
schema	Name of the schema.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table with schema. This property is supported for backward compatibility. Use `schema` and `table` for new workload.	No (if "query" in activity source is specified)

Example

```
{
    "name": "GreenplumDataset",
    "properties": {
        "type": "GreenplumTable",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Greenplum linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Greenplum source.

GreenplumSource as source

To copy data from Greenplum, set the source type in the copy activity to **GreenplumSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **GreenplumSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromGreenplum",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Greenplum input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "GreenplumSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from HBase using Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from HBase. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This HBase connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell

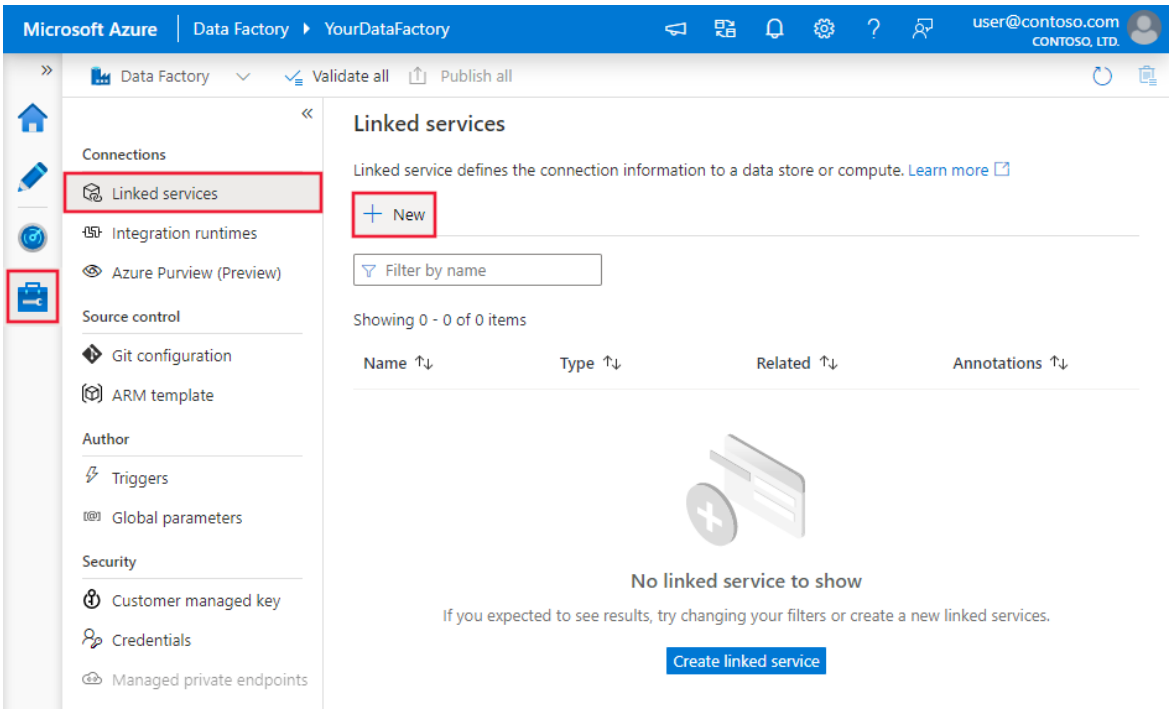
- The REST API
- The Azure Resource Manager template

Create a linked service to Hbase using UI

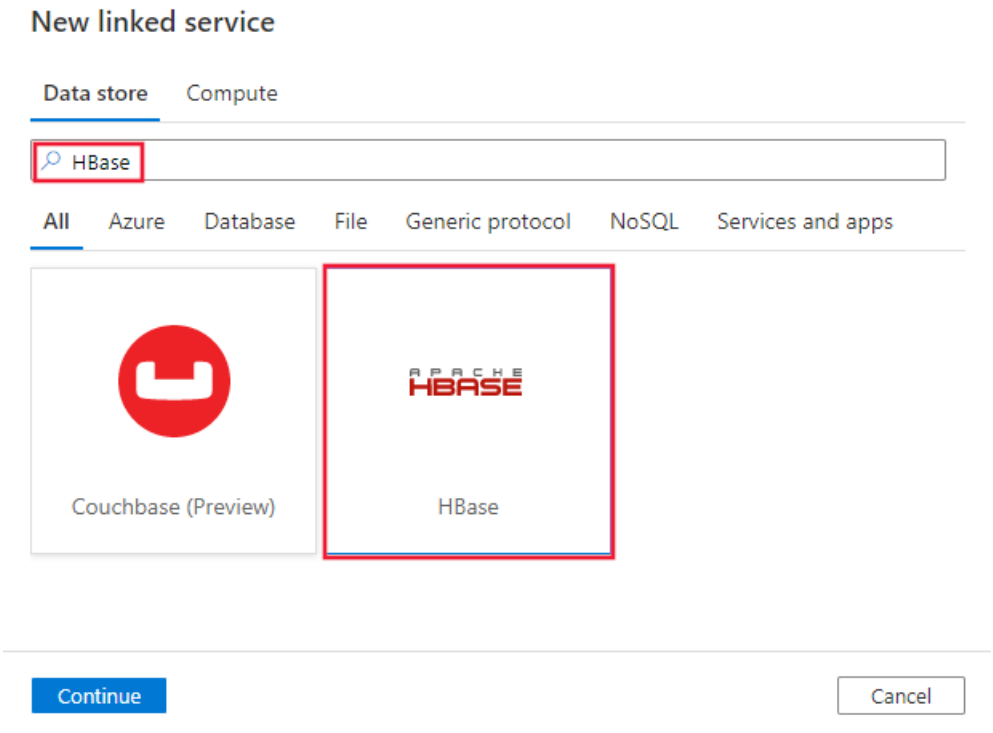
Use the following steps to create a linked service to Hbase in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

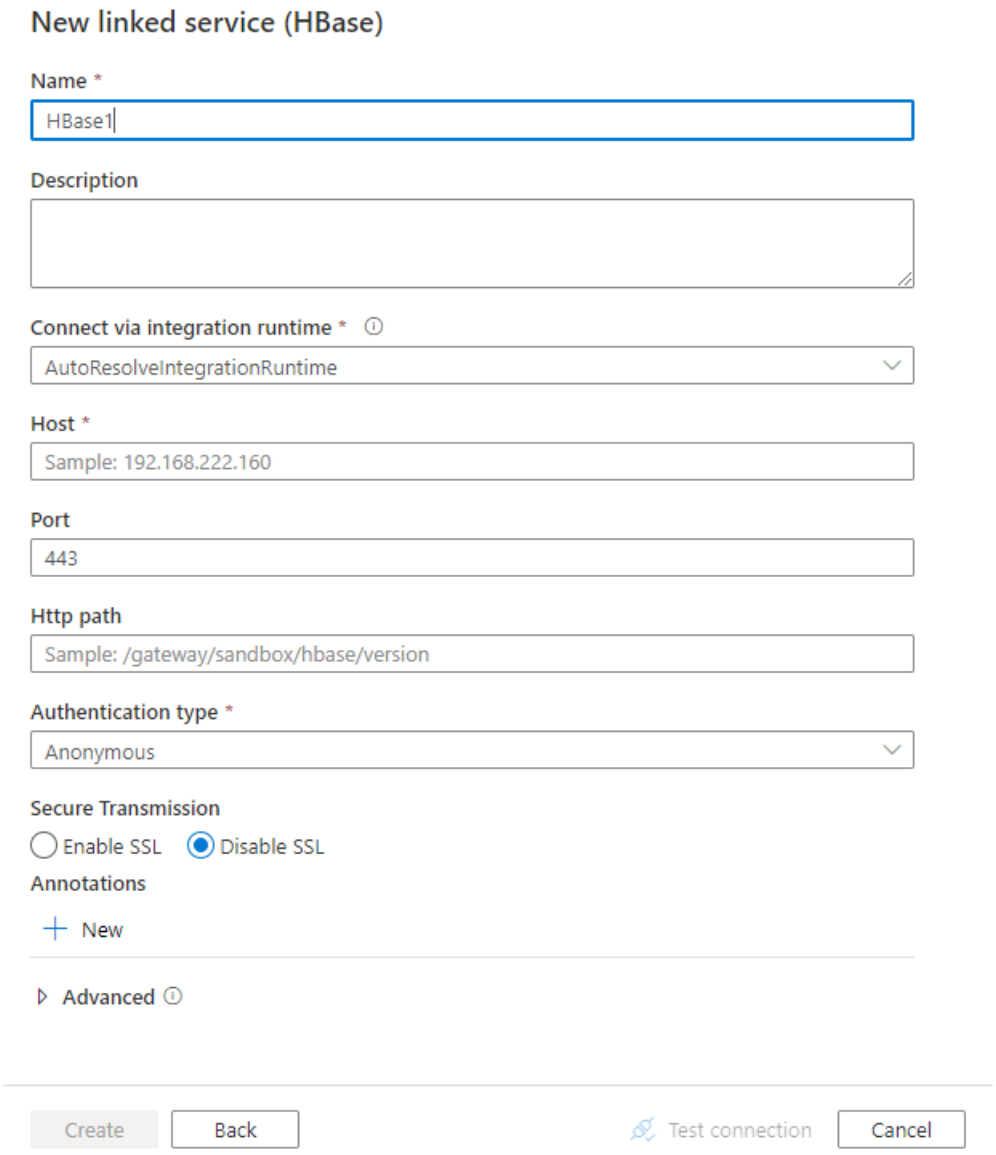
   - Azure Data Factory
   - Azure Synapse



2. Search for Hbase and select the Hbase connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to HBase connector.

Linked service properties

The following properties are supported for HBase linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **HBase**	Yes
host	The IP address or host name of the HBase server. (i.e. `[clustername].azurehdinsight.net`, `192.168.222.160`)	Yes

PROPERTY	DESCRIPTION	REQUIRED
port	The TCP port that the HBase instance uses to listen for client connections. The default value is 9090. If you connect to Azure HDInsights, specify port as 443.	No
httpPath	The partial URL corresponding to the HBase server, e.g. `/hbaserest0` when using HDInsights cluster.	No
authenticationType	The authentication mechanism to use to connect to the HBase server. Allowed values are: **Anonymous**, **Basic**	Yes
username	The user name used to connect to the HBase instance.	No
password	The password corresponding to the user name. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
enableSsl	Specifies whether the connections to the server are encrypted using TLS. The default value is false.	No
trustedCertPath	The full path of the .pem file containing trusted CA certificates for verifying the server when connecting over TLS. This property can only be set when using TLS on self-hosted IR. The default value is the cacerts.pem file installed with the IR.	No
allowHostNameCNMismatch	Specifies whether to require a CA-issued TLS/SSL certificate name to match the host name of the server when connecting over TLS. The default value is false.	No
allowSelfSignedServerCert	Specifies whether to allow self-signed certificates from the server. The default value is false.	No
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

NOTE

If your cluster doesn't support sticky session e.g. HDInsight, explicitly add node index at the end of the http path setting, e.g. specify `/hbaserest0` instead of `/hbaserest`.

Example for HDInsights HBase:

```
{
    "name": "HBaseLinkedService",
    "properties": {
        "type": "HBase",
        "typeProperties": {
            "host" : "<cluster name>.azurehdinsight.net",
            "port" : "443",
            "httpPath" : "/hbaserest0",
            "authenticationType" : "Basic",
            "username" : "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            },
            "enableSsl" : true
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example for generic HBase:

```
{
    "name": "HBaseLinkedService",
    "properties": {
        "type": "HBase",
        "typeProperties": {
            "host" : "<host e.g. 192.168.222.160>",
            "port" : "<port>",
            "httpPath" : "<e.g. /gateway/sandbox/hbase/version>",
            "authenticationType" : "Basic",
            "username" : "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            },
            "enableSsl" : true,
            "trustedCertPath" : "<trustedCertPath>",
            "allowHostNameCNMismatch" : true,
            "allowSelfSignedServerCert" : true
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section

provides a list of properties supported by HBase dataset.

To copy data from HBase, set the type property of the dataset to **HBaseObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **HBaseObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "HBaseDataset",
    "properties": {
        "type": "HBaseObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<HBase linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by HBase source.

HBaseSource as source

To copy data from HBase, set the source type in the copy activity to **HBaseSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **HBaseSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromHBase",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<HBase input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "HBaseSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from the HDFS server using Azure Data Factory or Synapse Analytics

8/19/2022 • 20 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to copy data from the Hadoop Distributed File System (HDFS) server. To learn more, read the introductory articles for Azure Data Factory and Synapse Analytics.

Supported capabilities

This HDFS connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②
Delete activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

Specifically, the HDFS connector supports:

- Copying files by using *Windows* (Kerberos) or *Anonymous* authentication.
- Copying files by using the *webhdfs* protocol or *built-in DistCp* support.
- Copying files as is or by parsing or generating files with the supported file formats and compression codecs.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

> **NOTE**
> Make sure that the integration runtime can access *all* the [name node server]:[name node port] and [data node servers]:[data node port] of the Hadoop cluster. The default [name node port] is 50070, and the default [data node port] is 50075.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

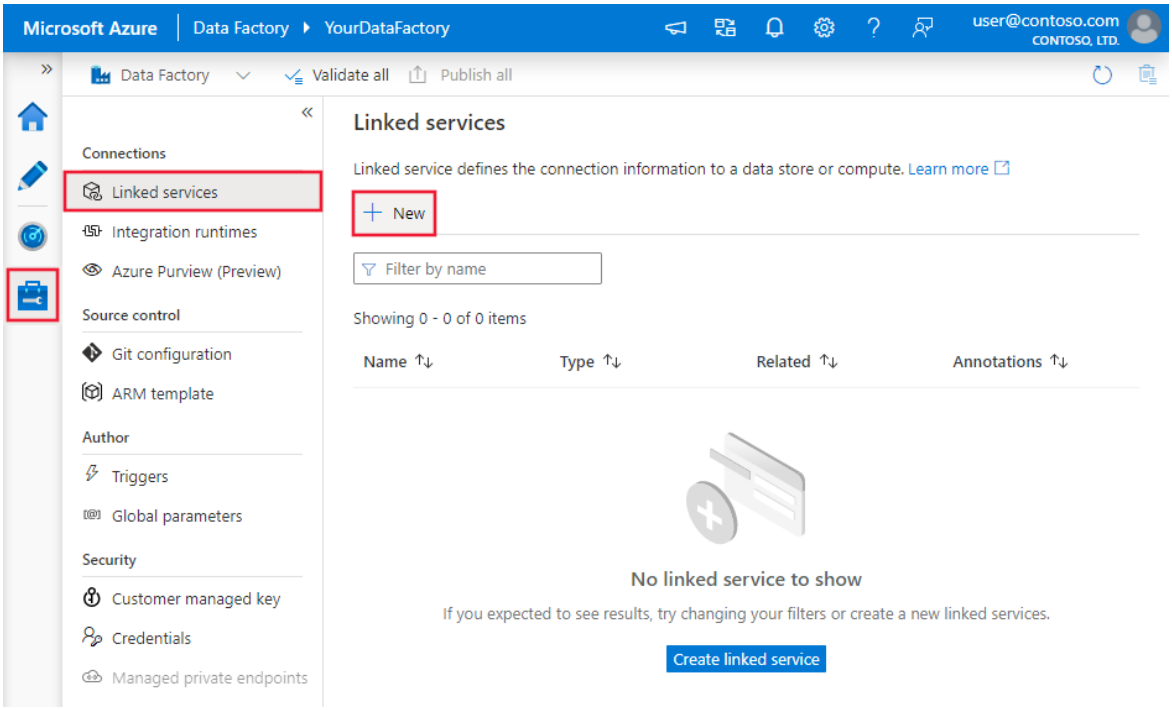
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to HDFS using UI

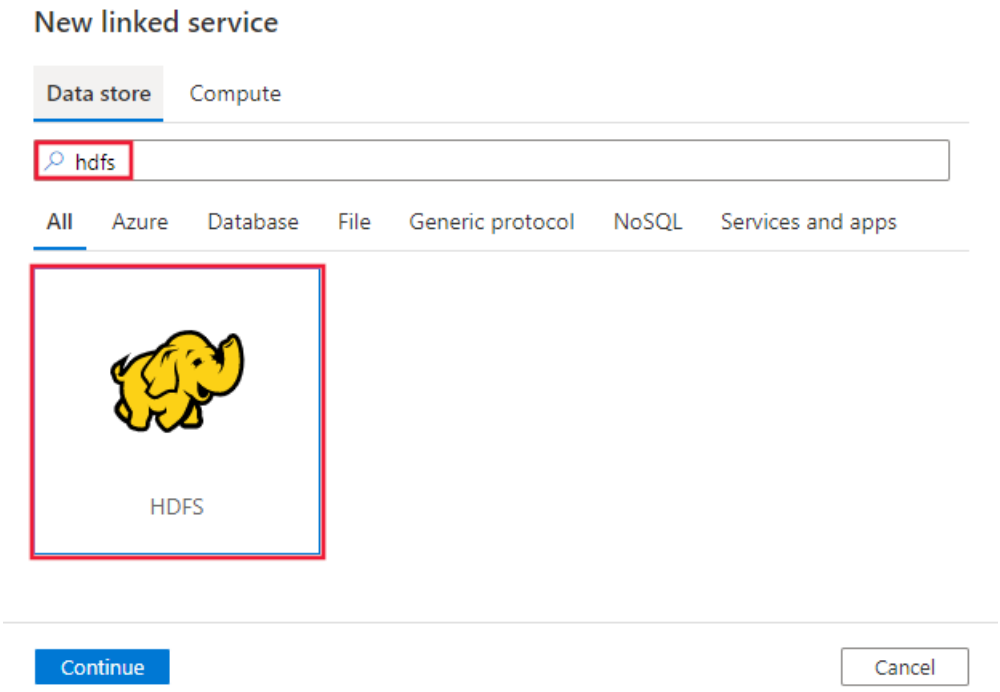
Use the following steps to create a linked service to HDFS in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

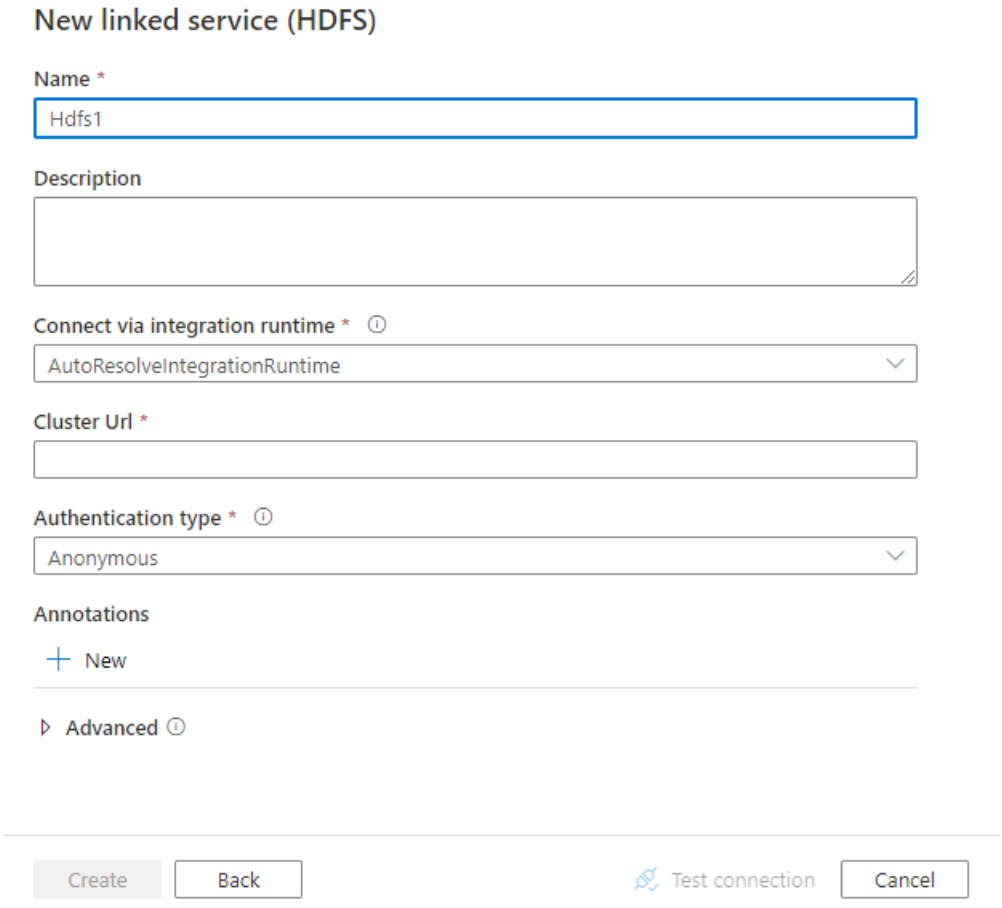
   - Azure Data Factory
   - Azure Synapse



2. Search for HDFS and select the HDFS connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to HDFS.

Linked service properties

The following properties are supported for the HDFS linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The *type* property must be set to *Hdfs*.	Yes
url	The URL to the HDFS	Yes
authenticationType	The allowed values are *Anonymous* or *Windows*. To set up your on-premises environment, see the Use Kerberos authentication for the HDFS connector section.	Yes
userName	The username for Windows authentication. For Kerberos authentication, specify **<username>@<domain>.com**.	Yes (for Windows authentication)
password	The password for Windows authentication. Mark this field as a SecureString to store it securely, or reference a secret stored in an Azure key vault.	Yes (for Windows Authentication)
connectVia	The integration runtime to be used to connect to the data store. To learn more, see the Prerequisites section. If the integration runtime isn't specified, the service uses the default Azure Integration Runtime.	No

Example: using Anonymous authentication

```
{
    "name": "HDFSLinkedService",
    "properties": {
        "type": "Hdfs",
        "typeProperties": {
            "url" : "http://<machine>:50070/webhdfs/v1/",
            "authenticationType": "Anonymous",
            "userName": "hadoop"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: using Windows authentication

```
{
    "name": "HDFSLinkedService",
    "properties": {
        "type": "Hdfs",
        "typeProperties": {
            "url" : "http://<machine>:50070/webhdfs/v1/",
            "authenticationType": "Windows",
            "userName": "<username>@<domain>.com (for Kerberos auth)",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties that are available for defining datasets, see Datasets.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for HDFS under `location` settings in the format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The *type* property under `location` in the dataset must be set to *HdfsLocation*.	Yes
folderPath	The path to the folder. If you want to use a wildcard to filter the folder, skip this setting and specify the path in activity source settings.	No
fileName	The file name under the specified folderPath. If you want to use a wildcard to filter files, skip this setting and specify the file name in activity source settings.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<HDFS linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "HdfsLocation",
                "folderPath": "root/folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties that are available for defining activities, see Pipelines and activities. This section provides a list of properties that are supported by the HDFS source.

HDFS as source

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for HDFS under `storeSettings` settings in the format-based Copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The *type* property under `storeSettings` must be set to **HdfsReadSettings**.	Yes
Locate the files to copy		
OPTION 1: static path	Copy from the folder or file path that's specified in the dataset. If you want to copy all files from a folder, additionally specify `wildcardFileName` as `*`.	

PROPERTY	DESCRIPTION	REQUIRED
OPTION 2: wildcard - wildcardFolderPath	The folder path with wildcard characters to filter source folders. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your actual folder name has a wildcard or this escape character inside. For more examples, see Folder and file filter examples.	No
OPTION 2: wildcard - wildcardFileName	The file name with wildcard characters under the specified folderPath/wildcardFolderPath to filter source files. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character); use `^` to escape if your actual file name has a wildcard or this escape character inside. For more examples, see Folder and file filter examples.	Yes
OPTION 3: a list of files - fileListPath	Indicates to copy a specified file set. Point to a text file that includes a list of files you want to copy (one file per line, with the relative path to the path configured in the dataset). When you use this option, do not specify file name in the dataset. For more examples, see File list examples.	No
Additional settings		
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. When `recursive` is set to *true* and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are *true* (default) and *false*. This property doesn't apply when you configure `fileListPath`.	No
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	Files are filtered based on the attribute *Last Modified*. The files are selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of *2018-12-01T05:00:00Z*. The properties can be NULL, which means that no file attribute filter is applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is NULL, it means that the files whose last modified attribute is greater than or equal to the datetime value are selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is NULL, it means that the files whose last modified attribute is less than the datetime value are selected. This property doesn't apply when you configure `fileListPath`.	No
modifiedDatetimeEnd	Same as above.	
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No
DistCp settings		
distcpSettings	The property group to use when you use HDFS DistCp.	No
resourceManagerEndpoint	The YARN (Yet Another Resource Negotiator) endpoint	Yes, if using DistCp
tempScriptPath	A folder path that's used to store the temp DistCp command script. The script file is generated and will be removed after the Copy job is finished.	Yes, if using DistCp
distcpOptions	Additional options provided to DistCp command.	No

Example:

```
"activities":[
    {
        "name": "CopyFromHDFS",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "HdfsReadSettings",
                    "recursive": true,
                    "distcpSettings": {
                        "resourceManagerEndpoint": "resourcemanagerendpoint:8088",
                        "tempScriptPath": "/usr/hadoop/tempscript",
                        "distcpOptions": "-m 100"
                    }
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Folder and file filter examples

This section describes the resulting behavior if you use a wildcard filter with the folder path and file name.

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`Folder*`	(empty, use default)	false	FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`Folder*`	(empty, use default)	true	FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv
`Folder*`	`*.csv`	false	FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	`*.csv`	true	FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This section describes the behavior that results from using a file list path in the Copy activity source. It assumes that you have the following source folder structure and want to copy the files that are in bold type:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	CONFIGURATION
root FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In the dataset:** - Folder path: `root/FolderA` **In the Copy activity source:** - File list path: `root/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy (one file per line, with the relative path to the path configured in the dataset).

Use DistCp to copy data from HDFS

DistCp is a Hadoop native command-line tool for doing a distributed copy in a Hadoop cluster. When you run a command in DistCp, it first lists all the files to be copied and then creates several Map jobs in the Hadoop cluster. Each Map job does a binary copy from the source to the sink.

The Copy activity supports using DistCp to copy files as is into Azure Blob storage (including staged copy) or an Azure data lake store. In this case, DistCp can take advantage of your cluster's power instead of running on the self-hosted integration runtime. Using DistCp provides better copy throughput, especially if your cluster is very powerful. Based on the configuration, the Copy activity automatically constructs a DistCp command, submits it to your Hadoop cluster, and monitors the copy status.

Prerequisites

To use DistCp to copy files as is from HDFS to Azure Blob storage (including staged copy) or the Azure data lake store, make sure that your Hadoop cluster meets the following requirements:

- The MapReduce and YARN services are enabled.
- YARN version is 2.5 or later.
- The HDFS server is integrated with your target data store: **Azure Blob storage** or **Azure Data Lake Store (ADLS Gen1)**:
  - Azure Blob FileSystem is natively supported since Hadoop 2.7. You need only to specify the JAR path in the Hadoop environment configuration.
  - Azure Data Lake Store FileSystem is packaged starting from Hadoop 3.0.0-alpha1. If your Hadoop cluster version is earlier than that version, you need to manually import Azure Data Lake Store-related JAR packages (azure-datalake-store.jar) into the cluster from here, and specify the JAR file path in the Hadoop environment configuration.
- Prepare a temp folder in HDFS. This temp folder is used to store a DistCp shell script, so it will occupy KB-level space.
- Make sure that the user account that's provided in the HDFS linked service has permission to:
  - Submit an application in YARN.
  - Create a subfolder and read/write files under the temp folder.

Configurations

For DistCp-related configurations and examples, go to the HDFS as source section.

Use Kerberos authentication for the HDFS connector

There are two options for setting up the on-premises environment to use Kerberos authentication for the HDFS connector. You can choose the one that better fits your situation.

- Option 1: Join a self-hosted integration runtime machine in the Kerberos realm
- Option 2: Enable mutual trust between the Windows domain and the Kerberos realm

For either option, make sure you turn on webhdfs for Hadoop cluster:

1. Create the HTTP principal and keytab for webhdfs.

   > **IMPORTANT**
   > The HTTP Kerberos principal must start with "**HTTP/**" according to Kerberos HTTP SPNEGO specification. Learn more from here.

   ```
   Kadmin> addprinc -randkey HTTP/<namenode hostname>@<REALM.COM>
   Kadmin> ktadd -k /etc/security/keytab/spnego.service.keytab HTTP/<namenode hostname>@<REALM.COM>
   ```

2. HDFS configuration options: add the following three properties in `hdfs-site.xml`.

```
<property>
    <name>dfs.webhdfs.enabled</name>
    <value>true</value>
</property>
<property>
    <name>dfs.web.authentication.kerberos.principal</name>
    <value>HTTP/_HOST@<REALM.COM></value>
</property>
<property>
    <name>dfs.web.authentication.kerberos.keytab</name>
    <value>/etc/security/keytab/spnego.service.keytab</value>
</property>
```

Option 1: Join a self-hosted integration runtime machine in the Kerberos realm

Requirements

- The self-hosted integration runtime machine needs to join the Kerberos realm and can't join any Windows domain.

How to configure

On the KDC server:

Create a principal, and specify the password.

> **IMPORTANT**
> The username should not contain the hostname.

```
Kadmin> addprinc <username>@<REALM.COM>
```

On the self-hosted integration runtime machine:

1. Run the Ksetup utility to configure the Kerberos Key Distribution Center (KDC) server and realm.

   The machine must be configured as a member of a workgroup, because a Kerberos realm is different from a Windows domain. You can achieve this configuration by setting the Kerberos realm and adding a KDC server by running the following commands. Replace *REALM.COM* with your own realm name.

   ```
   C:> Ksetup /setdomain REALM.COM
   C:> Ksetup /addkdc REALM.COM <your_kdc_server_address>
   ```

   After you run these commands, restart the machine.

2. Verify the configuration with the `Ksetup` command. The output should be like:

   ```
   C:> Ksetup
   default realm = REALM.COM (external)
   REALM.com:
       kdc = <your_kdc_server_address>
   ```

In your data factory or Synapse workspace:

- Configure the HDFS connector by using Windows authentication together with your Kerberos principal name and password to connect to the HDFS data source. For configuration details, check the HDFS linked service properties section.

Option 2: Enable mutual trust between the Windows domain and the Kerberos realm

Requirements

- The self-hosted integration runtime machine must join a Windows domain.
- You need permission to update the domain controller's settings.

How to configure

> **NOTE**
> Replace REALM.COM and AD.COM in the following tutorial with your own realm name and domain controller.

On the KDC server:

1. Edit the KDC configuration in the *krb5.conf* file to let KDC trust the Windows domain by referring to the following configuration template. By default, the configuration is located at */etc/krb5.conf*.

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 default_realm = REALM.COM
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true

[realms]
 REALM.COM = {
  kdc = node.REALM.COM
  admin_server = node.REALM.COM
 }
AD.COM = {
 kdc = windc.ad.com
 admin_server = windc.ad.com
}

[domain_realm]
 .REALM.COM = REALM.COM
 REALM.COM = REALM.COM
 .ad.com = AD.COM
 ad.com = AD.COM

[capaths]
 AD.COM = {
  REALM.COM = .
 }
```

   After you configure the file, restart the KDC service.

2. Prepare a principal named *krbtgt/REALM.COM@AD.COM* in the KDC server with the following command:

```
Kadmin> addprinc krbtgt/REALM.COM@AD.COM
```

3. In the *hadoop.security.auth_to_local* HDFS service configuration file, add `RULE:[1:$1@$0](.*\@AD.COM)s/\@.*//`.

On the domain controller:

1. Run the following `Ksetup` commands to add a realm entry:

```
C:> Ksetup /addkdc REALM.COM <your_kdc_server_address>
C:> ksetup /addhosttorealmmap HDFS-service-FQDN REALM.COM
```

2. Establish trust from the Windows domain to the Kerberos realm. [password] is the password for the principal *krbtgt/REALM.COM@AD.COM*.

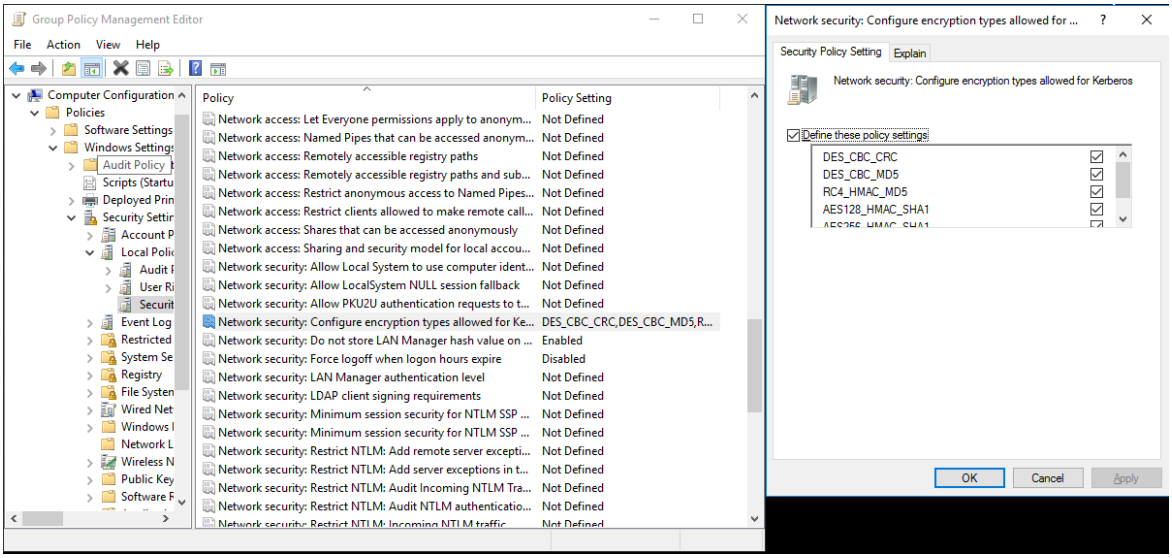
```
C:> netdom trust REALM.COM /Domain: AD.COM /add /realm /password:[password]
```

3. Select the encryption algorithm that's used in Kerberos.

   a. Select **Server Manager** > **Group Policy Management** > **Domain** > **Group Policy Objects** > **Default or Active Domain Policy**, and then select **Edit**.

   b. On the **Group Policy Management Editor** pane, select **Computer Configuration** > **Policies** > **Windows Settings** > **Security Settings** > **Local Policies** > **Security Options**, and then configure **Network security: Configure Encryption types allowed for Kerberos**.

   c. Select the encryption algorithm you want to use when you connect to the KDC server. You can select all the options.



   d. Use the `Ksetup` command to specify the encryption algorithm to be used on the specified realm.

```
C:> ksetup /SetEncTypeAttr REALM.COM DES-CBC-CRC DES-CBC-MD5 RC4-HMAC-MD5 AES128-CTS-HMAC-SHA1-96
AES256-CTS-HMAC-SHA1-96
```

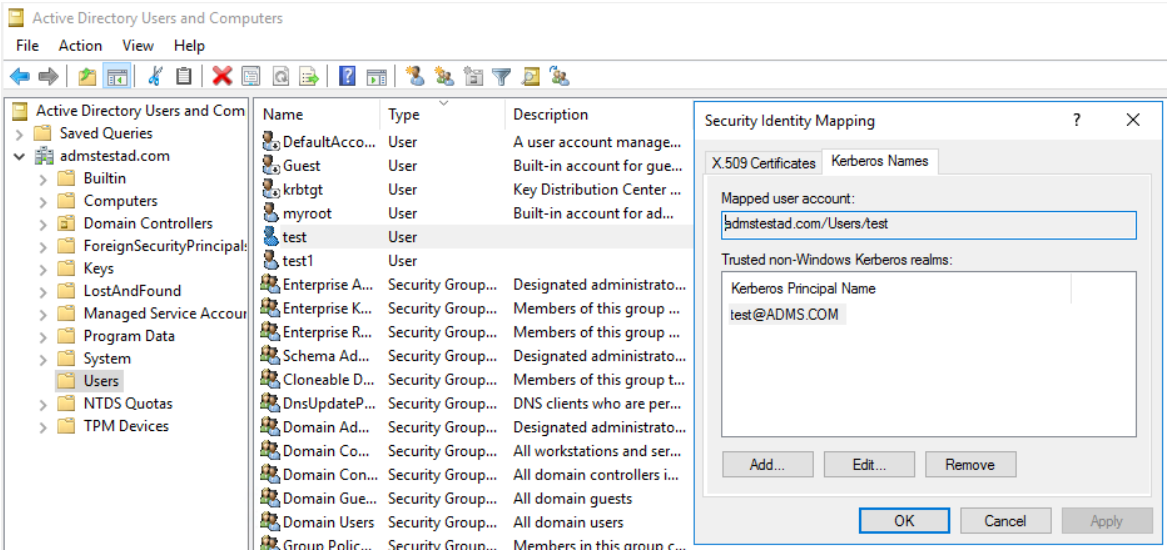
4. Create the mapping between the domain account and the Kerberos principal, so that you can use the Kerberos principal in the Windows domain.

   a. Select **Administrative tools** > **Active Directory Users and Computers**.

   b. Configure advanced features by selecting **View** > **Advanced Features**.

   c. On the **Advanced Features** pane, right-click the account to which you want to create mappings and, on the **Name Mappings** pane, select the **Kerberos Names** tab.

   d. Add a principal from the realm.



On the self-hosted integration runtime machine:

- Run the following `Ksetup` commands to add a realm entry.

```
C:> Ksetup /addkdc REALM.COM <your_kdc_server_address>
C:> ksetup /addhosttorealmmap HDFS-service-FQDN REALM.COM
```

In your data factory or Synapse workspace:

- Configure the HDFS connector by using Windows authentication together with either your domain account or Kerberos principal to connect to the HDFS data source. For configuration details, see the HDFS linked service properties section.

Lookup activity properties

For information about Lookup activity properties, see Lookup activity.

Delete activity properties

For information about Delete activity properties, see Delete activity.

Legacy models

> **NOTE**
> The following models are still supported as is for backward compatibility. We recommend that you use the previously discussed new model, because the authoring UI has switched to generating the new model.

Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED
type	The *type* property of the dataset must be set to *FileShare*	Yes

PROPERTY	DESCRIPTION	REQUIRED
folderPath	The path to the folder. A wildcard filter is supported. Allowed wildcards are `*` (matches zero or more characters) and `?` (matches zero or a single character); use `^` to escape if your actual file name has a wildcard or this escape character inside. Examples: rootfolder/subfolder/, see more examples in Folder and file filter examples.	Yes
fileName	The name or wildcard filter for the files under the specified "folderPath". If you don't specify a value for this property, the dataset points to all files in the folder. For filter, allowed wildcards are `*` (matches zero or more characters) and `?` (matches zero or a single character). - Example 1: `"fileName": "*.csv"` - Example 2: `"fileName": "???20180427.txt"` Use `^` to escape if your actual folder name has a wildcard or this escape character inside.	No
modifiedDatetimeStart	Files are filtered based on the attribute *Last Modified*. The files are selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format *2018-12-01T05:00:00Z*. Be aware that the overall performance of data movement will be affected by enabling this setting when you want to apply a file filter to large numbers of files. The properties can be NULL, which means that no file attribute filter is applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is NULL, it means that the files whose last modified attribute is greater than or equal to the datetime value are selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is NULL, it means that the files whose last modified attribute is less than the datetime value are selected.	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeEnd	Files are filtered based on the attribute *Last Modified*. The files are selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format *2018-12-01T05:00:00Z*. Be aware that the overall performance of data movement will be affected by enabling this setting when you want to apply a file filter to large numbers of files. The properties can be NULL, which means that no file attribute filter is applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is NULL, it means that the files whose last modified attribute is greater than or equal to the datetime value are selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is NULL, it means that the files whose last modified attribute is less than the datetime value are selected.	No
format	If you want to copy files as is between file-based stores (binary copy), skip the format section in both the input and output dataset definitions. If you want to parse files with a specific format, the following file format types are supported: *TextFormat*, *JsonFormat*, *AvroFormat*, *OrcFormat*, *ParquetFormat*. Set the *type* property under format to one of these values. For more information, see the Text format, JSON format, Avro format, ORC format, and Parquet format sections.	No (only for binary copy scenario)
compression	Specify the type and level of compression for the data. For more information, see Supported file formats and compression codecs. Supported types are: *Gzip*, *Deflate*, *Bzip2*, and *ZipDeflate*. Supported levels are: *Optimal* and *Fastest*.	No

TIP

To copy all files under a folder, specify **folderPath** only.
To copy a single file with a specified name, specify **folderPath** with folder part and **fileName** with file name.
To copy a subset of files under a folder, specify **folderPath** with folder part and **fileName** with wildcard filter.

Example:

```
{
    "name": "HDFSDataset",
    "properties": {
        "type": "FileShare",
        "linkedServiceName":{
            "referenceName": "<HDFS linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "folderPath": "folder/subfolder/",
            "fileName": "*",
            "modifiedDatetimeStart": "2018-12-01T05:00:00Z",
            "modifiedDatetimeEnd": "2018-12-01T06:00:00Z",
            "format": {
                "type": "TextFormat",
                "columnDelimiter": ",",
                "rowDelimiter": "\n"
            },
            "compression": {
                "type": "GZip",
                "level": "Optimal"
            }
        }
    }
}
```

Legacy Copy activity source model

PROPERTY	DESCRIPTION	REQUIRED
type	The *type* property of the Copy activity source must be set to *HdfsSource*.	Yes
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. When recursive is set to *true* and the sink is a file-based store, an empty folder or subfolder will not be copied or created at the sink. Allowed values are *true* (default) and *false*.	No
distcpSettings	The property group when you're using HDFS DistCp.	No
resourceManagerEndpoint	The YARN Resource Manager endpoint	Yes, if using DistCp

PROPERTY	DESCRIPTION	REQUIRED
tempScriptPath	A folder path that's used to store the temp DistCp command script. The script file is generated and will be removed after the Copy job is finished.	Yes, if using DistCp
distcpOptions	Additional options are provided to DistCp command.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example: HDFS source in Copy activity using DistCp

```
"source": {
    "type": "HdfsSource",
    "distcpSettings": {
        "resourceManagerEndpoint": "resourcemanagerendpoint:8088",
        "tempScriptPath": "/usr/hadoop/tempscript",
        "distcpOptions": "-m 100"
    }
}
```

Next steps

For a list of data stores that are supported as sources and sinks by the Copy activity, see supported data stores.

Copy and transform data from Hive using Azure Data Factory

8/19/2022 • 7 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Hive. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Hive connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Mapping data flow (source/-)	①
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK

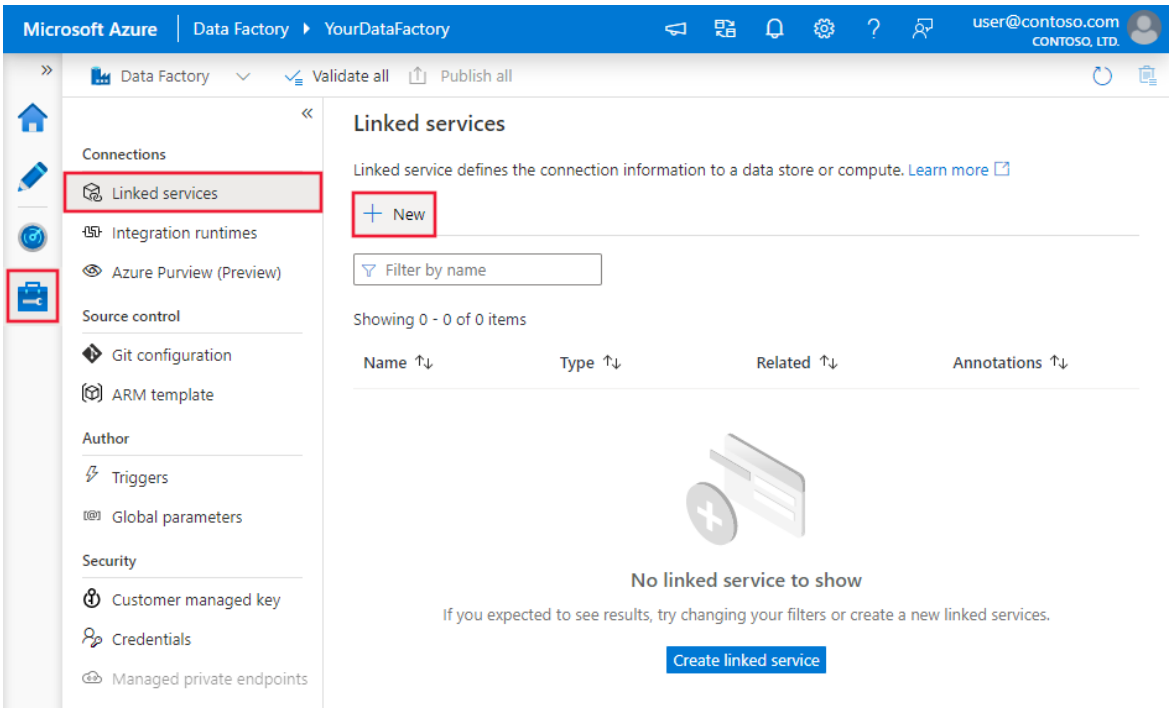
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Hive using UI

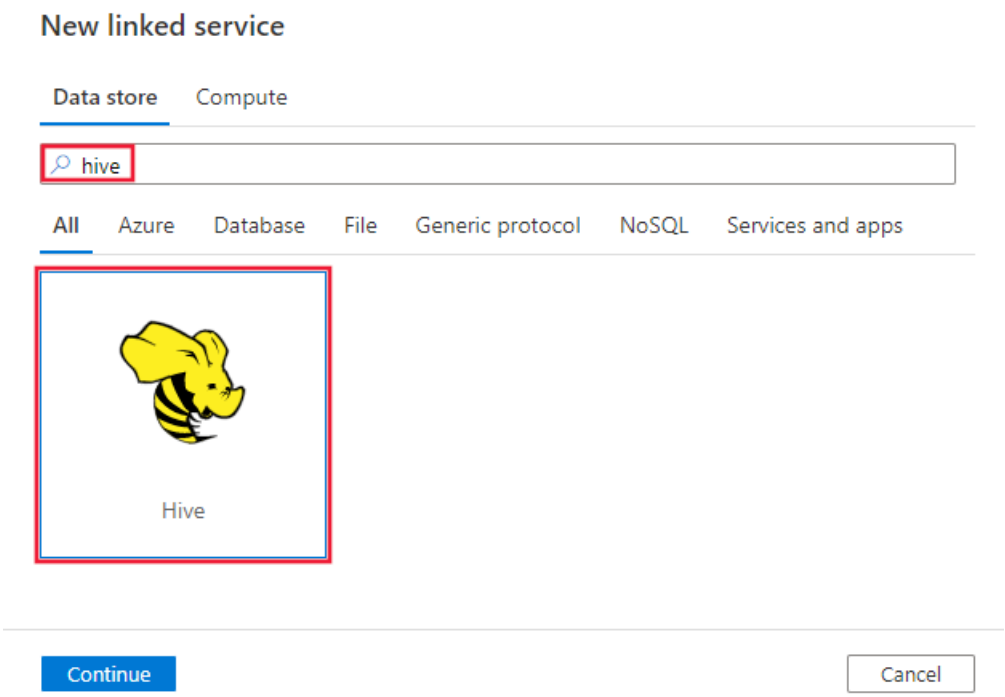
Use the following steps to create a linked service to Hive in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

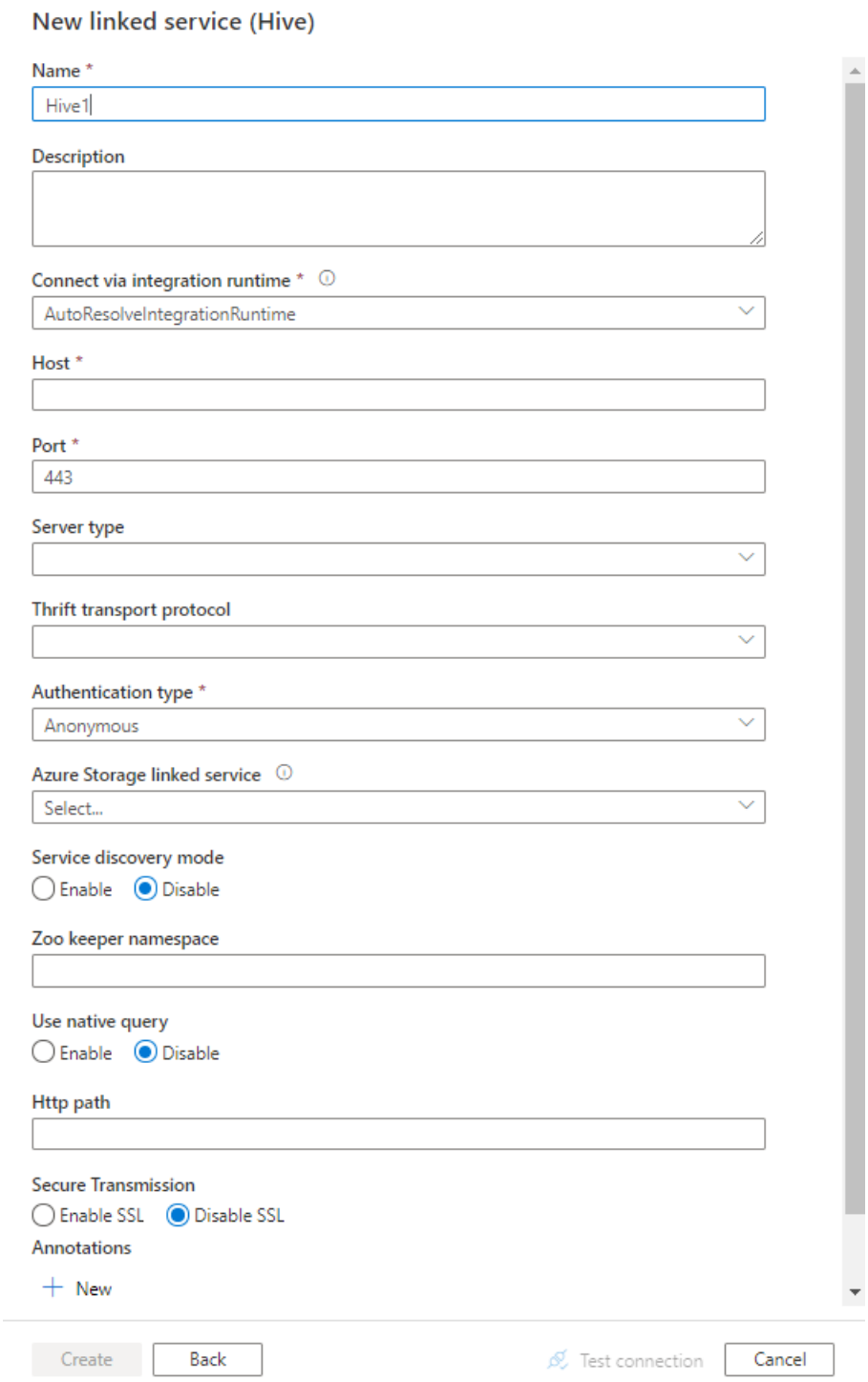
   - Azure Data Factory
   - Azure Synapse



2. Search for Hive and select the Hive connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Hive connector.

Linked service properties

The following properties are supported for Hive linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Hive**	Yes

PROPERTY	DESCRIPTION	REQUIRED
host	IP address or host name of the Hive server, separated by ';' for multiple hosts (only when serviceDiscoveryMode is enabled).	Yes
port	The TCP port that the Hive server uses to listen for client connections. If you connect to Azure HDInsight, specify port as 443.	Yes
serverType	The type of Hive server. Allowed values are: **HiveServer1**, **HiveServer2**, **HiveThriftServer**	No
thriftTransportProtocol	The transport protocol to use in the Thrift layer. Allowed values are: **Binary**, **SASL**, **HTTP**	No
authenticationType	The authentication method used to access the Hive server. Allowed values are: **Anonymous**, **Username**, **UsernameAndPassword**, **WindowsAzureHDInsightService**. Kerberos authentication is not supported now.	Yes
serviceDiscoveryMode	true to indicate using the ZooKeeper service, false not.	No
zooKeeperNameSpace	The namespace on ZooKeeper under which Hive Server 2 nodes are added.	No
useNativeQuery	Specifies whether the driver uses native HiveQL queries, or converts them into an equivalent form in HiveQL.	No
username	The user name that you use to access Hive Server.	No
password	The password corresponding to the user. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
httpPath	The partial URL corresponding to the Hive server.	No
enableSsl	Specifies whether the connections to the server are encrypted using TLS. The default value is false.	No

PROPERTY	DESCRIPTION	REQUIRED
trustedCertPath	The full path of the .pem file containing trusted CA certificates for verifying the server when connecting over TLS. This property can only be set when using TLS on self-hosted IR. The default value is the cacerts.pem file installed with the IR.	No
useSystemTrustStore	Specifies whether to use a CA certificate from the system trust store or from a specified PEM file. The default value is false.	No
allowHostNameCNMismatch	Specifies whether to require a CA-issued TLS/SSL certificate name to match the host name of the server when connecting over TLS. The default value is false.	No
allowSelfSignedServerCert	Specifies whether to allow self-signed certificates from the server. The default value is false.	No
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No
storageReference	A reference to the linked service of the storage account used for staging data in mapping data flow. This is required only when using the Hive linked service in mapping data flow	No

Example:

```
{
    "name": "HiveLinkedService",
    "properties": {
        "type": "Hive",
        "typeProperties": {
            "host" : "<cluster>.azurehdinsight.net",
            "port" : "<port>",
            "authenticationType" : "WindowsAzureHDInsightService",
            "username" : "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section

provides a list of properties supported by Hive dataset.

To copy data from Hive, set the type property of the dataset to **HiveObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **HiveObject**	Yes
schema	Name of the schema.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table including schema part. This property is supported for backward compatibility. For new workload, use `schema` and `table`.	No (if "query" in activity source is specified)

Example

```
{
    "name": "HiveDataset",
    "properties": {
        "type": "HiveObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Hive linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Hive source.

HiveSource as source

To copy data from Hive, set the source type in the copy activity to **HiveSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **HiveSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromHive",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Hive input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "HiveSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Mapping data flow properties

The hive connector is supported as an inline dataset source in mapping data flows. Read using a query or directly from a Hive table in HDInsight. Hive data gets staged in a storage account as parquet files before getting transformed as part of a data flow.

Source properties

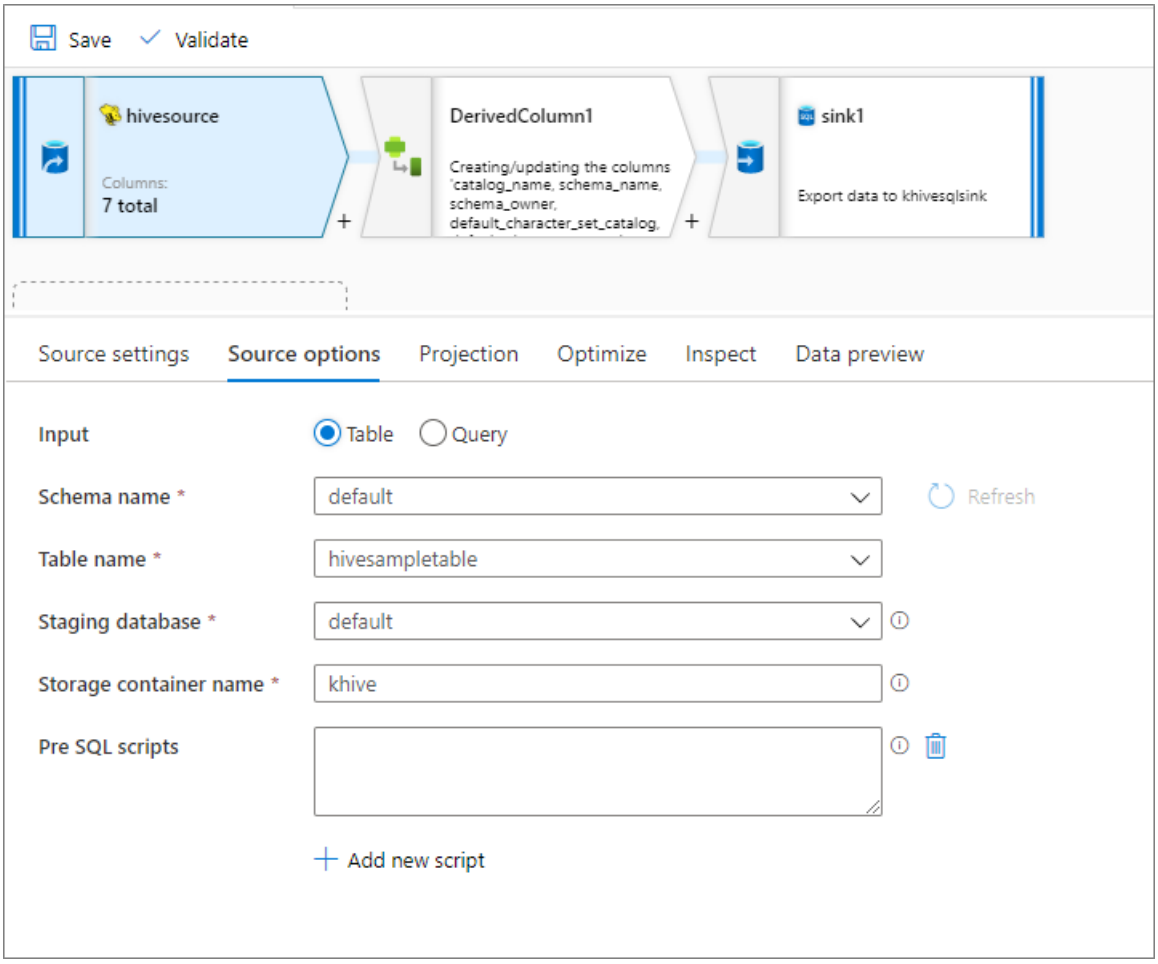
The below table lists the properties supported by a hive source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Store	Store must be `hive`	yes	`hive`	store
Format	Whether you are reading from a table or query	yes	`table` or `query`	format
Schema name	If reading from a table, the schema of the source table	yes, if format is `table`	String	schemaName
Table name	If reading from a table, the table name	yes, if format is `table`	String	tableName
Query	If format is `query`, the source query on the Hive linked service	yes, if format is `query`	String	query

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Staged	Hive table will always be staged.	yes	`true`	staged
Storage Container	Storage container used to stage data before reading from Hive or writing to Hive. The hive cluster must have access to this container.	yes	String	storageContainer
Staging database	The schema/database where the user account specified in the linked service has access to. It is used to create external tables during staging and dropped afterwards	no	`true` or `false`	stagingDatabaseName
Pre SQL Scripts	SQL code to run on the Hive table before reading the data	no	String	preSQLs

Source example

Below is an example of a Hive source configuration:



These settings translate into the following data flow script:

```
source(
    allowSchemaDrift: true,
    validateSchema: false,
    ignoreNoFilesFound: false,
    format: 'table',
    store: 'hive',
    schemaName: 'default',
    tableName: 'hivesampletable',
    staged: true,
    storageContainer: 'khive',
    storageFolderPath: '',
    stagingDatabaseName: 'default') ~> hivesource
```

Known limitations

- Complex types such as arrays, maps, structs, and unions are not supported for read.
- Hive connector only supports Hive tables in Azure HDInsight of version 4.0 or greater (Apache Hive 3.1.0)

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from an HTTP endpoint by using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 10 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory and Azure Synapse to copy data from an HTTP endpoint. The article builds on Copy Activity, which presents a general overview of Copy Activity.

The difference among this HTTP connector, the REST connector and the Web table connector are:

- **REST connector** specifically support copying data from RESTful APIs;
- **HTTP connector** is generic to retrieve data from any HTTP endpoint, e.g. to download file. Before REST connector becomes available, you may happen to use the HTTP connector to copy data from RESTful APIs, which is supported but less functional comparing to REST connector.
- **Web table connector** extracts table content from an HTML webpage.

Supported capabilities

This HTTP connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see Supported data stores.

You can use this HTTP connector to:

- Retrieve data from an HTTP/S endpoint by using the HTTP **GET** or **POST** methods.
- Retrieve data by using one of the following authentications: **Anonymous**, **Basic**, **Digest**, **Windows**, or **ClientCertificate**.
- Copy the HTTP response as-is or parse it by using supported file formats and compression codecs.

> **TIP**
> To test an HTTP request for data retrieval before you configure the HTTP connector, learn about the API specification for header and body requirements. You can use tools like Postman or a web browser to validate.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow

list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

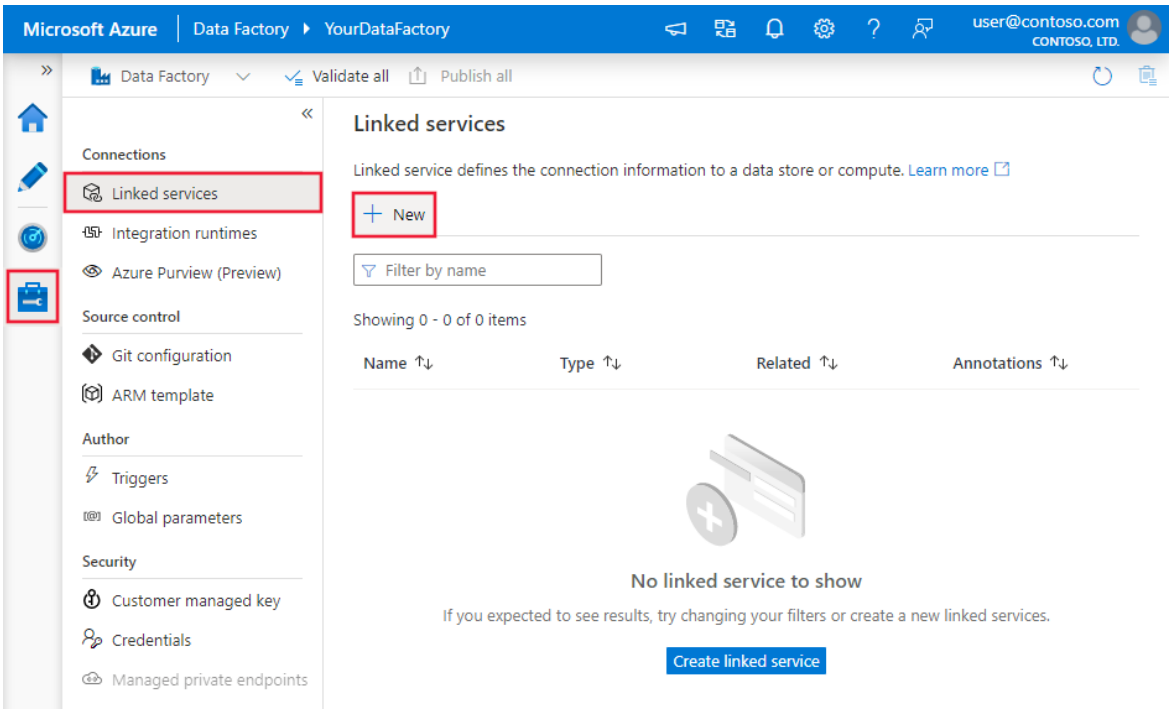
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to an HTTP source using UI

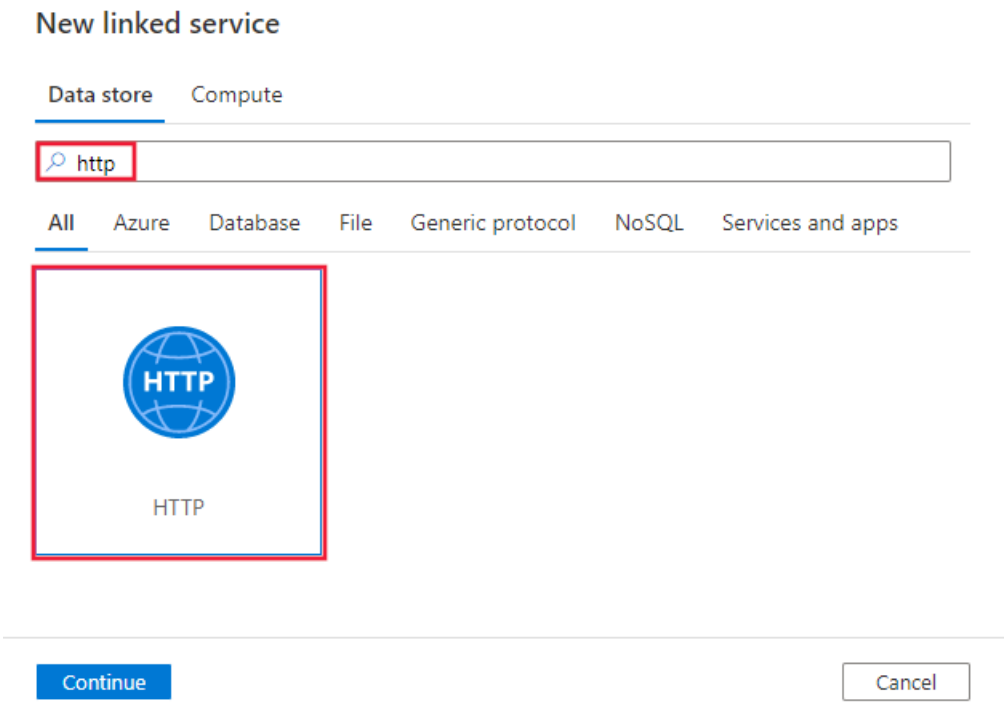
Use the following steps to create a linked service to an HTTP source in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

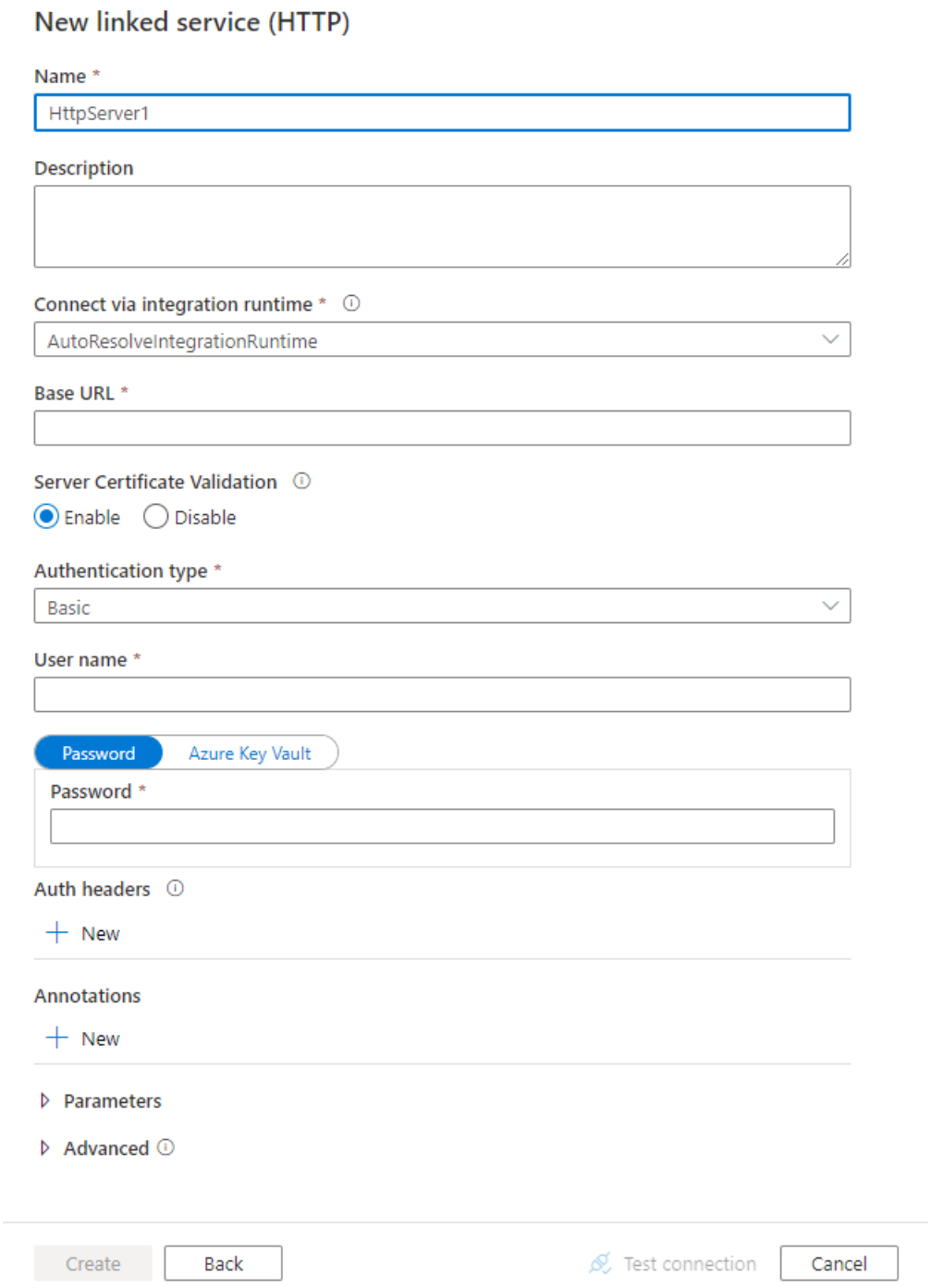
   - Azure Data Factory
   - Azure Synapse



2. Search for HTTP and select the HTTP connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties you can use to define entities that are specific to the HTTP connector.

Linked service properties

The following properties are supported for the HTTP linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **HttpServer**.	Yes
url	The base URL to the web server.	Yes

PROPERTY	DESCRIPTION	REQUIRED
enableServerCertificateValidation	Specify whether to enable server TLS/SSL certificate validation when you connect to an HTTP endpoint. If your HTTPS server uses a self-signed certificate, set this property to **false**.	No (the default is **true**)
authenticationType	Specifies the authentication type. Allowed values are **Anonymous**, **Basic**, **Digest**, **Windows**, and **ClientCertificate**. You can additionally configure authentication headers in `authHeader` property. See the sections that follow this table for more properties and JSON samples for these authentication types.	Yes
authHeaders	Additional HTTP request headers for authentication. For example, to use API key authentication, you can select authentication type as "Anonymous" and specify API key in the header.	No
connectVia	The Integration Runtime to use to connect to the data store. Learn more from Prerequisites section. If not specified, the default Azure Integration Runtime is used.	No

Using Basic, Digest, or Windows authentication

Set the **authenticationType** property to **Basic**, **Digest**, or **Windows**. In addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
userName	The user name to use to access the HTTP endpoint.	Yes
password	The password for the user (the **userName** value). Mark this field as a **SecureString** type to store it securely. You can also reference a secret stored in Azure Key Vault.	Yes

Example

```
{
    "name": "HttpLinkedService",
    "properties": {
        "type": "HttpServer",
        "typeProperties": {
            "authenticationType": "Basic",
            "url" : "<HTTP endpoint>",
            "userName": "<user name>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Using ClientCertificate authentication

To use ClientCertificate authentication, set the **authenticationType** property to **ClientCertificate**. In addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
embeddedCertData	Base64-encoded certificate data.	Specify either **embeddedCertData** or **certThumbprint**.
certThumbprint	The thumbprint of the certificate that's installed on your self-hosted Integration Runtime machine's cert store. Applies only when the self-hosted type of Integration Runtime is specified in the **connectVia** property.	Specify either **embeddedCertData** or **certThumbprint**.
password	The password that's associated with the certificate. Mark this field as a **SecureString** type to store it securely. You can also reference a secret stored in Azure Key Vault.	No

If you use **certThumbprint** for authentication and the certificate is installed in the personal store of the local computer, grant read permissions to the self-hosted Integration Runtime:

1. Open the Microsoft Management Console (MMC). Add the **Certificates** snap-in that targets **Local Computer**.
2. Expand **Certificates** > **Personal**, and then select **Certificates**.
3. Right-click the certificate from the personal store, and then select **All Tasks** > **Manage Private Keys**.
4. On the **Security** tab, add the user account under which the Integration Runtime Host Service (DIAHostService) is running, with read access to the certificate.
5. The HTTP connector loads only trusted certificates. If you're using a self-signed or nonintegrated CA-issued certificate, to enable trust, the certificate must also be installed in one of the following stores:
   - Trusted People
   - Third-Party Root Certification Authorities
   - Trusted Root Certification Authorities

Example 1: Using certThumbprint

```
{
    "name": "HttpLinkedService",
    "properties": {
        "type": "HttpServer",
        "typeProperties": {
            "authenticationType": "ClientCertificate",
            "url": "<HTTP endpoint>",
            "certThumbprint": "<thumbprint of certificate>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 2: Using embeddedCertData

```
{
    "name": "HttpLinkedService",
    "properties": {
        "type": "HttpServer",
        "typeProperties": {
            "authenticationType": "ClientCertificate",
            "url": "<HTTP endpoint>",
            "embeddedCertData": "<Base64-encoded cert data>",
            "password": {
                "type": "SecureString",
                "value": "password of cert"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Using authentication headers

In addition, you can configure request headers for authentication along with the built-in authentication types.

Example: Using API key authentication

```
{
    "name": "HttpLinkedService",
    "properties": {
        "type": "HttpServer",
        "typeProperties": {
            "url": "<HTTP endpoint>",
            "authenticationType": "Anonymous",
            "authHeader": {
                "x-api-key": {
                    "type": "SecureString",
                    "value": "<API key>"
                }
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for HTTP under `location` settings in format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `location` in dataset must be set to **HttpServerLocation**.	Yes
relativeUrl	A relative URL to the resource that contains the data. The HTTP connector copies data from the combined URL: `[URL specified in linked service][relative URL specified in dataset]` .	No

NOTE

The supported HTTP request payload size is around 500 KB. If the payload size you want to pass to your web endpoint is larger than 500 KB, consider batching the payload in smaller chunks.

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<HTTP linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "HttpServerLocation",
                "relativeUrl": "<relative url>"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy Activity properties

This section provides a list of properties that the HTTP source supports.

For a full list of sections and properties that are available for defining activities, see Pipelines.

HTTP as source

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for HTTP under `storeSettings` settings in format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under `storeSettings` must be set to **HttpReadSettings**.	Yes
requestMethod	The HTTP method. Allowed values are **Get** (default) and **Post**.	No
additionalHeaders	Additional HTTP request headers.	No
requestBody	The body for the HTTP request.	No

PROPERTY	DESCRIPTION	REQUIRED
httpRequestTimeout	The timeout (the **TimeSpan** value) for the HTTP request to get a response. This value is the timeout to get a response, not the timeout to read response data. The default value is **00:01:40**.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromHTTP",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "HttpReadSettings",
                    "requestMethod": "Post",
                    "additionalHeaders": "<header key: header value>\n<header key: header value>\n",
                    "requestBody": "<body for POST HTTP request>"
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Legacy models

> **NOTE**
>
> The following models are still supported as-is for backward compatibility. You are suggested to use the new model mentioned in above sections going forward, and the authoring UI has switched to generating the new model.

Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the dataset must be set to **HttpFile**.	Yes
relativeUrl	A relative URL to the resource that contains the data. When this property isn't specified, only the URL that's specified in the linked service definition is used.	No
requestMethod	The HTTP method. Allowed values are **Get** (default) and **Post**.	No
additionalHeaders	Additional HTTP request headers.	No
requestBody	The body for the HTTP request.	No
format	If you want to retrieve data from the HTTP endpoint as-is without parsing it, and then copy the data to a file-based store, skip the **format** section in both the input and output dataset definitions. If you want to parse the HTTP response content during copy, the following file format types are supported: **TextFormat**, **JsonFormat**, **AvroFormat**, **OrcFormat**, and **ParquetFormat**. Under **format**, set the **type** property to one of these values. For more information, see JSON format, Text format, Avro format, Orc format, and Parquet format.	No
compression	Specify the type and level of compression for the data. For more information, see Supported file formats and compression codecs. Supported types: **GZip**, **Deflate**, **BZip2**, and **ZipDeflate**. Supported levels: **Optimal** and **Fastest**.	No

> **NOTE**
>
> The supported HTTP request payload size is around 500 KB. If the payload size you want to pass to your web endpoint is larger than 500 KB, consider batching the payload in smaller chunks.

Example 1: Using the Get method (default)

```
{
    "name": "HttpSourceDataInput",
    "properties": {
        "type": "HttpFile",
        "linkedServiceName": {
            "referenceName": "<HTTP linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "relativeUrl": "<relative url>",
            "additionalHeaders": "Connection: keep-alive\nUser-Agent: Mozilla/5.0\n"
        }
    }
}
```

Example 2: Using the Post method

```
{
    "name": "HttpSourceDataInput",
    "properties": {
        "type": "HttpFile",
        "linkedServiceName": {
            "referenceName": "<HTTP linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "relativeUrl": "<relative url>",
            "requestMethod": "Post",
            "requestBody": "<body for POST HTTP request>"
        }
    }
}
```

Legacy copy activity source model

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the copy activity source must be set to **HttpSource**.	Yes
httpRequestTimeout	The timeout (the **TimeSpan** value) for the HTTP request to get a response. This value is the timeout to get a response, not the timeout to read response data. The default value is **00:01:40**.	No

Example

```
"activities":[
    {
        "name": "CopyFromHTTP",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<HTTP input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "HttpSource",
                "httpRequestTimeout": "00:01:00"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Next steps

For a list of data stores that Copy Activity supports as sources and sinks, see Supported data stores and formats.

Copy data from HubSpot using Azure Data Factory or Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from HubSpot. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This HubSpot connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks , see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

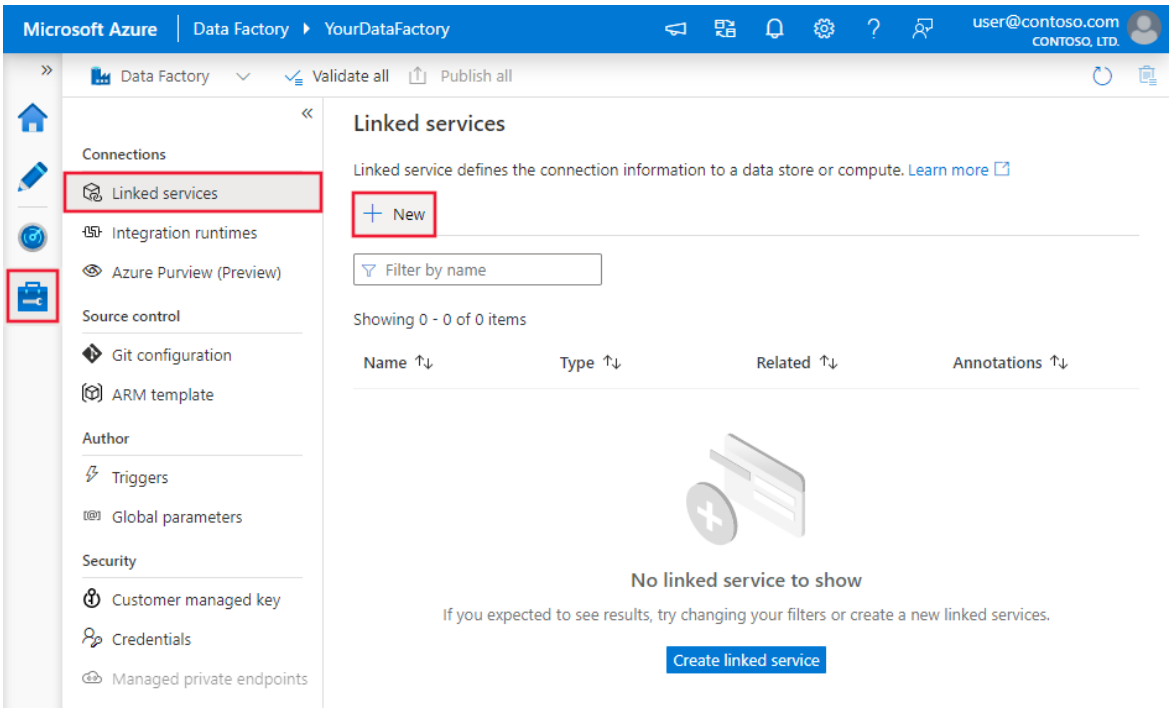
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to HubSpot using UI

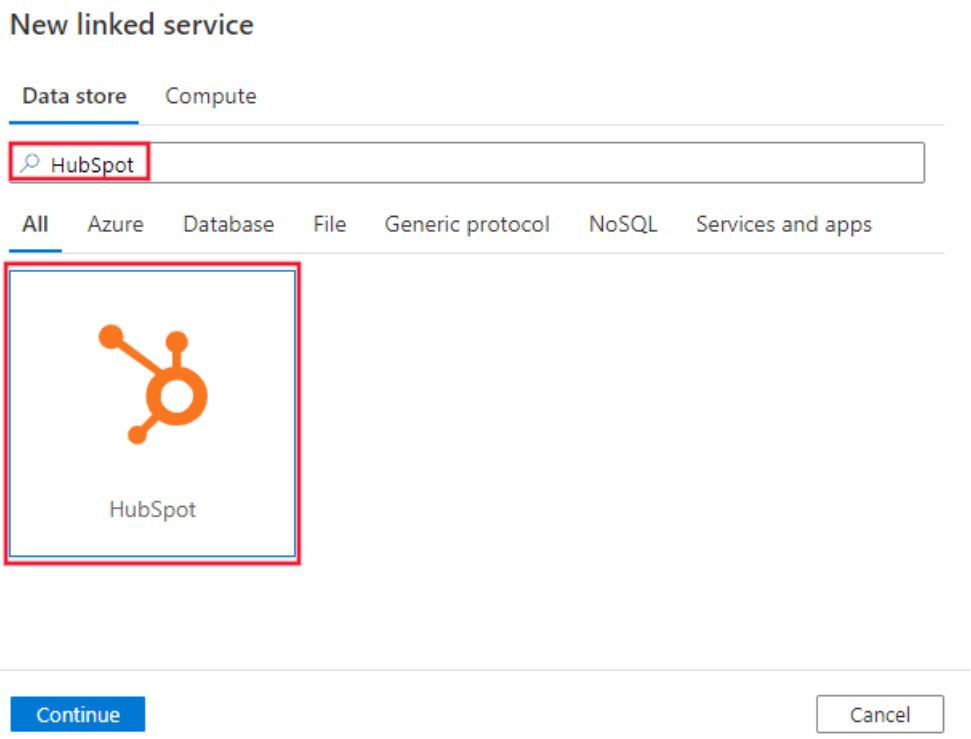
Use the following steps to create a linked service to HubSpot in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

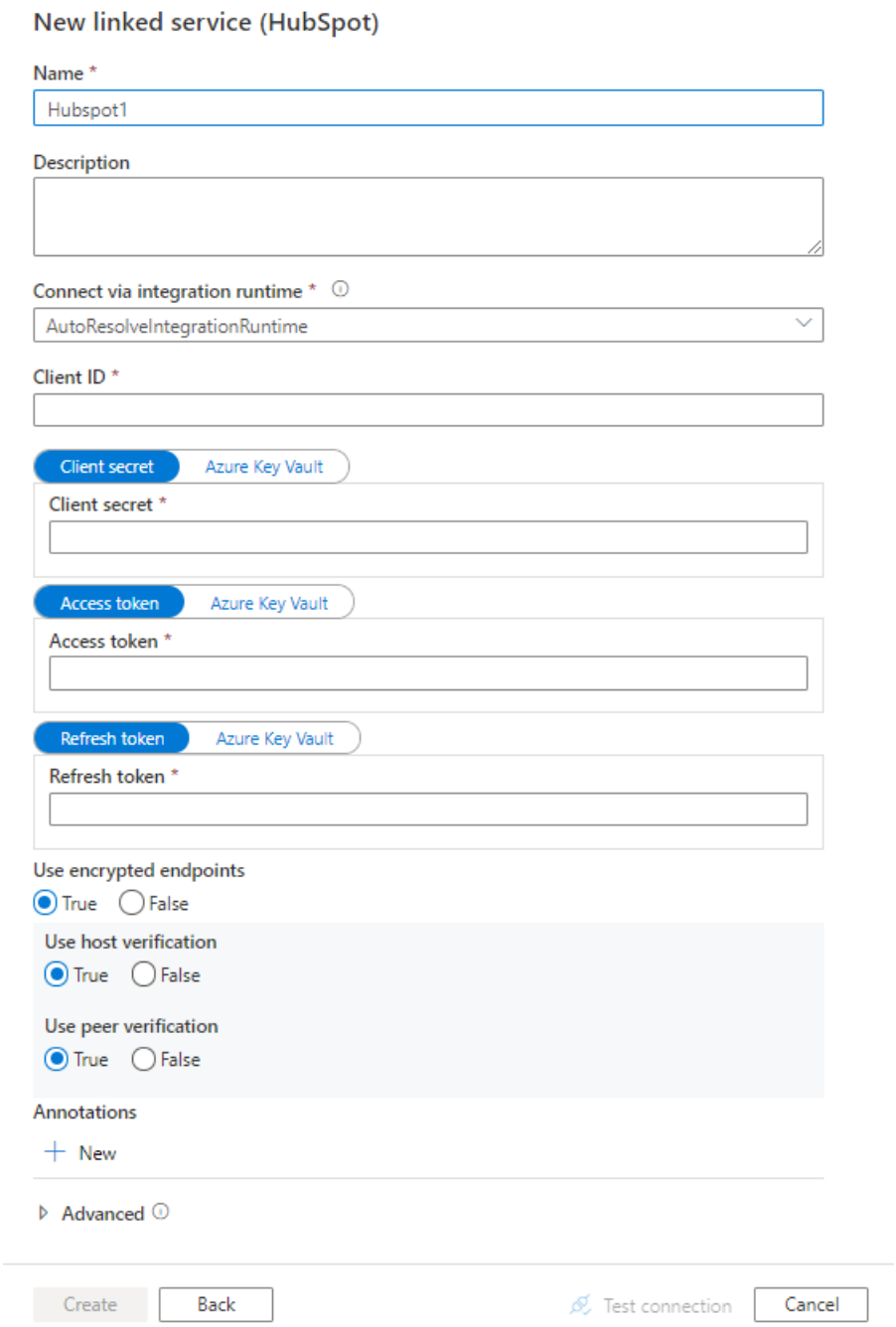
   - Azure Data Factory
   - Azure Synapse



2. Search for HubSpot and select the HubSpot connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to HubSpot connector.

Linked service properties

The following properties are supported for HubSpot linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Hubspot**	Yes
clientId	The client ID associated with your HubSpot application. Learn how to create an app in HubSpot from here.	Yes

PROPERTY	DESCRIPTION	REQUIRED
clientSecret	The client secret associated with your HubSpot application. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
accessToken	The access token obtained when initially authenticating your OAuth integration. Learn how to get access token with your client ID and secret from here. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
refreshToken	The refresh token obtained when initially authenticating your OAuth integration. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
    "name": "HubSpotLinkedService",
    "properties": {
        "type": "Hubspot",
        "typeProperties": {
            "clientId" : "<clientId>",
            "clientSecret": {
                "type": "SecureString",
                "value": "<clientSecret>"
            },
            "accessToken": {
                "type": "SecureString",
                "value": "<accessToken>"
            },
            "refreshToken": {
                "type": "SecureString",
                "value": "<refreshToken>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by HubSpot dataset.

To copy data from HubSpot, set the type property of the dataset to **HubspotObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **HubspotObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "HubSpotDataset",
    "properties": {
        "type": "HubspotObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<HubSpot linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by HubSpot source.

HubspotSource as source

To copy data from HubSpot, set the source type in the copy activity to **HubspotSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **HubspotSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM Companies where Company_Id = xxx"` .	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromHubspot",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<HubSpot input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "HubspotSource",
                "query": "SELECT * FROM Companies where Company_Id = xxx"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Impala using Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Impala. It builds on the Copy Activity overview article that presents a general overview of the copy activity.

Supported capabilities

This Impala connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity. Therefore, you don't need to manually install a driver to use this connector.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK

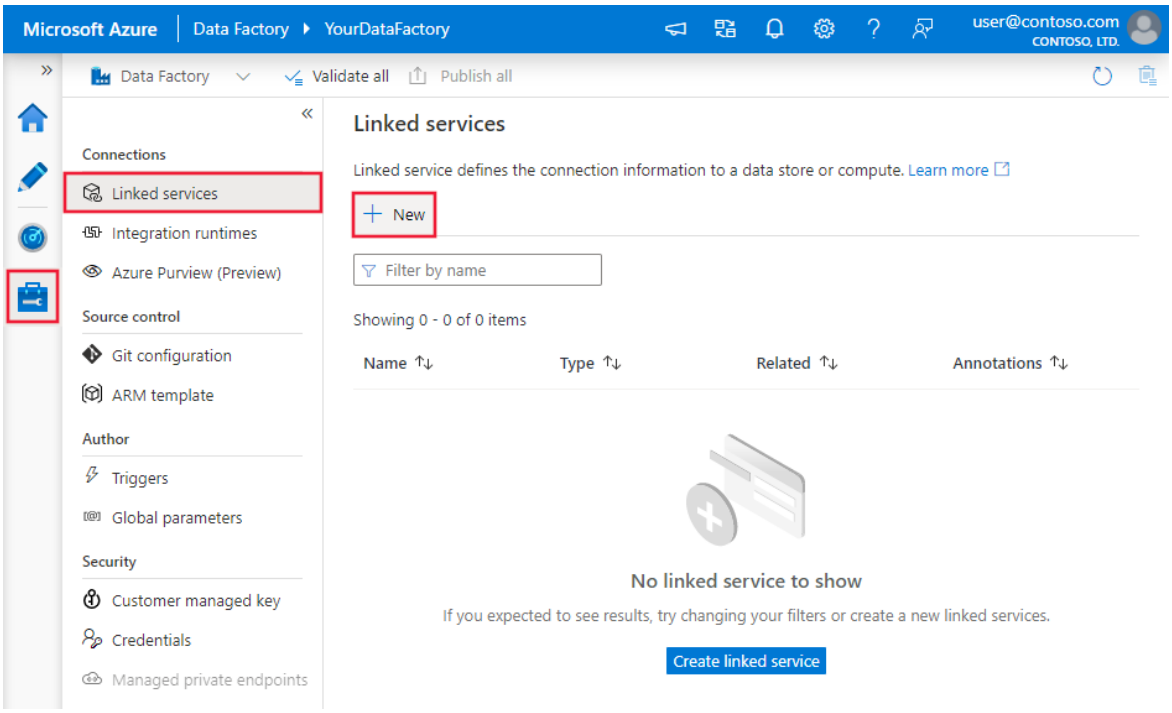
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Impala using UI

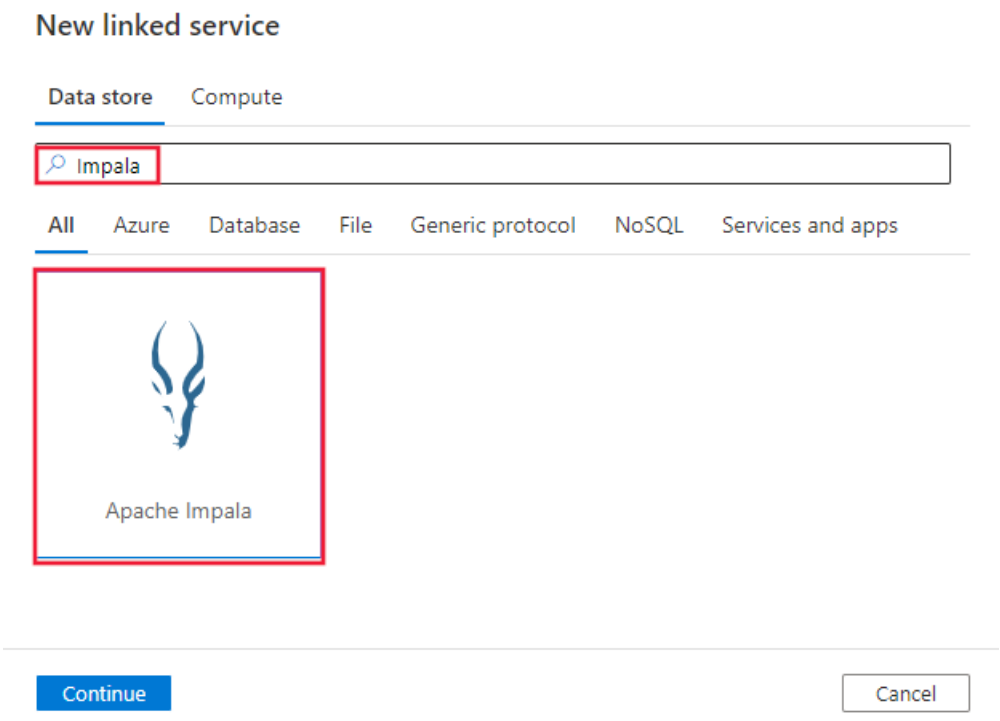
Use the following steps to create a linked service to Impala in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

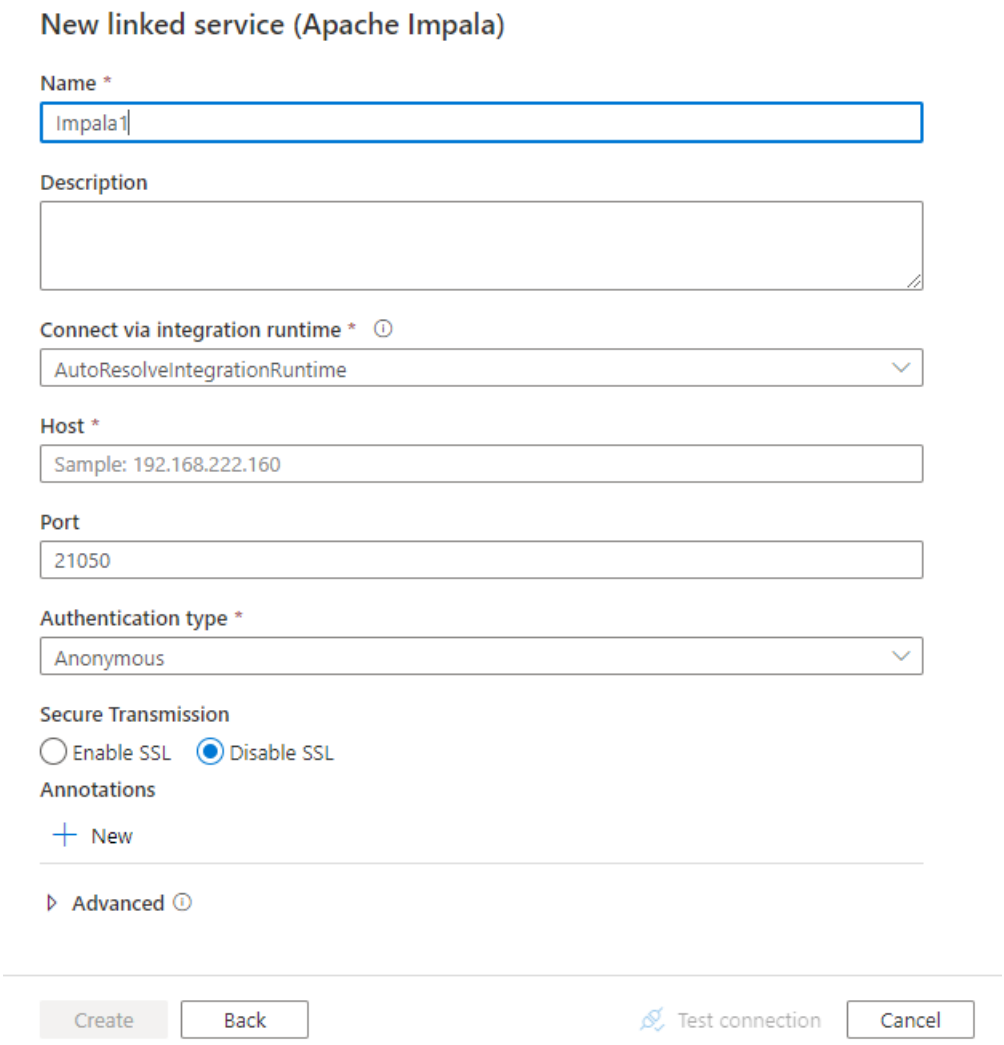
   - Azure Data Factory
   - Azure Synapse



2. Search for Impala and select the Impala connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to the Impala connector.

Linked service properties

The following properties are supported for Impala linked service.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **Impala**.	Yes
host	The IP address or host name of the Impala server (that is, 192.168.222.160).	Yes
port	The TCP port that the Impala server uses to listen for client connections. The default value is 21050.	No

PROPERTY	DESCRIPTION	REQUIRED
authenticationType	The authentication type to use. Allowed values are **Anonymous**, **SASLUsername**, and **UsernameAndPassword**.	Yes
username	The user name used to access the Impala server. The default value is anonymous when you use SASLUsername.	No
password	The password that corresponds to the user name when you use UsernameAndPassword. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
enableSsl	Specifies whether the connections to the server are encrypted by using TLS. The default value is **false**.	No
trustedCertPath	The full path of the .pem file that contains trusted CA certificates used to verify the server when you connect over TLS. This property can be set only when you use TLS on Self-hosted Integration Runtime. The default value is the cacerts.pem file installed with the integration runtime.	No
useSystemTrustStore	Specifies whether to use a CA certificate from the system trust store or from a specified PEM file. The default value is **false**.	No
allowHostNameCNMismatch	Specifies whether to require a CA-issued TLS/SSL certificate name to match the host name of the server when you connect over TLS. The default value is **false**.	No
allowSelfSignedServerCert	Specifies whether to allow self-signed certificates from the server. The default value is **false**.	No
connectVia	The integration runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "ImpalaLinkedService",
    "properties": {
        "type": "Impala",
        "typeProperties": {
            "host" : "<host>",
            "port" : "<port>",
            "authenticationType" : "UsernameAndPassword",
            "username" : "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the Impala dataset.

To copy data from Impala, set the type property of the dataset to **ImpalaObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **ImpalaObject**	Yes
schema	Name of the schema.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table with schema. This property is supported for backward compatibility. Use `schema` and `table` for new workload.	No (if "query" in activity source is specified)

Example

```
{
    "name": "ImpalaDataset",
    "properties": {
        "type": "ImpalaObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Impala linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the Impala source type.

Impala as a source type

To copy data from Impala, set the source type in the copy activity to **ImpalaSource**. The following properties are supported in the copy activity **source** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **ImpalaSource**.	Yes
query	Use the custom SQL query to read data. An example is `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromImpala",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Impala input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "ImpalaSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from and to IBM Informix using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from an IBM Informix data store. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Informix connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	②
Lookup activity	②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

Prerequisites

To use this Informix connector, you need to:

- Set up a Self-hosted Integration Runtime. See Self-hosted Integration Runtime article for details.
- Download the 64-bit Client SDK for Informix to create an ODBC connection for the data store on the Integration Runtime machine. For SDK download and setup, refer this article for details, or contact IBM support team for driver installation guidance.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

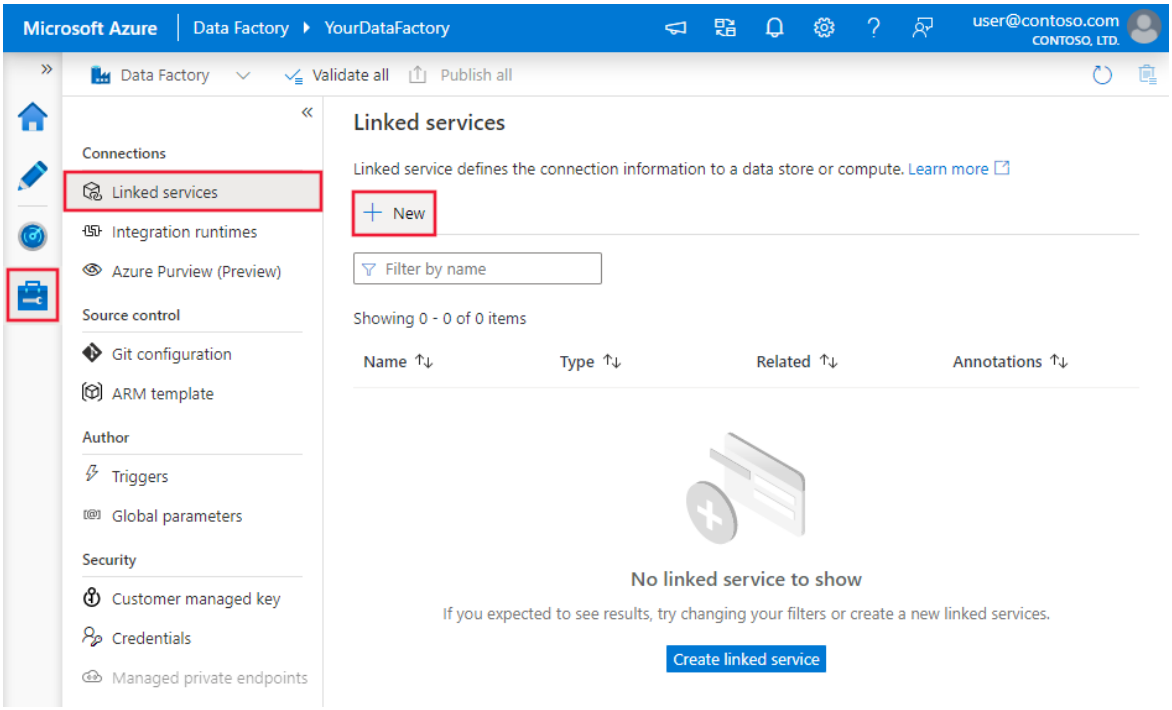
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Informix using UI

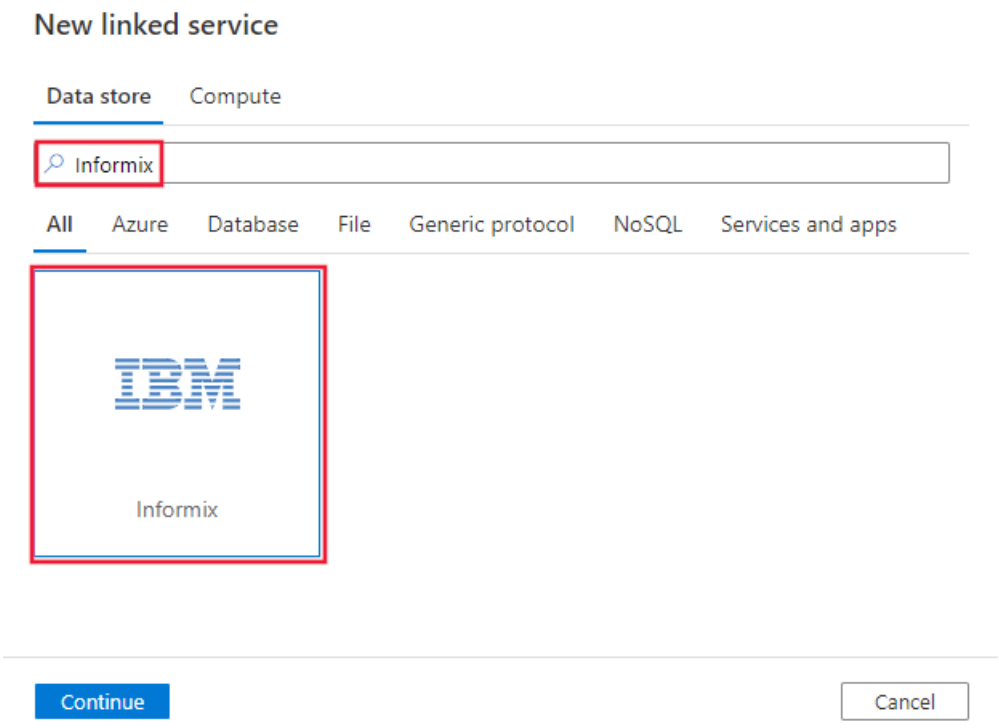
Use the following steps to create a linked service to Informix in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

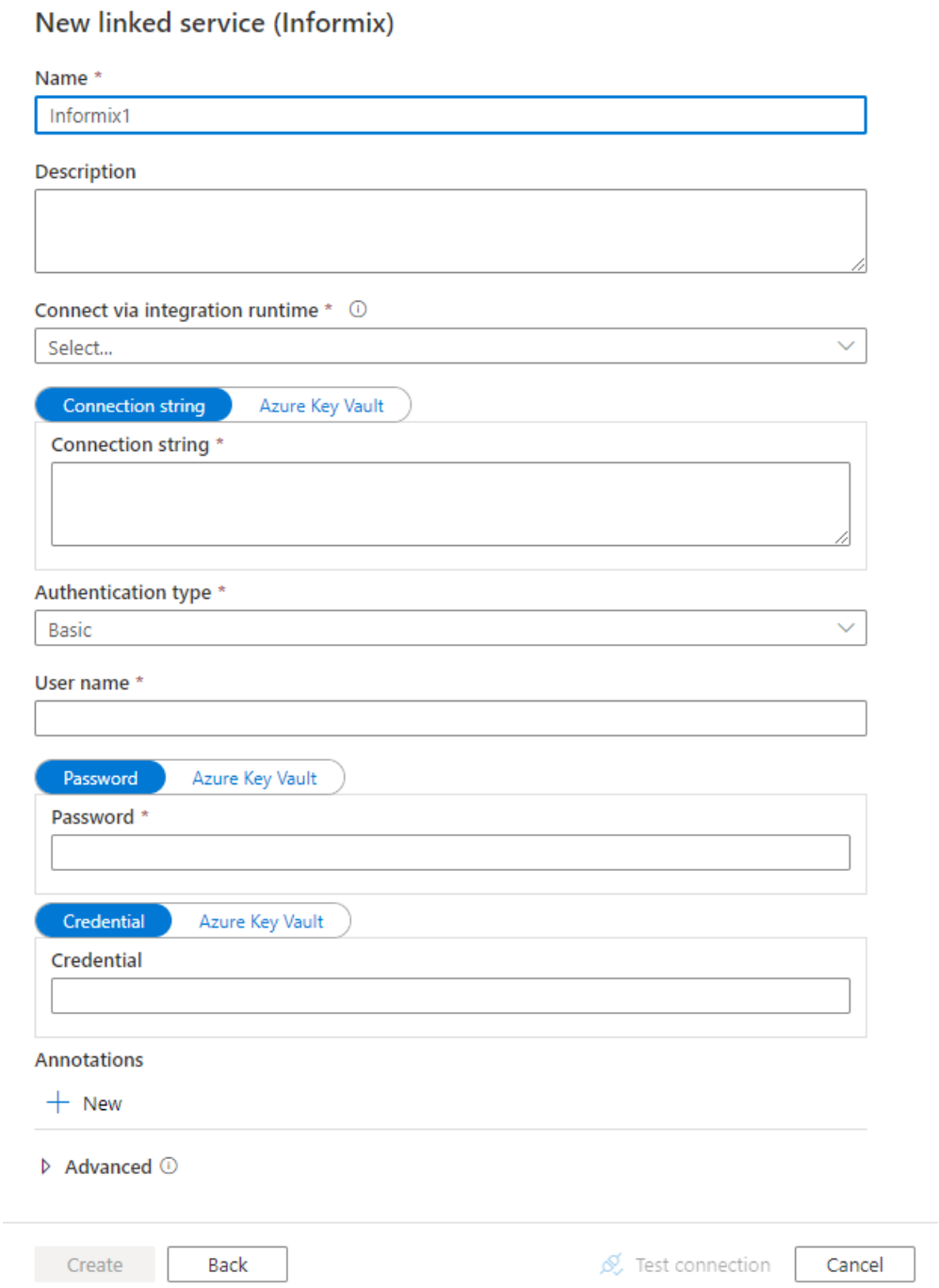
   - Azure Data Factory
   - Azure Synapse



2. Search for Informix and select the Informix connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Informix connector.

Linked service properties

The following properties are supported for Informix linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Informix**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	The ODBC connection string excluding the credential portion. You can specify the connection string or use the system DSN (Data Source Name) you set up on the Integration Runtime machine (you need still specify the credential portion in linked service accordingly). You can also put a password in Azure Key Vault and pull the `password` configuration out of the connection string. Refer to Store credentials in Azure Key Vault with more details.	Yes
authenticationType	Type of authentication used to connect to the Informix data store. Allowed values are: **Basic** and **Anonymous**.	Yes
userName	Specify user name if you are using Basic authentication.	No
password	Specify password for the user account you specified for the userName. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
credential	The access credential portion of the connection string specified in driver-specific property-value format. Mark this field as a SecureString.	No
connectVia	The Integration Runtime to be used to connect to the data store. A Self-hosted Integration Runtime is required as mentioned in Prerequisites.	Yes

Example:

```
{
    "name": "InformixLinkedService",
    "properties": {
        "type": "Informix",
        "typeProperties": {
            "connectionString": "<Informix connection string or DSN>",
            "authenticationType": "Basic",
            "userName": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Informix dataset.

To copy data from Informix, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **InformixTable**	Yes
tableName	Name of the table in the Informix.	No for source (if "query" in activity source is specified); Yes for sink

Example

```
{
    "name": "InformixDataset",
    "properties": {
        "type": "InformixTable",
        "linkedServiceName": {
            "referenceName": "<Informix linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "tableName": "<table name>"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Informix source.

Informix as source

To copy data from Informix, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **InformixSource**	Yes
query	Use the custom query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromInformix",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Informix input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "InformixSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Informix as sink

To copy data to Informix, the following properties are supported in the copy activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to: **InformixSink**	Yes
writeBatchTimeout	Wait time for the batch insert operation to complete before it times out. Allowed values are: timespan. Example: "00:30:00" (30 minutes).	No
writeBatchSize	Inserts data into the SQL table when the buffer size reaches writeBatchSize. Allowed values are: integer (number of rows).	No (default is 0 - auto detected)

PROPERTY	DESCRIPTION	REQUIRED
preCopyScript	Specify a SQL query for Copy Activity to execute before writing data into data store in each run. You can use this property to clean up the pre-loaded data.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToInformix",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Informix output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "InformixSink"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Jira using Azure Data Factory or Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Jira. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Jira connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

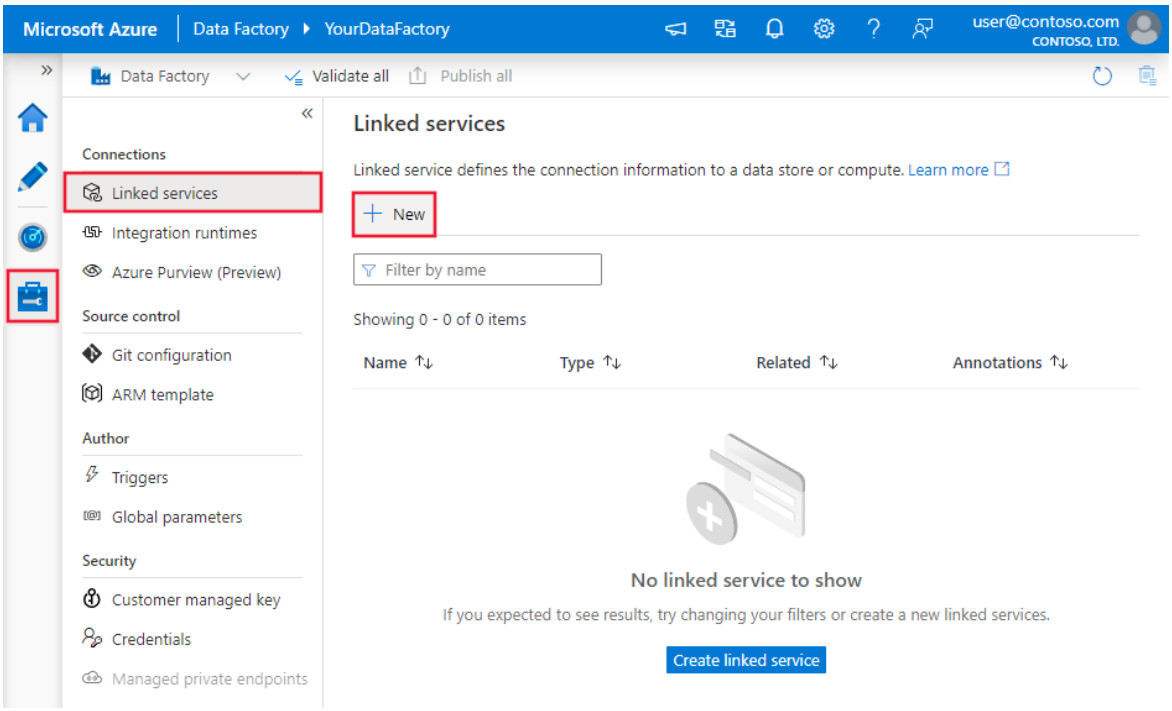
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Jira using UI

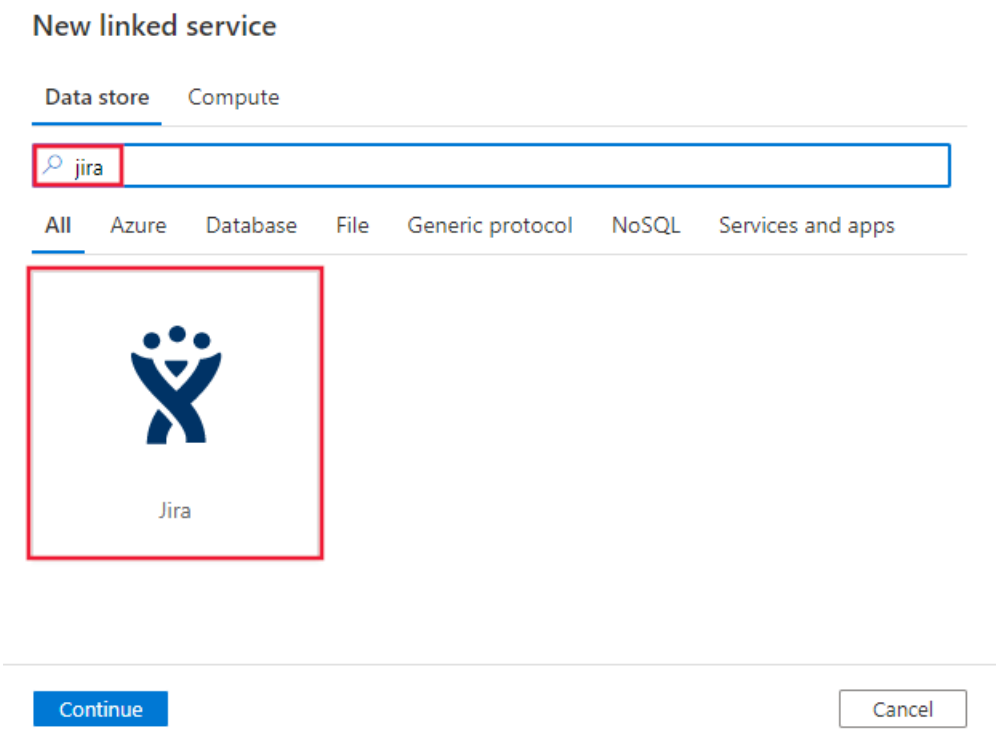
Use the following steps to create a linked service to Jira in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

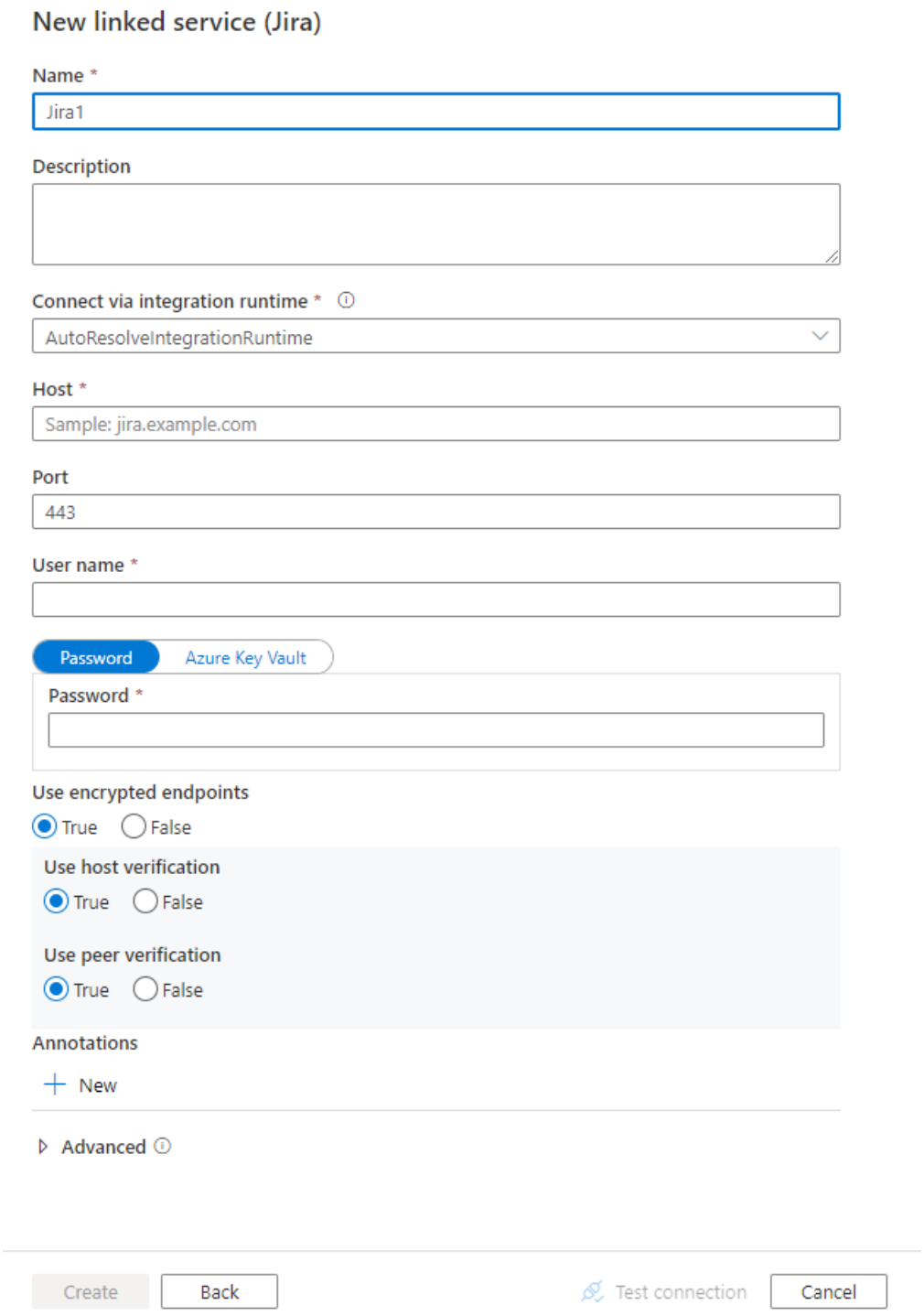
   - Azure Data Factory
   - Azure Synapse



2. Search for Jira and select the Jira connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Jira connector.

Linked service properties

The following properties are supported for Jira linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Jira**	Yes
host	The IP address or host name of the Jira service. (for example, jira.example.com)	Yes

PROPERTY	DESCRIPTION	REQUIRED
port	The TCP port that the Jira server uses to listen for client connections. The default value is 443 if connecting through HTTPS, or 8080 if connecting through HTTP.	No
username	The user name that you use to access Jira Service.	Yes
password	The Atlassian API token (if two-step verification is enabled on the Atlassian account) or password corresponding to the user name that you provided in the username field. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
    "name": "JiraLinkedService",
    "properties": {
        "type": "Jira",
        "typeProperties": {
            "host" : "<host>",
            "port" : "<port>",
            "username" : "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Jira dataset.

To copy data from Jira, set the type property of the dataset to **JiraObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **JiraObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "JiraDataset",
    "properties": {
        "type": "JiraObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Jira linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Jira source.

JiraSource as source

To copy data from Jira, set the source type in the copy activity to **JiraSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **JiraSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromJira",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Jira input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "JiraSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

JSON format in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 10 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Follow this article when you want to **parse the JSON files or write the data into JSON format**.

JSON format is supported for the following connectors:

- Amazon S3
- Amazon S3 Compatible Storage,
- Azure Blob
- Azure Data Lake Storage Gen1
- Azure Data Lake Storage Gen2
- Azure Files
- File System
- FTP
- Google Cloud Storage
- HDFS
- HTTP
- Oracle Cloud Storage
- SFTP

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the JSON dataset.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **Json**.	Yes
location	Location settings of the file(s). Each file-based connector has its own location type and supported properties under `location`. **See details in connector article -> Dataset properties section.**	Yes

PROPERTY	DESCRIPTION	REQUIRED
encodingName	The encoding type used to read/write test files. Allowed values are as follows: "UTF-8","UTF-8 without BOM", "UTF-16", "UTF-16BE", "UTF-32", "UTF-32BE", "US-ASCII", "UTF-7", "BIG5", "EUC-JP", "EUC-KR", "GB2312", "GB18030", "JOHAB", "SHIFT-JIS", "CP875", "CP866", "IBM00858", "IBM037", "IBM273", "IBM437", "IBM500", "IBM737", "IBM775", "IBM850", "IBM852", "IBM855", "IBM857", "IBM860", "IBM861", "IBM863", "IBM864", "IBM865", "IBM869", "IBM870", "IBM01140", "IBM01141", "IBM01142", "IBM01143", "IBM01144", "IBM01145", "IBM01146", "IBM01147", "IBM01148", "IBM01149", "ISO-2022-JP", "ISO-2022-KR", "ISO-8859-1", "ISO-8859-2", "ISO-8859-3", "ISO-8859-4", "ISO-8859-5", "ISO-8859-6", "ISO-8859-7", "ISO-8859-8", "ISO-8859-9", "ISO-8859-13", "ISO-8859-15", "WINDOWS-874", "WINDOWS-1250", "WINDOWS-1251", "WINDOWS-1252", "WINDOWS-1253", "WINDOWS-1254", "WINDOWS-1255", "WINDOWS-1256", "WINDOWS-1257", "WINDOWS-1258".	No
compression	Group of properties to configure file compression. Configure this section when you want to do compression/decompression during activity execution.	No
type (*under* `compression`)	The compression codec used to read/write JSON files. Allowed values are **bzip2**, **gzip**, **deflate**, **ZipDeflate**, **TarGzip**, **Tar**, **snappy**, or **lz4**. Default is not compressed. **Note** currently Copy activity doesn't support "snappy" & "lz4", and mapping data flow doesn't support "ZipDeflate"", "TarGzip" and "Tar". **Note** when using copy activity to decompress **ZipDeflate/TarGzip/Tar** file(s) and write to file-based sink data store, by default files are extracted to the folder: `<path specified in dataset>/<folder named as source compressed file>/` , use `preserveZipFileNameAsFolder` / `preserveCompressionFileNameAsFolder` on copy activity source to control whether to preserve the name of the compressed file(s) as folder structure.	No.

PROPERTY	DESCRIPTION	REQUIRED
level (*under* `compression`)	The compression ratio. Allowed values are **Optimal** or **Fastest**. - **Fastest:** The compression operation should complete as quickly as possible, even if the resulting file is not optimally compressed. - **Optimal**: The compression operation should be optimally compressed, even if the operation takes a longer time to complete. For more information, see Compression Level topic.	No

Below is an example of JSON dataset on Azure Blob Storage:

```
{
    "name": "JSONDataset",
    "properties": {
        "type": "Json",
        "linkedServiceName": {
            "referenceName": "<Azure Blob Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "container": "containername",
                "folderPath": "folder/subfolder",
            },
            "compression": {
                "type": "gzip"
            }
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the JSON source and sink.

Learn about how to extract data from JSON files and map to sink data store/format or vice versa from schema mapping.

JSON as source

The following properties are supported in the copy activity ***source*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **JSONSource**.	Yes
formatSettings	A group of properties. Refer to **JSON read settings** table below.	No

PROPERTY	DESCRIPTION	REQUIRED
storeSettings	A group of properties on how to read data from a data store. Each file-based connector has its own supported read settings under `storeSettings`. **See details in connector article -> Copy activity properties section**.	No

Supported **JSON read settings** under `formatSettings`:

PROPERTY	DESCRIPTION	REQUIRED
type	The type of formatSettings must be set to **JsonReadSettings**.	Yes
compressionProperties	A group of properties on how to decompress data for a given compression codec.	No
preserveZipFileNameAsFolder (*under* `compressionProperties` -> `type` *as* `ZipDeflateReadSettings`)	Applies when input dataset is configured with **ZipDeflate** compression. Indicates whether to preserve the source zip file name as folder structure during copy. - When set to **true (default)**, the service writes unzipped files to `<path specified in dataset>/<folder named as source zip file>/`. - When set to **false**, the service writes unzipped files directly to `<path specified in dataset>`. Make sure you don't have duplicated file names in different source zip files to avoid racing or unexpected behavior.	No
preserveCompressionFileNameAsFolder (*under* `compressionProperties` -> `type` *as* `TarGZipReadSettings` *or* `TarReadSettings`)	Applies when input dataset is configured with **TarGzip/Tar** compression. Indicates whether to preserve the source compressed file name as folder structure during copy. - When set to **true (default)**, the service writes decompressed files to `<path specified in dataset>/<folder named as source compressed file>/`. - When set to **false**, the service writes decompressed files directly to `<path specified in dataset>`. Make sure you don't have duplicated file names in different source files to avoid racing or unexpected behavior.	No

JSON as sink

The following properties are supported in the copy activity ***sink*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **JSONSink**.	Yes
formatSettings	A group of properties. Refer to **JSON write settings** table below.	No
storeSettings	A group of properties on how to write data to a data store. Each file-based connector has its own supported write settings under `storeSettings`. **See details in connector article -> Copy activity properties section**.	No

Supported **JSON write settings** under `formatSettings`:

PROPERTY	DESCRIPTION	REQUIRED
type	The type of formatSettings must be set to **JsonWriteSettings**.	Yes
filePattern	Indicate the pattern of data stored in each JSON file. Allowed values are: **setOfObjects** (JSON Lines) and **arrayOfObjects**. The **default** value is **setOfObjects**. See JSON file patterns section for details about these patterns.	No

JSON file patterns

When copying data from JSON files, copy activity can automatically detect and parse the following patterns of JSON files. When writing data to JSON files, you can configure the file pattern on copy activity sink.

- **Type I: setOfObjects**

  Each file contains single object, JSON lines, or concatenated objects.

  - **single object JSON example**

    ```
    {
        "time": "2015-04-29T07:12:20.9100000Z",
        "callingimsi": "466920403025604",
        "callingnum1": "678948008",
        "callingnum2": "567834760",
        "switch1": "China",
        "switch2": "Germany"
    }
    ```

  - **JSON Lines (default for sink)**

```
{"time":"2015-04-
29T07:12:20.9100000Z","callingimsi":"466920403025604","callingnum1":"678948008","callingnum2":
"567834760","switch1":"China","switch2":"Germany"}
{"time":"2015-04-
29T07:13:21.0220000Z","callingimsi":"466922202613463","callingnum1":"123436380","callingnum2":
"789037573","switch1":"US","switch2":"UK"}
{"time":"2015-04-
29T07:13:21.4370000Z","callingimsi":"466923101048691","callingnum1":"678901578","callingnum2":
"345626404","switch1":"Germany","switch2":"UK"}
```

- concatenated JSON example

```
{
    "time": "2015-04-29T07:12:20.9100000Z",
    "callingimsi": "466920403025604",
    "callingnum1": "678948008",
    "callingnum2": "567834760",
    "switch1": "China",
    "switch2": "Germany"
}
{
    "time": "2015-04-29T07:13:21.0220000Z",
    "callingimsi": "466922202613463",
    "callingnum1": "123436380",
    "callingnum2": "789037573",
    "switch1": "US",
    "switch2": "UK"
}
{
    "time": "2015-04-29T07:13:21.4370000Z",
    "callingimsi": "466923101048691",
    "callingnum1": "678901578",
    "callingnum2": "345626404",
    "switch1": "Germany",
    "switch2": "UK"
}
```

- Type II: arrayOfObjects

  Each file contains an array of objects.

```
[
    {
        "time": "2015-04-29T07:12:20.9100000Z",
        "callingimsi": "466920403025604",
        "callingnum1": "678948008",
        "callingnum2": "567834760",
        "switch1": "China",
        "switch2": "Germany"
    },
    {
        "time": "2015-04-29T07:13:21.0220000Z",
        "callingimsi": "466922202613463",
        "callingnum1": "123436380",
        "callingnum2": "789037573",
        "switch1": "US",
        "switch2": "UK"
    },
    {
        "time": "2015-04-29T07:13:21.4370000Z",
        "callingimsi": "466923101048691",
        "callingnum1": "678901578",
        "callingnum2": "345626404",
        "switch1": "Germany",
        "switch2": "UK"
    }
]
```

Mapping data flow properties

In mapping data flows, you can read and write to JSON format in the following data stores: Azure Blob Storage, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2 and SFTP, and you can read JSON format in Amazon S3.

Source properties

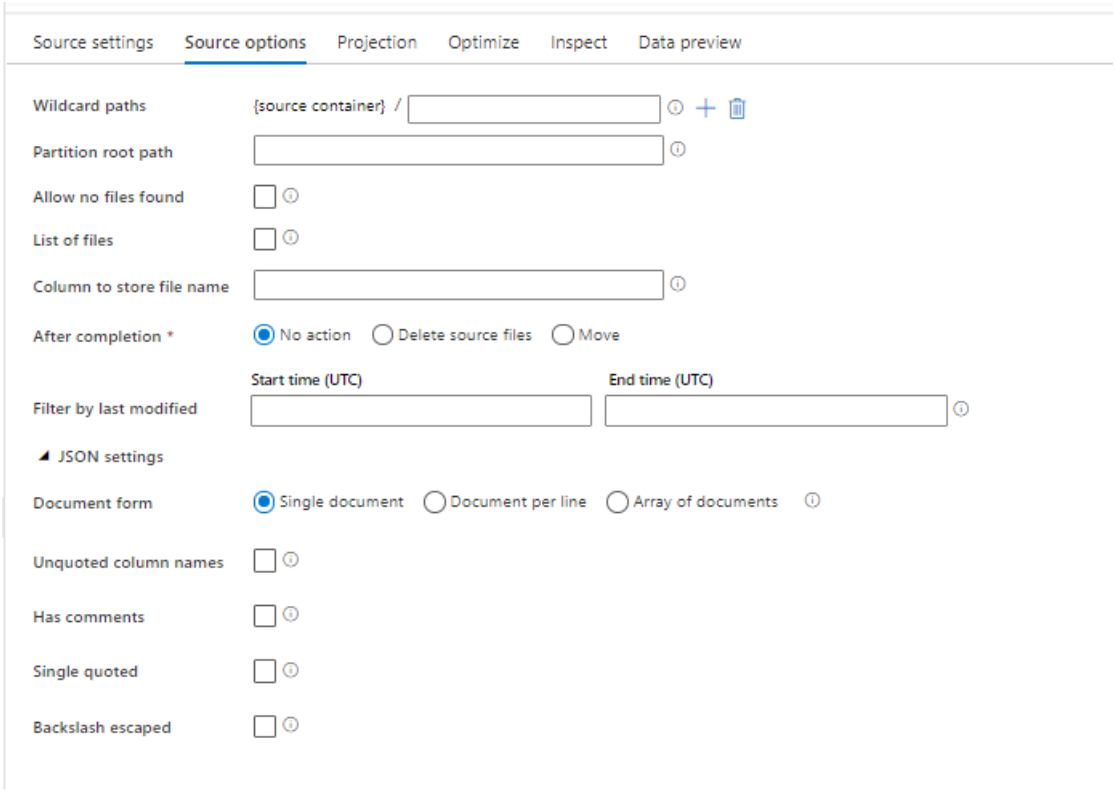
The below table lists the properties supported by a json source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Wild card paths	All files matching the wildcard path will be processed. Overrides the folder and file path set in the dataset.	no	String[]	wildcardPaths
Partition root path	For file data that is partitioned, you can enter a partition root path in order to read partitioned folders as columns	no	String	partitionRootPath
List of files	Whether your source is pointing to a text file that lists files to process	no	`true` or `false`	fileList

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Column to store file name	Create a new column with the source file name and path	no	String	rowUrlColumn
After completion	Delete or move the files after processing. File path starts from the container root	no	Delete: `true` or `false` Move: `['<from>', '<to>']`	purgeFiles moveFiles
Filter by last modified	Choose to filter files based upon when they were last altered	no	Timestamp	modifiedAfter modifiedBefore
Single document	Mapping data flows read one JSON document from each file	no	`true` or `false`	singleDocument
Unquoted column names	If **Unquoted column names** is selected, mapping data flows reads JSON columns that aren't surrounded by quotes.	no	`true` or `false`	unquotedColumnNames
Has comments	Select **Has comments** if the JSON data has C or C++ style commenting	no	`true` or `false`	asComments
Single quoted	Reads JSON columns that aren't surrounded by quotes	no	`true` or `false`	singleQuoted
Backslash escaped	Select **Backslash escaped** if backslashes are used to escape characters in the JSON data	no	`true` or `false`	backslashEscape
Allow no files found	If true, an error is not thrown if no files are found	no	`true` or `false`	ignoreNoFilesFound

Source format options

Using a JSON dataset as a source in your data flow allows you to set five additional settings. These settings can be found under the **JSON settings** accordion in the **Source Options** tab. For **Document Form** setting, you can select one of **Single document**, **Document per line** and **Array of documents** types.



Default

By default, JSON data is read in the following format.

```
{ "json": "record 1" }
{ "json": "record 2" }
{ "json": "record 3" }
```

Single document

If **Single document** is selected, mapping data flows read one JSON document from each file.

```
File1.json
{
    "json": "record 1"
}
File2.json
{
    "json": "record 2"
}
File3.json
{
    "json": "record 3"
}
```

If **Document per line** is selected, mapping data flows read one JSON document from each line in a file.

```
File1.json
{"json": "record 1"}

File2.json
 {"time":"2015-04-
29T07:12:20.9100000Z","callingimsi":"466920403025604","callingnum1":"678948008","callingnum2":"567834760","s
witch1":"China","switch2":"Germany"}
 {"time":"2015-04-
29T07:13:21.0220000Z","callingimsi":"466922202613463","callingnum1":"123436380","callingnum2":"789037573","s
witch1":"US","switch2":"UK"}

File3.json
 {"time":"2015-04-
29T07:12:20.9100000Z","callingimsi":"466920403025604","callingnum1":"678948008","callingnum2":"567834760","s
witch1":"China","switch2":"Germany"}
 {"time":"2015-04-
29T07:13:21.0220000Z","callingimsi":"466922202613463","callingnum1":"123436380","callingnum2":"789037573","s
witch1":"US","switch2":"UK"}
 {"time":"2015-04-
29T07:13:21.4370000Z","callingimsi":"466923101048691","callingnum1":"678901578","callingnum2":"345626404","s
witch1":"Germany","switch2":"UK"}
```

If **Array of documents** is selected, mapping data flows read one array of document from a file.

```
File.json
[
        {
            "time": "2015-04-29T07:12:20.9100000Z",
            "callingimsi": "466920403025604",
            "callingnum1": "678948008",
            "callingnum2": "567834760",
            "switch1": "China",
            "switch2": "Germany"
        },
        {
            "time": "2015-04-29T07:13:21.0220000Z",
            "callingimsi": "466922202613463",
            "callingnum1": "123436380",
            "callingnum2": "789037573",
            "switch1": "US",
            "switch2": "UK"
        },
        {
            "time": "2015-04-29T07:13:21.4370000Z",
            "callingimsi": "466923101048691",
            "callingnum1": "678901578",
            "callingnum2": "345626404",
            "switch1": "Germany",
            "switch2": "UK"
        }
    ]
```

> **NOTE**
> If data flows throw an error stating "corrupt_record" when previewing your JSON data, it is likely that your data contains contains a single document in your JSON file. Setting "single document" should clear that error.

Unquoted column names

If **Unquoted column names** is selected, mapping data flows reads JSON columns that aren't surrounded by quotes.

```
{ json: "record 1" }
{ json: "record 2" }
{ json: "record 3" }
```

Has comments

Select **Has comments** if the JSON data has C or C++ style commenting.

```
{ "json": /** comment **/ "record 1" }
{ "json": "record 2" }
{ /** comment **/ "json": "record 3" }
```

Single quoted

Select **Single quoted** if the JSON fields and values use single quotes instead of double quotes.

```
{ 'json': 'record 1' }
{ 'json': 'record 2' }
{ 'json': 'record 3' }
```

Backslash escaped

Select **Backslash escaped** if backslashes are used to escape characters in the JSON data.

```
{ "json": "record 1" }
{ "json": "\} \" \' \\ \n \\n record 2" }
{ "json": "record 3" }
```

Sink Properties

The below table lists the properties supported by a json sink. You can edit these properties in the **Settings** tab.

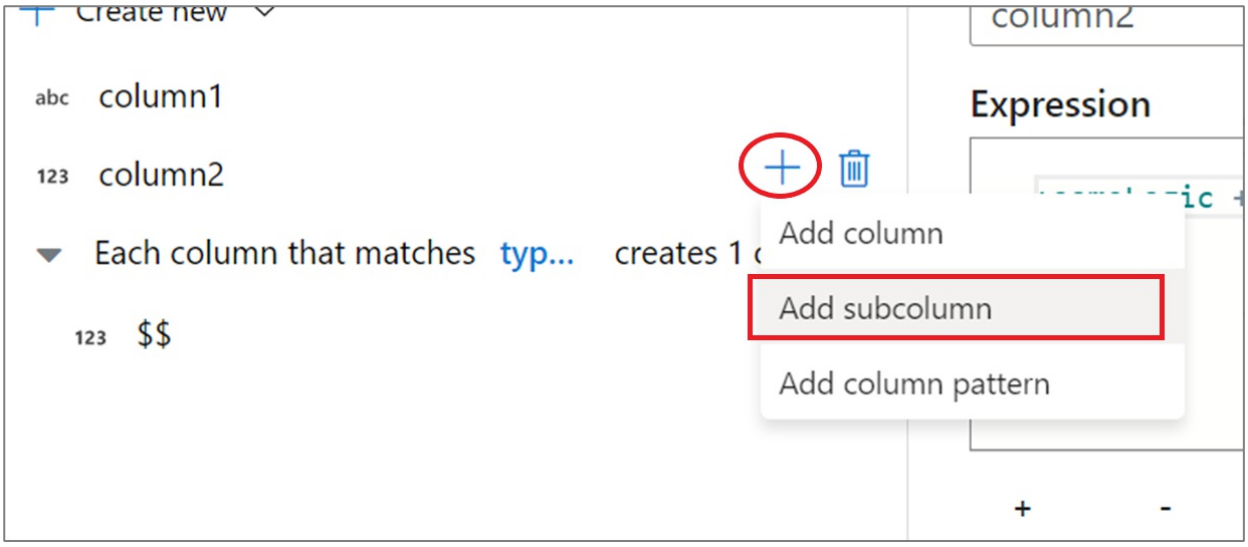
NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Clear the folder	If the destination folder is cleared prior to write	no	`true` or `false`	truncate
File name option	The naming format of the data written. By default, one file per partition in format `part-#####-tid-<guid>`	no	Pattern: String Per partition: String[] As data in column: String Output to single file: `['<fileName>']`	filePattern partitionFileNames rowUrlColumn partitionFileNames

Creating JSON structures in a derived column

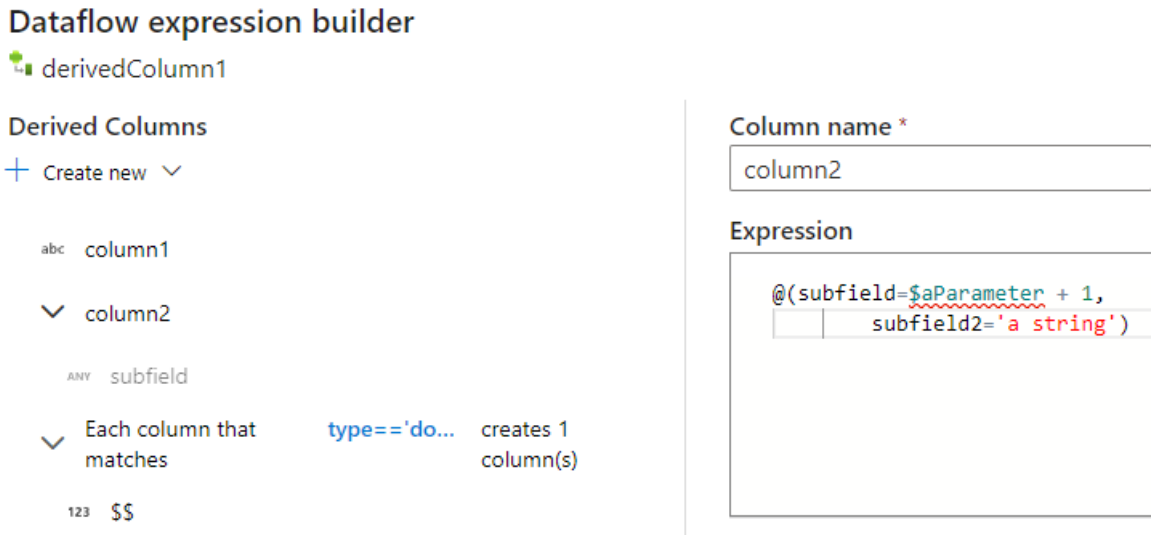
You can add a complex column to your data flow via the derived column expression builder. In the derived column transformation, add a new column and open the expression builder by clicking on the blue box. To make a column complex, you can enter the JSON structure manually or use the UX to add subcolumns interactively.

Using the expression builder UX

In the output schema side pane, hover over a column and click the plus icon. Select **Add subcolumn** to make the column a complex type.



You can add additional columns and subcolumns in the same way. For each non-complex field, an expression can be added in the expression editor to the right.



Entering the JSON structure manually

To manually add a JSON structure, add a new column and enter the expression in the editor. The expression follows the following general format:

```
@(
    field1=0,
    field2=@(
        field1=0
    )
)
```

If this expression were entered for a column named "complexColumn", then it would be written to the sink as the following JSON:

```
{
    "complexColumn": {
        "field1": 0,
        "field2": {
            "field1": 0
        }
    }
}
```

Sample manual script for complete hierarchical definition

```
@(
    title=Title,
    firstName=FirstName,
    middleName=MiddleName,
    lastName=LastName,
    suffix=Suffix,
    contactDetails=@(
        email=EmailAddress,
        phone=Phone
    ),
    address=@(
        line1=AddressLine1,
        line2=AddressLine2,
        city=City,
        state=StateProvince,
        country=CountryRegion,
        postCode=PostalCode
    ),
    ids=[
        toString(CustomerID), toString(AddressID), rowguid
    ]
)
```

Related connectors and formats

Here are some common connectors and formats related to the JSON format:

- Azure Blob Storage (connector-azure-blob-storage.md)
- Delimited text format(format-delimited-text.md)
- OData connector(connector-odata.md)
- Parquet format(format-parquet.md)

Next steps

- Copy activity overview
- Mapping data flow
- Lookup activity
- GetMetadata activity

Copy data from Magento using Azure Data Factory or Synapse Analytics(Preview)

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Magento. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Magento connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

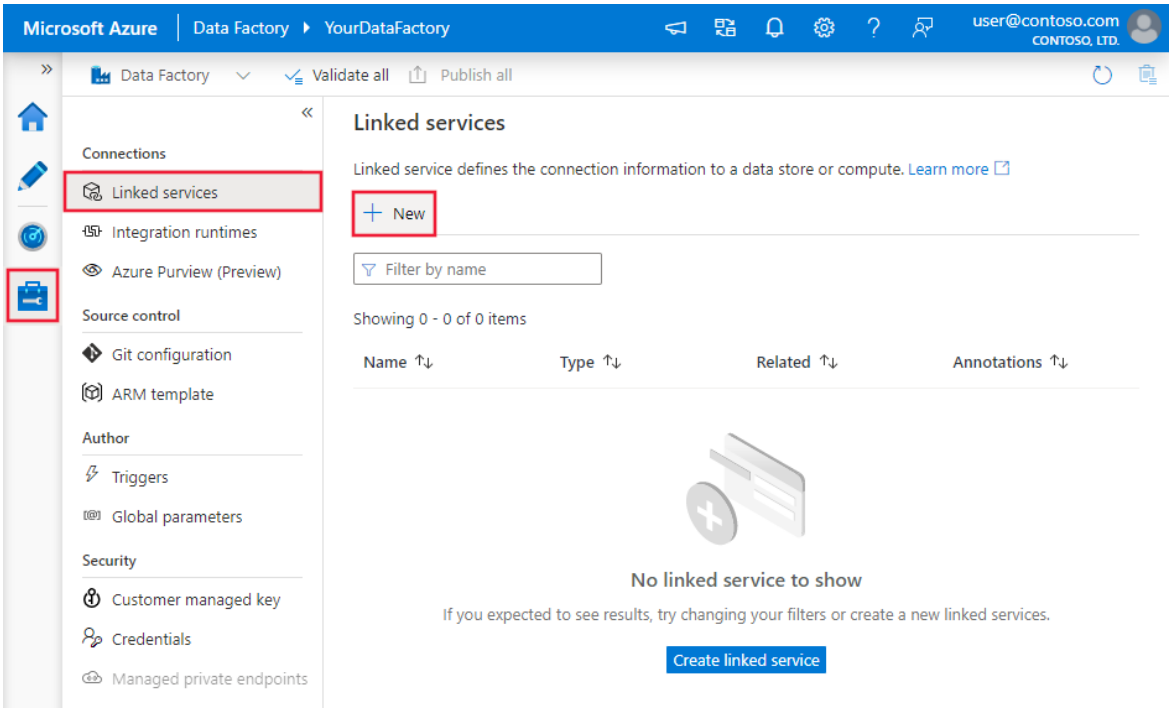
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Magento using UI

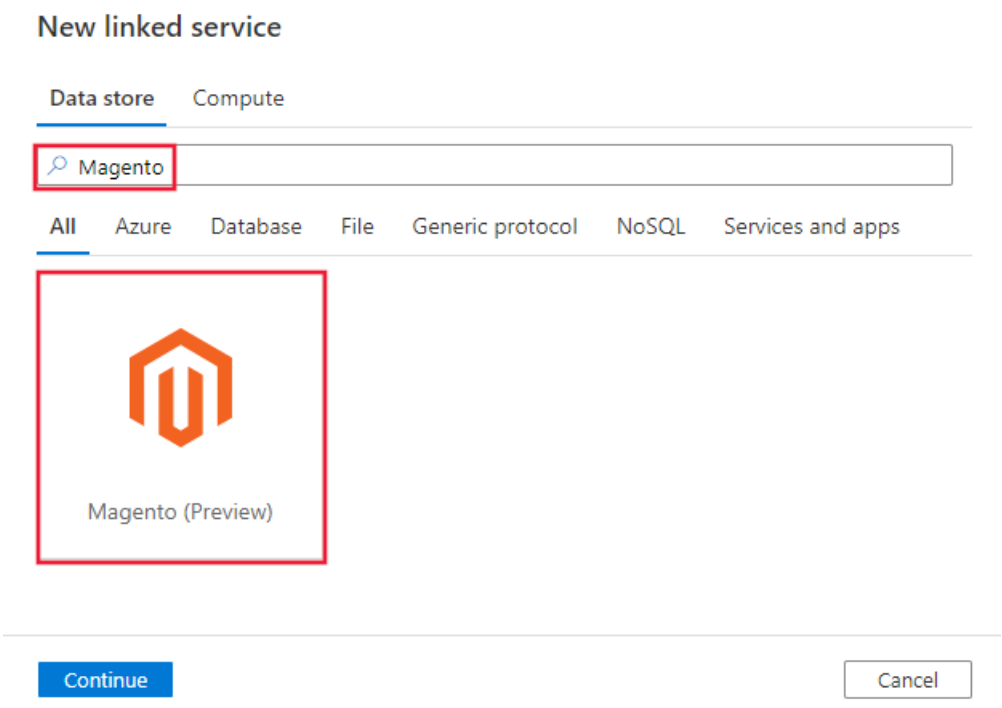
Use the following steps to create a linked service to Magento in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

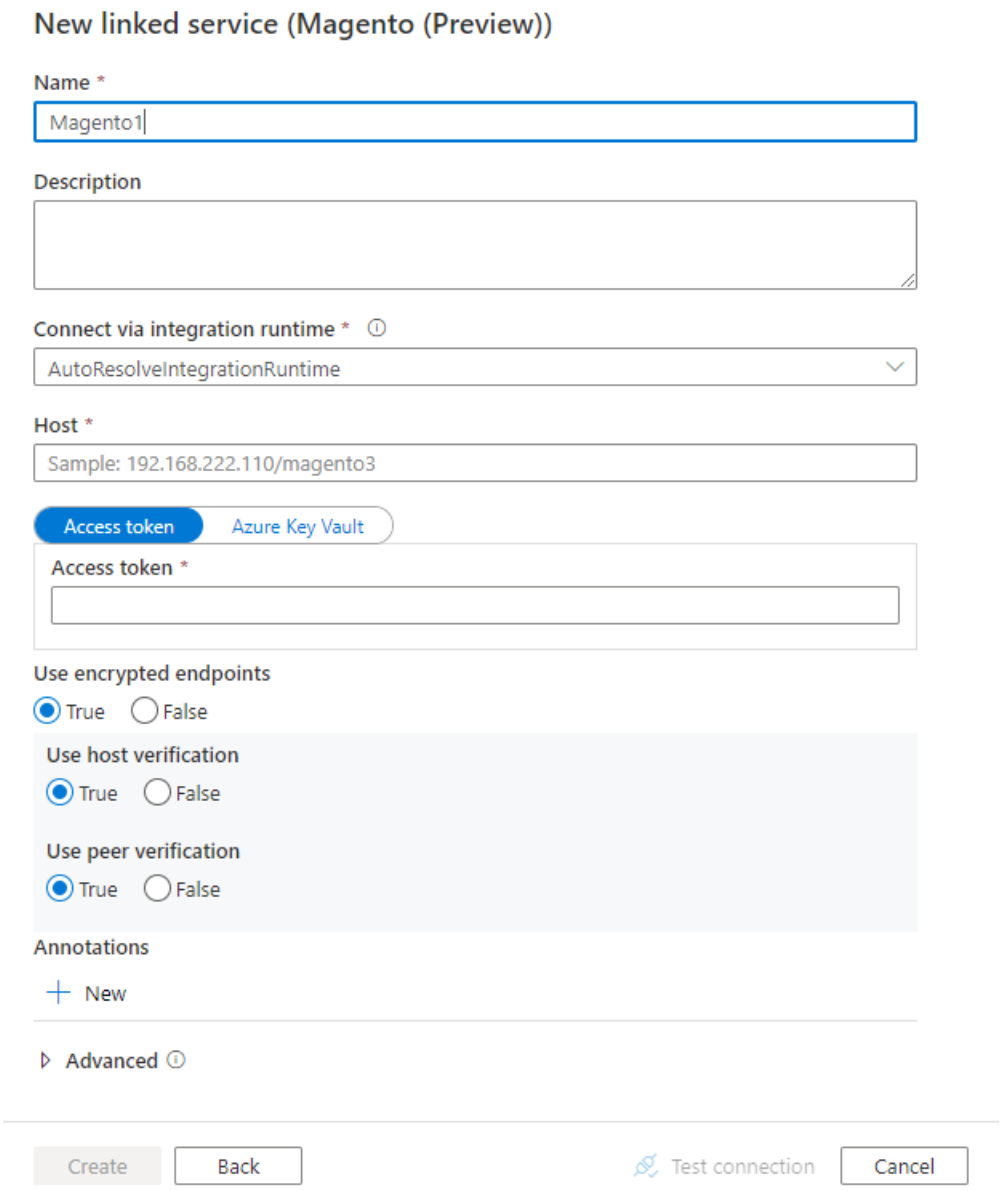
- Azure Data Factory
- Azure Synapse



2. Search for Magento and select the Magento connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Magento connector.

Linked service properties

The following properties are supported for Magento linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Magento**	Yes
host	The URL of the Magento instance. (that is, 192.168.222.110/magento3)	Yes
accessToken	The access token from Magento. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes

PROPERTY	DESCRIPTION	REQUIRED
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
    "name": "MagentoLinkedService",
    "properties": {
        "type": "Magento",
        "typeProperties": {
            "host" : "192.168.222.110/magento3",
            "accessToken": {
                "type": "SecureString",
                "value": "<accessToken>"
            },
            "useEncryptedEndpoints" : true,
            "useHostVerification" : true,
            "usePeerVerification" : true
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Magento dataset.

To copy data from Magento, set the type property of the dataset to **MagentoObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **MagentoObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "MagentoDataset",
    "properties": {
        "type": "MagentoObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Magento linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Magento source.

Magento as source

To copy data from Magento, set the source type in the copy activity to **MagentoSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **MagentoSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM Customers"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromMagento",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Magento input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "MagentoSource",
                "query": "SELECT * FROM Customers where Id > XXX"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from MariaDB using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from MariaDB. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This MariaDB connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

This connector currently supports MariaDB of version 10.0 to 10.5.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK

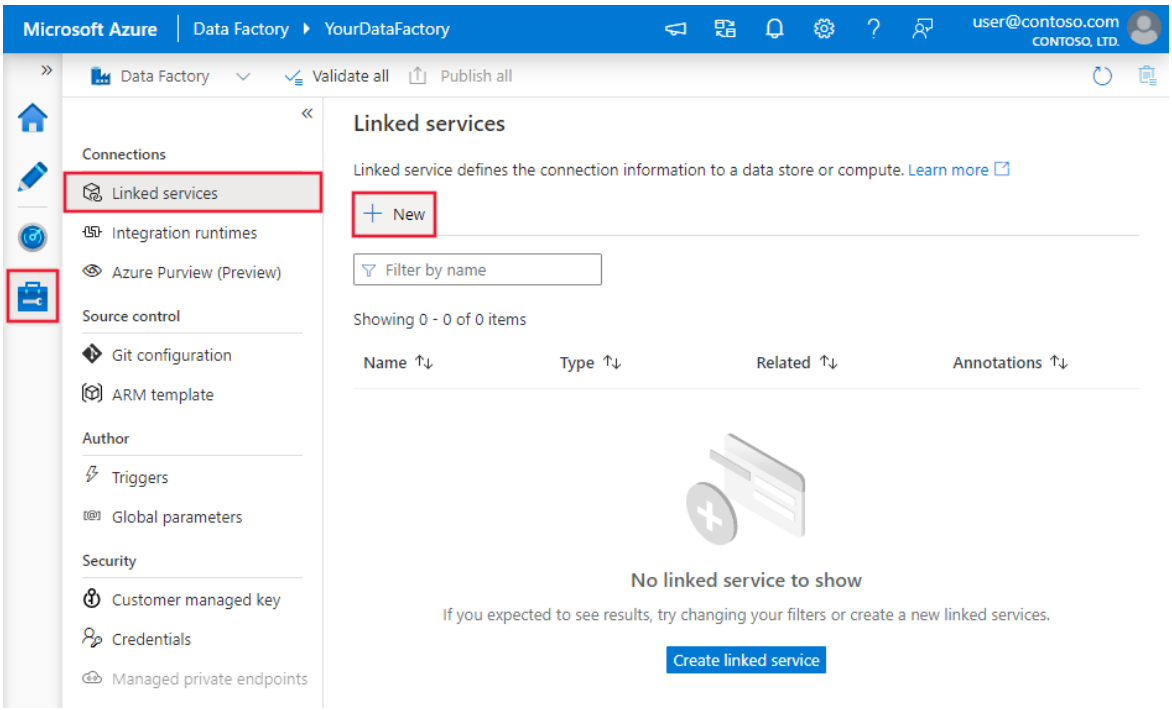
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to MariaDB using UI

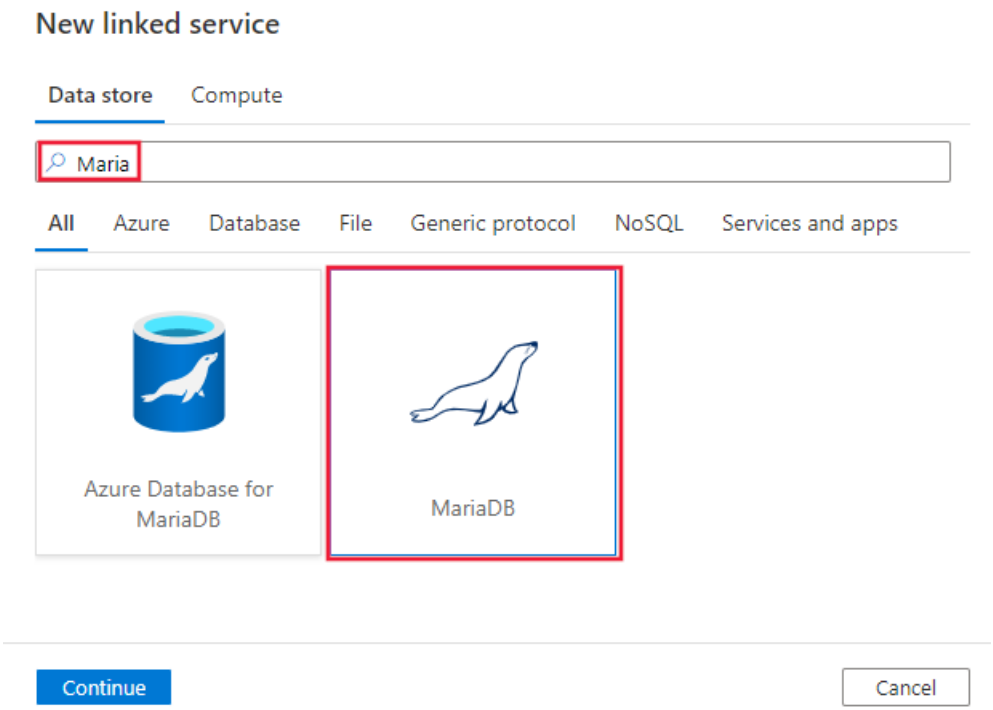
Use the following steps to create a linked service to MariaDB in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

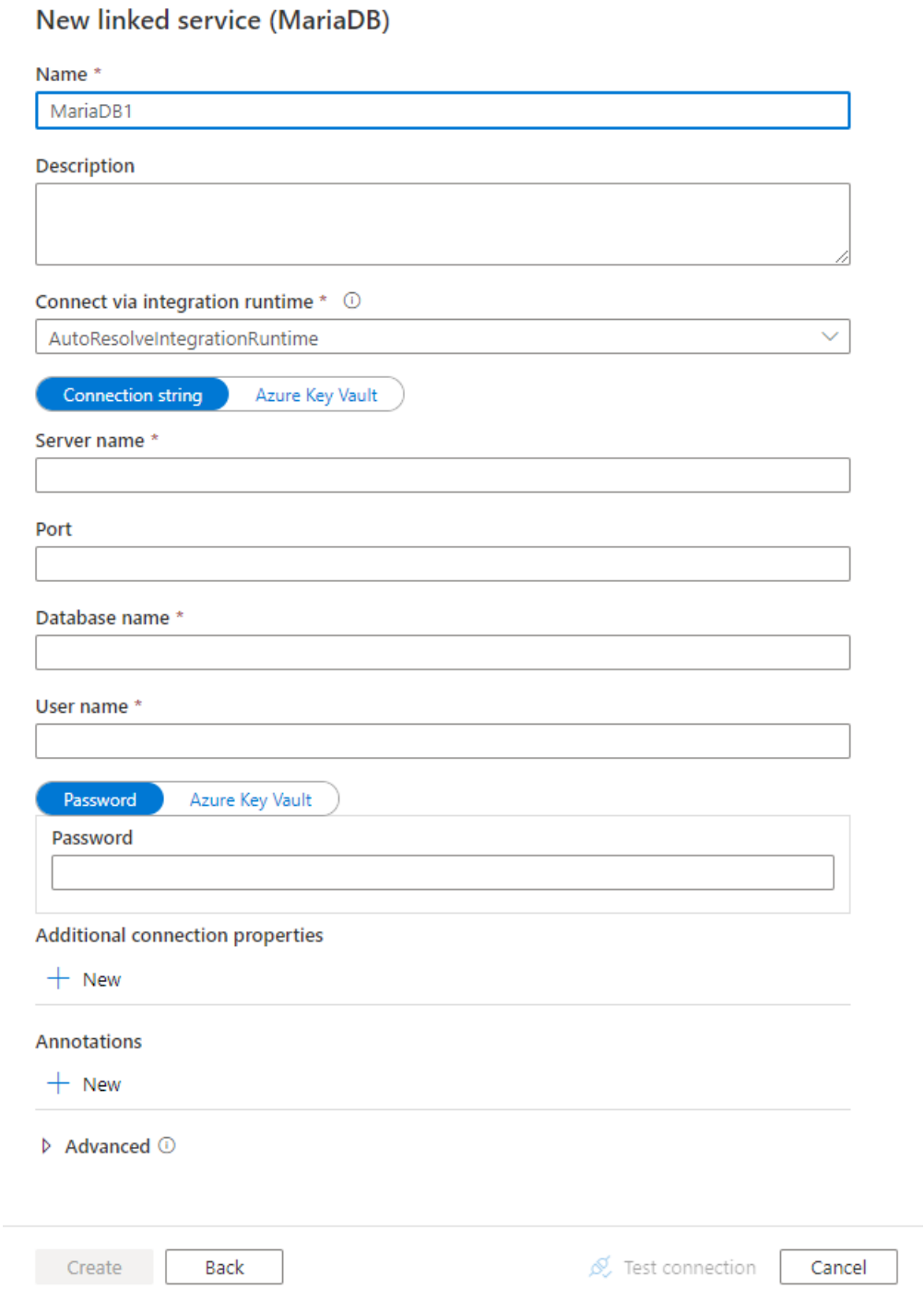
   - Azure Data Factory
   - Azure Synapse



2. Search for Maria and select the MariaDB connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to MariaDB connector.

Linked service properties

The following properties are supported for MariaDB linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **MariaDB**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	An ODBC connection string to connect to MariaDB. You can also put password in Azure Key Vault and pull the `pwd` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "MariaDBLinkedService",
    "properties": {
        "type": "MariaDB",
        "typeProperties": {
            "connectionString": "Server=<host>;Port=<port>;Database=<database>;UID=<user name>;PWD=
<password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "MariaDBLinkedService",
    "properties": {
        "type": "MariaDB",
        "typeProperties": {
            "connectionString": "Server=<host>;Port=<port>;Database=<database>;UID=<user name>;",
            "pwd": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section

provides a list of properties supported by MariaDB dataset.

To copy data from MariaDB, set the type property of the dataset to **MariaDBTable**. There is no additional type-specific property in this type of dataset.

Example

```
{
    "name": "MariaDBDataset",
    "properties": {
        "type": "MariaDBTable",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<MariaDB linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by MariaDB source.

MariaDB as source

To copy data from MariaDB, set the source type in the copy activity to **MariaDBSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **MariaDBSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromMariaDB",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<MariaDB input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "MariaDBSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Marketo using Azure Data Factory or Synapse Analytics (Preview)

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Marketo. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Marketo connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Currently, Marketo instance which is integrated with external CRM is not supported.

> **NOTE**
> This Marketo connector is built on top of the Marketo REST API. Be aware that the Marketo has concurrent request limit on service side. If you hit errors saying "Error while attempting to use REST API: Max rate limit '100' exceeded with in '20' secs (606)" or "Error while attempting to use REST API: Concurrent access limit '10' reached (615)", consider to reduce the concurrent copy activity runs to reduce the number of requests to the service.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

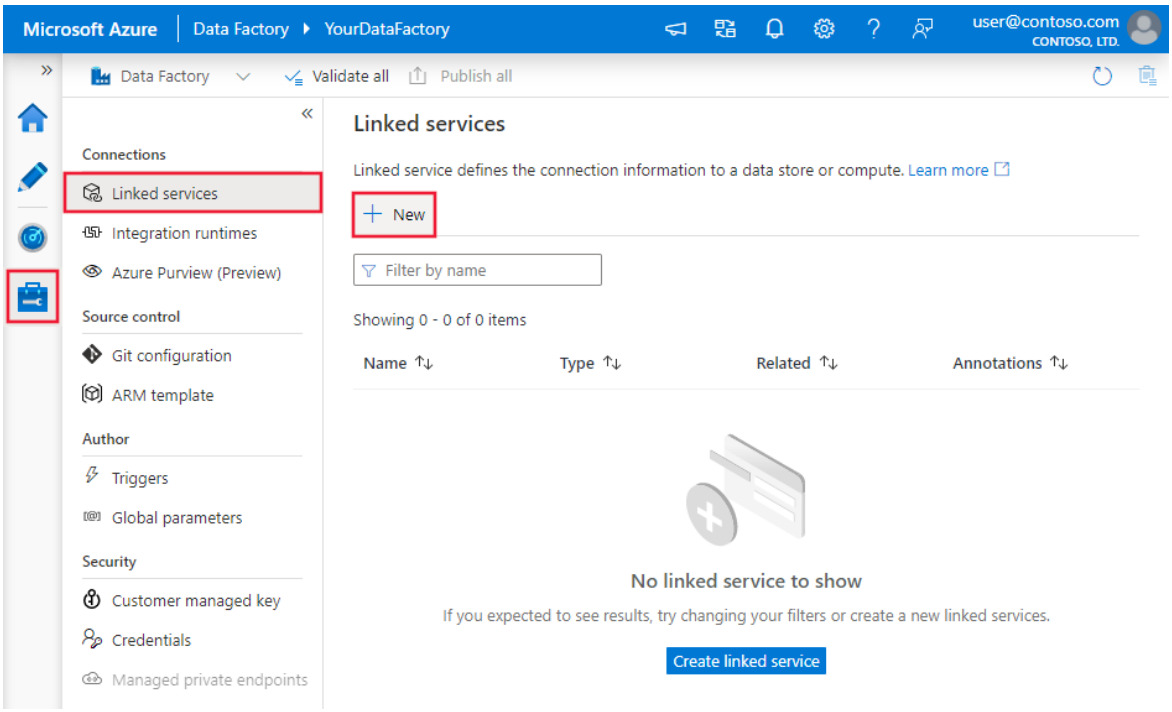
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Marketo using UI

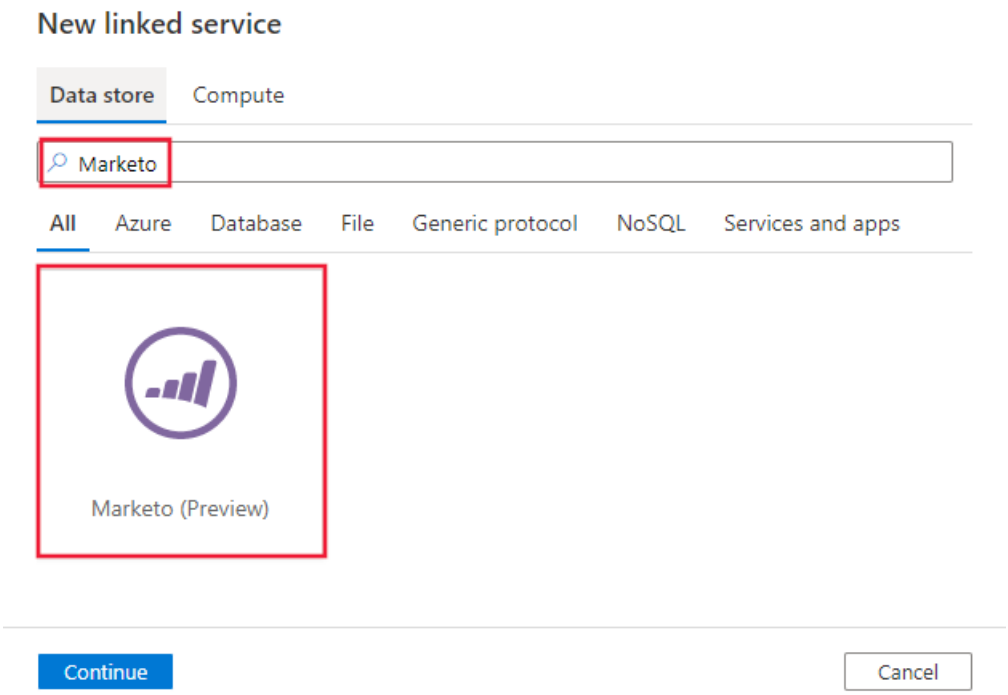
Use the following steps to create a linked service to Marketo in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

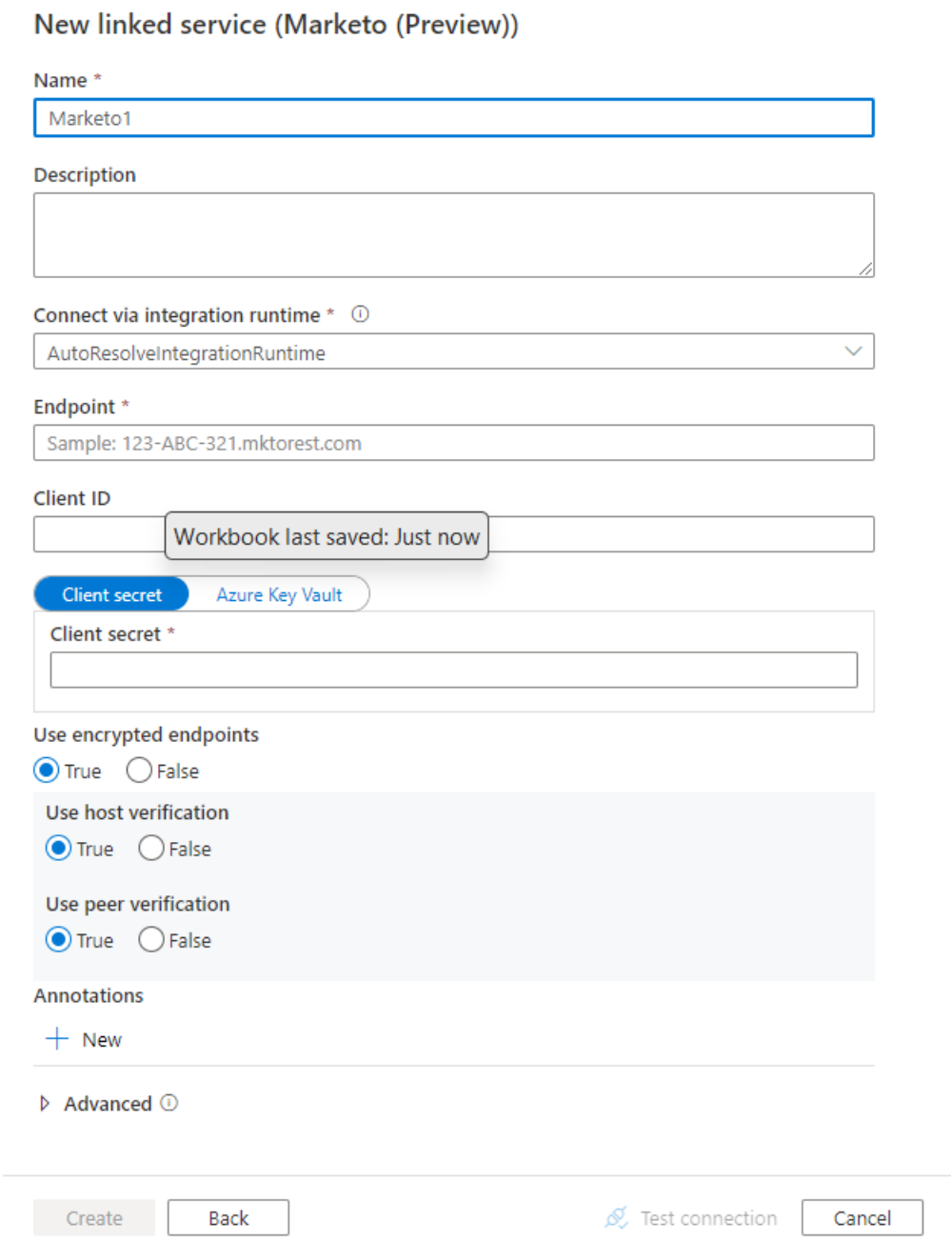
   - Azure Data Factory
   - Azure Synapse



2. Search for Marketo and select the Marketo connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Marketo connector.

Linked service properties

The following properties are supported for Marketo linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Marketo**	Yes
endpoint	The endpoint of the Marketo server. (i.e. 123-ABC-321.mktorest.com)	Yes
clientId	The client Id of your Marketo service.	Yes

PROPERTY	DESCRIPTION	REQUIRED
clientSecret	The client secret of your Marketo service. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
    "name": "MarketoLinkedService",
    "properties": {
        "type": "Marketo",
        "typeProperties": {
            "endpoint" : "123-ABC-321.mktorest.com",
            "clientId" : "<clientId>",
            "clientSecret": {
                "type": "SecureString",
                "value": "<clientSecret>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Marketo dataset.

To copy data from Marketo, set the type property of the dataset to **MarketoObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **MarketoObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "MarketoDataset",
    "properties": {
        "type": "MarketoObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Marketo linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Marketo source.

Marketo as source

To copy data from Marketo, set the source type in the copy activity to **MarketoSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **MarketoSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM Activitiy_Types"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromMarketo",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Marketo input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "MarketoSource",
                "query": "SELECT top 1000 * FROM Activitiy_Types"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from and to Microsoft Access using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from a Microsoft Access data store. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Microsoft Access connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	②
Lookup activity	②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

Prerequisites

To use this Microsoft Access connector, you need to:

- Set up a Self-hosted Integration Runtime. See Self-hosted Integration Runtime article for details.
- Install the Microsoft Access ODBC driver for the data store on the Integration Runtime machine.

> **NOTE**
> Microsoft Access 2016 version of ODBC driver doesn't work with this connector. Use Microsoft Access 2013 or 2010 version of ODBC driver instead.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

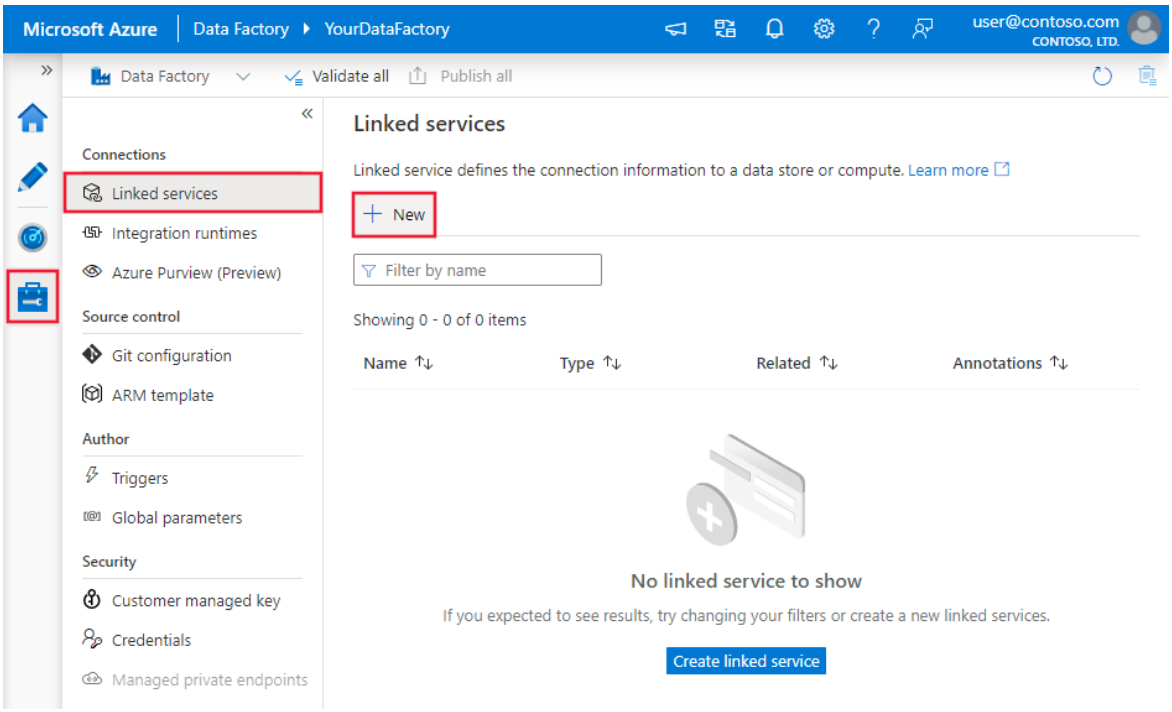
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Microsoft Access using UI

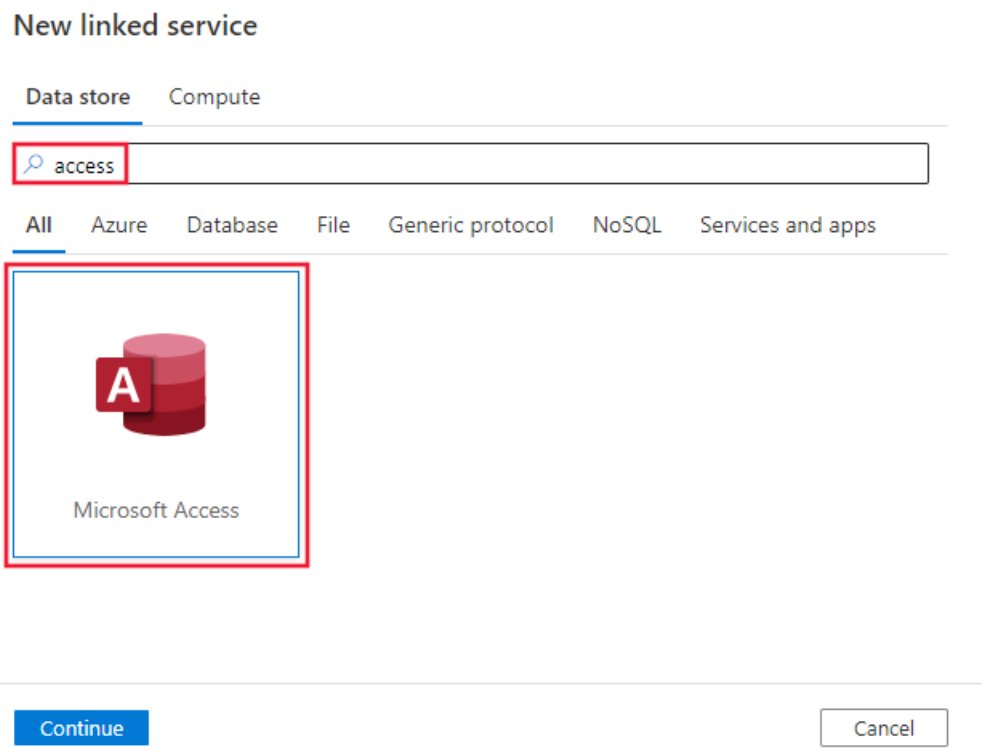
Use the following steps to create a linked service to Microsoft Access in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

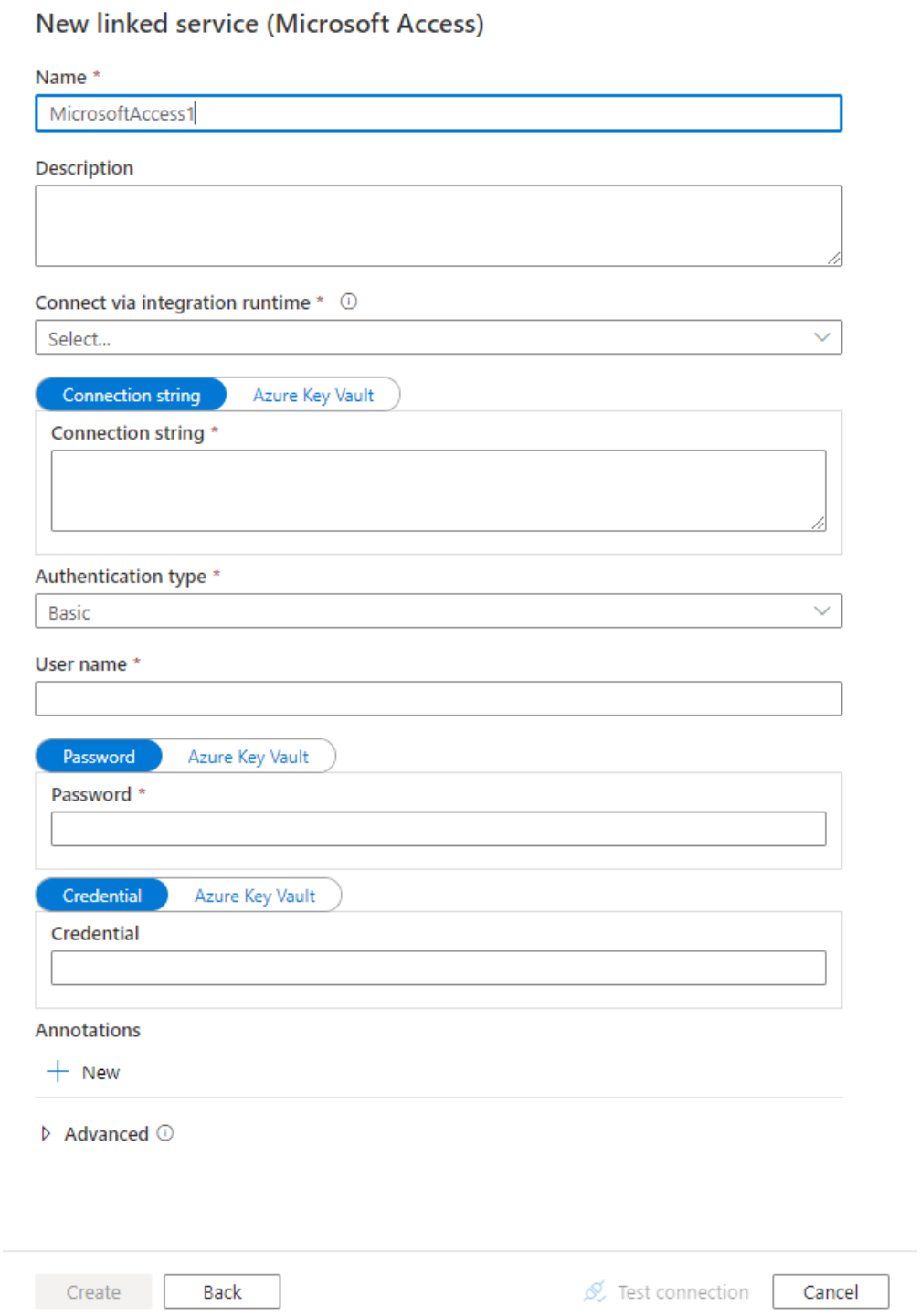
   - Azure Data Factory
   - Azure Synapse



2. Search for Access and select the Microsoft Access connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Microsoft Access connector.

Linked service properties

The following properties are supported for Microsoft Access linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **MicrosoftAccess**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	The ODBC connection string excluding the credential portion. You can specify the connection string or use the system DSN (Data Source Name) you set up on the Integration Runtime machine (you need still specify the credential portion in linked service accordingly). You can also put a password in Azure Key Vault and pull the `password` configuration out of the connection string. Refer to Store credentials in Azure Key Vault with more details.	Yes
authenticationType	Type of authentication used to connect to the Microsoft Access data store. Allowed values are: **Basic** and **Anonymous**.	Yes
userName	Specify user name if you are using Basic authentication.	No
password	Specify password for the user account you specified for the userName. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
credential	The access credential portion of the connection string specified in driver-specific property-value format. Mark this field as a SecureString.	No
connectVia	The Integration Runtime to be used to connect to the data store. A Self-hosted Integration Runtime is required as mentioned in Prerequisites.	Yes

Example:

```
{
    "name": "MicrosoftAccessLinkedService",
    "properties": {
        "type": "MicrosoftAccess",
        "typeProperties": {
            "connectionString": "Driver={Microsoft Access Driver (*.mdb, *.accdb)};Dbq=<path to your DB file
e.g. C:\\mydatabase.accdb>;",
            "authenticationType": "Basic",
            "userName": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Microsoft Access dataset.

To copy data from Microsoft Access, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **MicrosoftAccessTable**	Yes
tableName	Name of the table in the Microsoft Access.	No for source (if "query" in activity source is specified); Yes for sink

Example

```
{
    "name": "MicrosoftAccessDataset",
    "properties": {
        "type": "MicrosoftAccessTable",
        "linkedServiceName": {
            "referenceName": "<Microsoft Access linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "tableName": "<table name>"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Microsoft Access source.

Microsoft Access as source

To copy data from Microsoft Access, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **MicrosoftAccessSource**	Yes
query	Use the custom query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromMicrosoftAccess",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Microsoft Access input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "MicrosoftAccessSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Microsoft Access as sink

To copy data to Microsoft Access, the following properties are supported in the copy activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to: **MicrosoftAccessSink**	Yes
writeBatchTimeout	Wait time for the batch insert operation to complete before it times out. Allowed values are: timespan. Example: “00:30:00” (30 minutes).	No

PROPERTY	DESCRIPTION	REQUIRED
writeBatchSize	Inserts data into the SQL table when the buffer size reaches writeBatchSize. Allowed values are: integer (number of rows).	No (default is 0 - auto detected)
preCopyScript	Specify a SQL query for Copy Activity to execute before writing data into data store in each run. You can use this property to clean up the pre-loaded data.	No
maxConcurrentConnections	The upper limit of concurrent connecti ons established to the data store durin g the activity run.Specify a value only when you want to limit concurrent con nections.	No

Example:

```
"activities":[
    {
        "name": "CopyToMicrosoftAccess",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Microsoft Access output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "MicrosoftAccessSink"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from or to MongoDB using Azure Data Factory or Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory Synapse Analytics pipelines to copy data from and to a MongoDB database. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> The new MongoDB connector provides improved native MongoDB support. If you are using the legacy MongoDB connector in your solution, supported as-is for backward compatibility only, refer to MongoDB connector (legacy) article.

Supported capabilities

This MongoDB connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Specifically, this MongoDB connector supports **versions up to 4.2**.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK

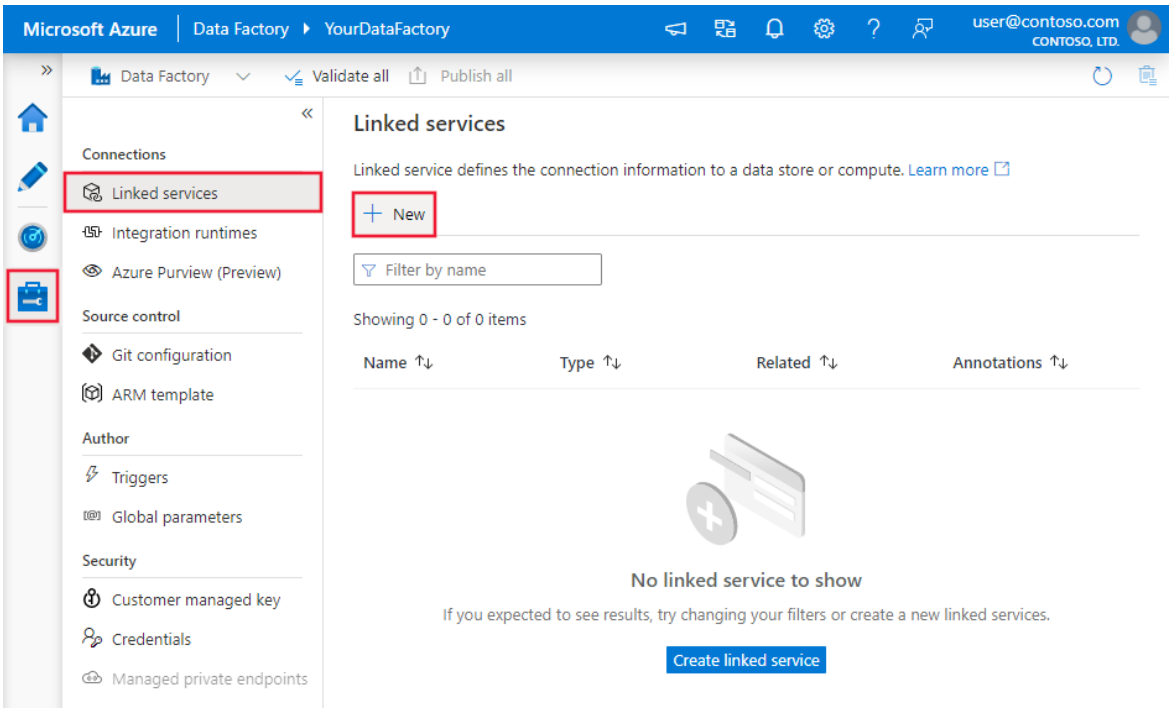
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to MongoDB using UI

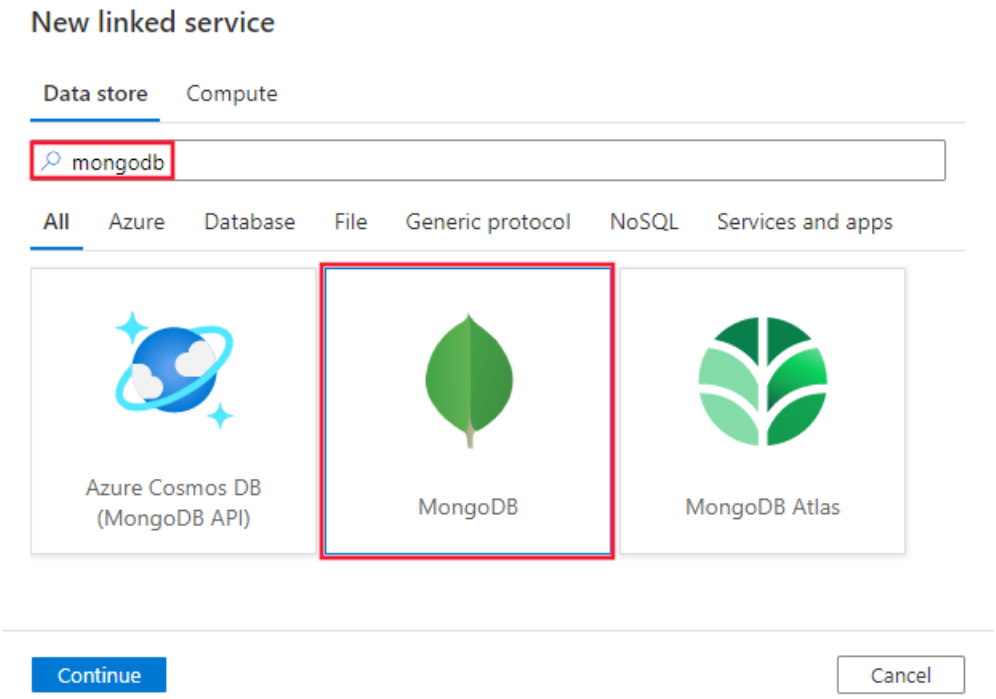
Use the following steps to create a linked service to MongoDB in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

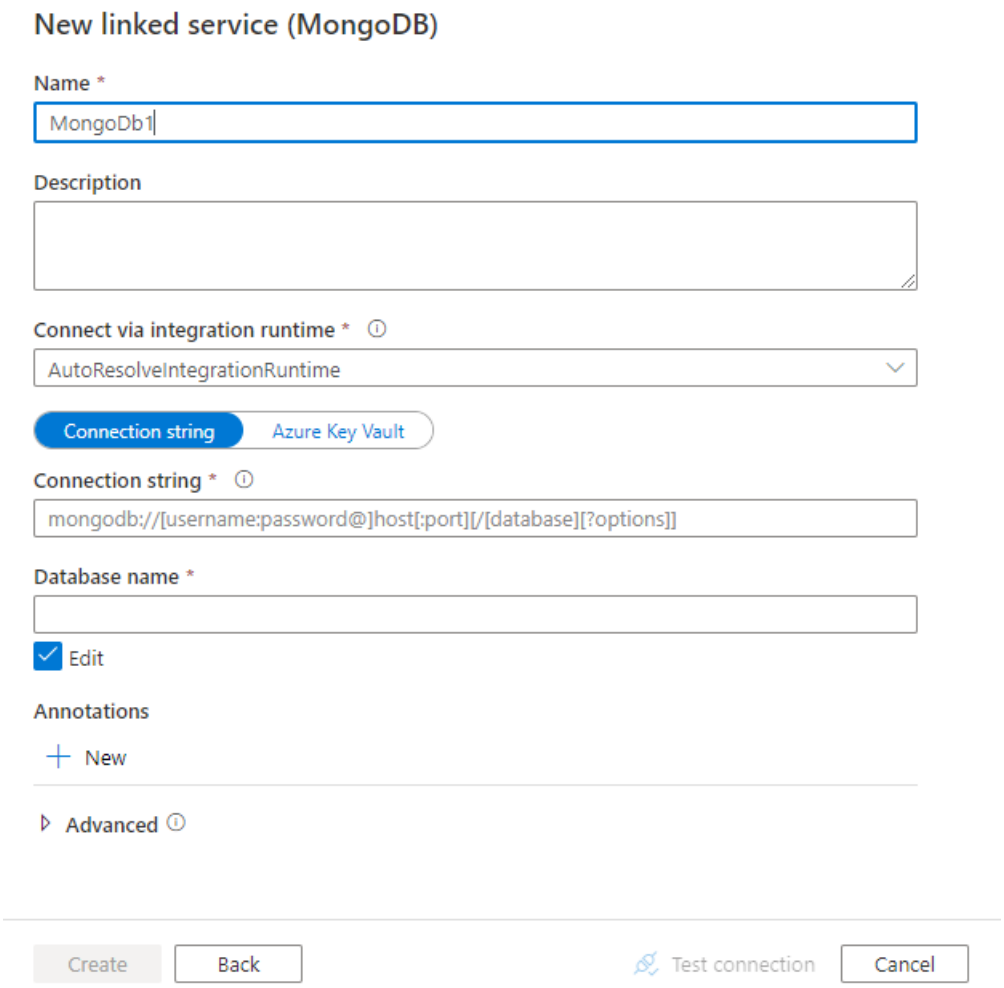
   - Azure Data Factory
   - Azure Synapse



2. Search for MongoDB and select the MongoDB connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to MongoDB connector.

Linked service properties

The following properties are supported for MongoDB linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **MongoDbV2**	Yes
connectionString	Specify the MongoDB connection string e.g. `mongodb://[username:password@]host[:port][/[database][?options]]`. Refer to MongoDB manual on connection string for more details. You can also put a connection string in Azure Key Vault. Refer to Store credentials in Azure Key Vault with more details.	Yes

PROPERTY	DESCRIPTION	REQUIRED
database	Name of the database that you want to access.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "MongoDBLinkedService",
    "properties": {
        "type": "MongoDbV2",
        "typeProperties": {
            "connectionString": "mongodb://[username:password@]host[:port][/[database][?options]]",
            "database": "myDatabase"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties that are available for defining datasets, see Datasets and linked services. The following properties are supported for MongoDB dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **MongoDbV2Collection**	Yes
collectionName	Name of the collection in MongoDB database.	Yes

Example:

```
{
    "name": "MongoDbDataset",
    "properties": {
        "type": "MongoDbV2Collection",
        "typeProperties": {
            "collectionName": "<Collection name>"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<MongoDB linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by MongoDB source and sink.

MongoDB as source

The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **MongoDbV2Source**	Yes
filter	Specifies selection filter using query operators. To return all documents in a collection, omit this parameter or pass an empty document ({}).	No
cursorMethods.project	Specifies the fields to return in the documents for projection. To return all fields in the matching documents, omit this parameter.	No
cursorMethods.sort	Specifies the order in which the query returns matching documents. Refer to cursor.sort().	No
cursorMethods.limit	Specifies the maximum number of documents the server returns. Refer to cursor.limit().	No
cursorMethods.skip	Specifies the number of documents to skip and from where MongoDB begins to return results. Refer to cursor.skip().	No
batchSize	Specifies the number of documents to return in each batch of the response from MongoDB instance. In most cases, modifying the batch size will not affect the user or the application. Cosmos DB limits each batch cannot exceed 40 MB in size, which is the sum of the batchSize number of documents' size, so decrease this value if your document size being large.	No (the default is **100**)

TIP

The service supports consuming BSON document in **Strict mode**. Make sure your filter query is in Strict mode instead of Shell mode. More description can be found at MongoDB manual.

Example:

```
"activities":[
    {
        "name": "CopyFromMongoDB",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<MongoDB input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "MongoDbV2Source",
                "filter": "{datetimeData: {$gte: ISODate(\"2018-12-11T00:00:00.000Z\"),$lt: ISODate(\"2018-
12-12T00:00:00.000Z\")}, _id: ObjectId(\"5acd7c3d0000000000000000\") }",
                "cursorMethods": {
                    "project": "{ _id : 1, name : 1, age: 1, datetimeData: 1 }",
                    "sort": "{ age : 1 }",
                    "skip": 3,
                    "limit": 3
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

MongoDB as sink

The following properties are supported in the Copy Activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the Copy Activity sink must be set to **MongoDbV2Sink**.	Yes
writeBehavior	Describes how to write data to MongoDB. Allowed values: **insert** and **upsert**. The behavior of **upsert** is to replace the document if a document with the same `_id` already exists; otherwise, insert the document. **Note**: The service automatically generates an `_id` for a document if an `_id` isn't specified either in the original document or by column mapping. This means that you must ensure that, for **upsert** to work as expected, your document has an ID.	No (the default is **insert**)

PROPERTY	DESCRIPTION	REQUIRED
writeBatchSize	The **writeBatchSize** property controls the size of documents to write in each batch. You can try increasing the value for **writeBatchSize** to improve performance and decreasing the value if your document size being large.	No (the default is **10,000**)
writeBatchTimeout	The wait time for the batch insert operation to finish before it times out. The allowed value is timespan.	No (the default is **00:30:00** - 30 minutes)

TIP

To import JSON documents as-is, refer to Import or export JSON documents section; to copy from tabular-shaped data, refer to Schema mapping.

Example

```
"activities":[
    {
        "name": "CopyToMongoDB",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Document DB output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "MongoDbV2Sink",
                "writeBehavior": "upsert"
            }
        }
    }
]
```

Import and export JSON documents

You can use this MongoDB connector to easily:

- Copy documents between two MongoDB collections as-is.
- Import JSON documents from various sources to MongoDB, including from Azure Cosmos DB, Azure Blob storage, Azure Data Lake Store, and other supported file-based stores.
- Export JSON documents from a MongoDB collection to various file-based stores.

To achieve such schema-agnostic copy, skip the "structure" (also called *schema*) section in dataset and schema mapping in copy activity.

Schema mapping

To copy data from MongoDB to tabular sink or reversed, refer to schema mapping.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from MongoDB using Azure Data Factory or Synapse Analytics (legacy)

8/19/2022 • 7 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from a MongoDB database. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> The service has released a new MongoDB connector which provides better native MongoDB support comparing to this ODBC-based implementation, refer to MongoDB connector article on details. This legacy MongoDB connector is kept supported as-is for backward compatibility, while for any new workload, please use the new connector.

Supported capabilities

You can copy data from MongoDB database to any supported sink data store. For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

Specifically, this MongoDB connector supports:

- MongoDB **versions 2.4, 2.6, 3.0, 3.2, 3.4 and 3.6**.
- Copying data using **Basic** or **Anonymous** authentication.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

The Integration Runtime provides a built-in MongoDB driver, therefore you don't need to manually install any driver when copying data from MongoDB.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK

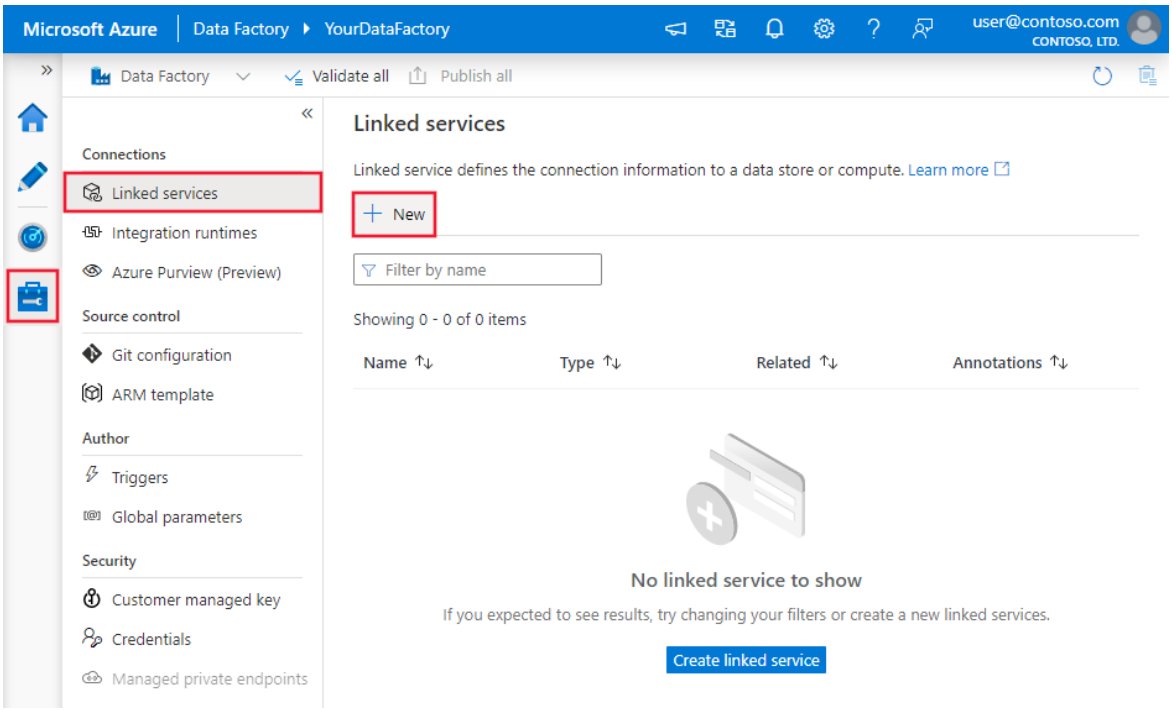
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to MongoDB using UI

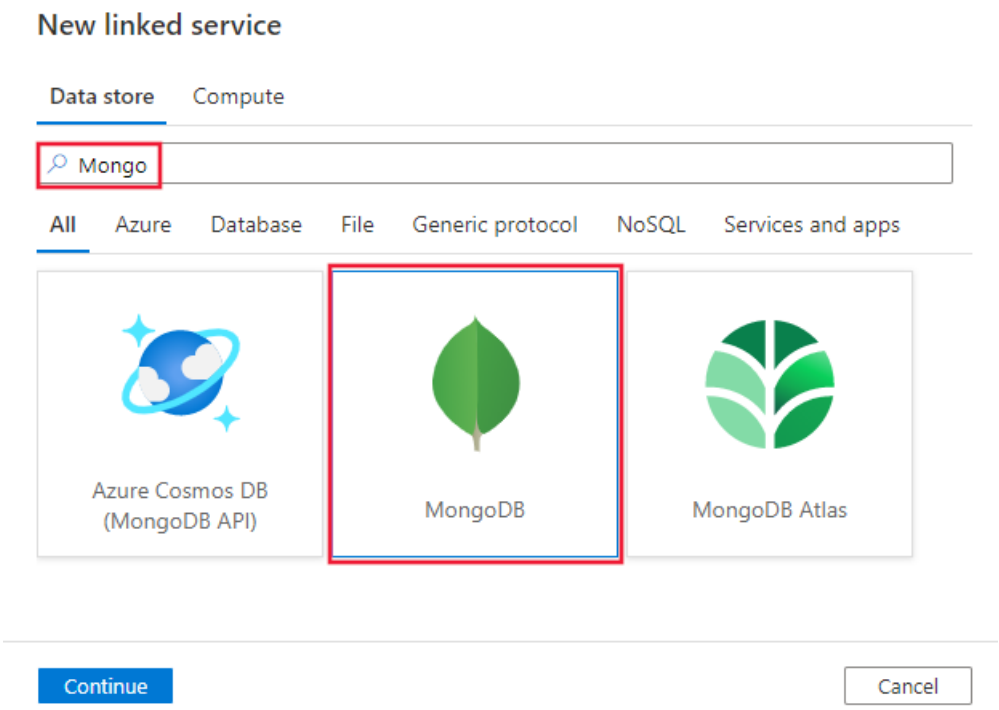
Use the following steps to create a linked service to MongoDB in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

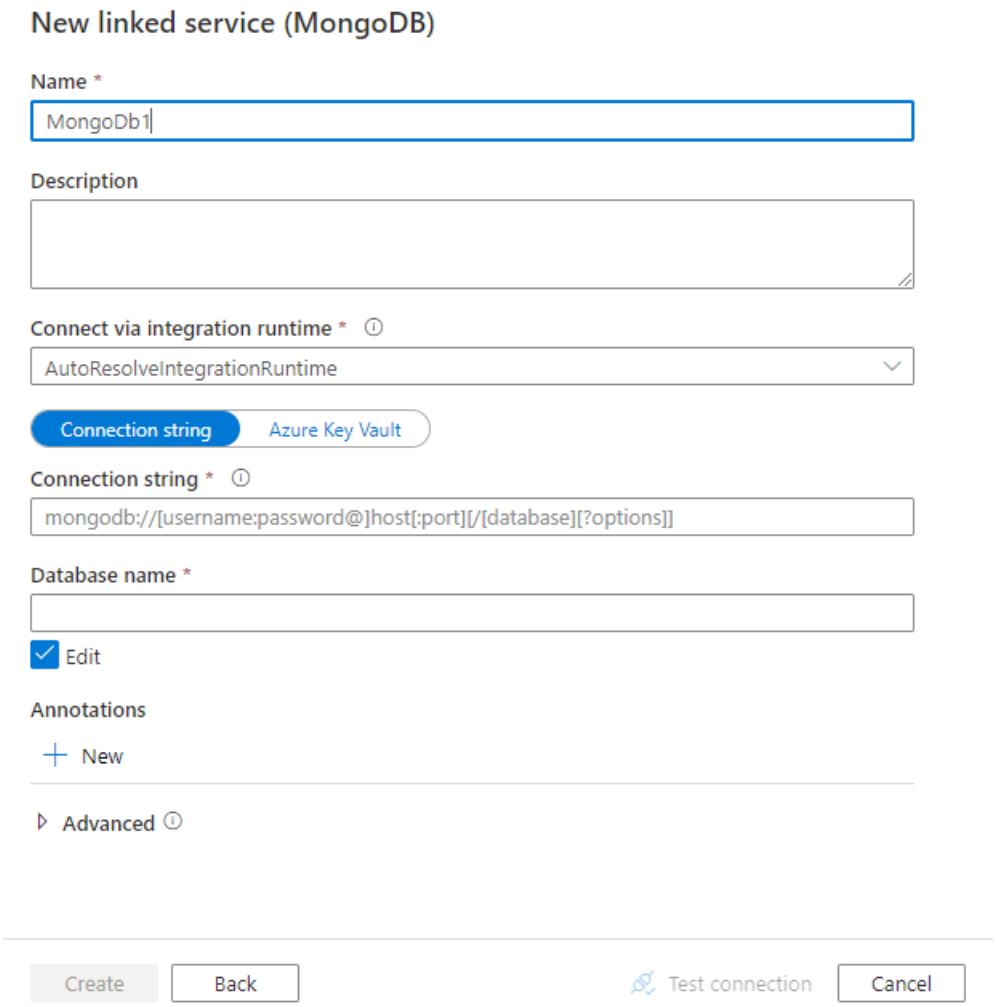
   - Azure Data Factory
   - Azure Synapse



2. Search for Mongo and select the MongoDB connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to MongoDB connector.

Linked service properties

The following properties are supported for MongoDB linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **MongoDb**	Yes
server	IP address or host name of the MongoDB server.	Yes
port	TCP port that the MongoDB server uses to listen for client connections.	No (default is 27017)
databaseName	Name of the MongoDB database that you want to access.	Yes
authenticationType	Type of authentication used to connect to the MongoDB database. Allowed values are: **Basic**, and **Anonymous**.	Yes
username	User account to access MongoDB.	Yes (if basic authentication is used).
password	Password for the user. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes (if basic authentication is used).
authSource	Name of the MongoDB database that you want to use to check your credentials for authentication.	No. For basic authentication, default is to use the admin account and the database specified using databaseName property.
enableSsl	Specifies whether the connections to the server are encrypted using TLS. The default value is false.	No
allowSelfSignedServerCert	Specifies whether to allow self-signed certificates from the server. The default value is false.	No
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "MongoDBLinkedService",
    "properties": {
        "type": "MongoDb",
        "typeProperties": {
            "server": "<server name>",
            "databaseName": "<database name>",
            "authenticationType": "Basic",
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties that are available for defining datasets, see Datasets and linked services. The following properties are supported for MongoDB dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **MongoDbCollection**	Yes
collectionName	Name of the collection in MongoDB database.	Yes

Example:

```
{
    "name": "MongoDbDataset",
    "properties": {
        "type": "MongoDbCollection",
        "linkedServiceName": {
            "referenceName": "<MongoDB linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "collectionName": "<Collection name>"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by MongoDB source.

MongoDB as source

The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **MongoDbSource**	Yes
query	Use the custom SQL-92 query to read data. For example: select * from MyTable.	No (if "collectionName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromMongoDB",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<MongoDB input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "MongoDbSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

TIP

When specify the SQL query, pay attention to the DateTime format. For example: `SELECT * FROM Account WHERE LastModifiedDate >= '2018-06-01' AND LastModifiedDate < '2018-06-02'` or to use parameter

```
SELECT * FROM Account WHERE LastModifiedDate >= '@{formatDateTime(pipeline().parameters.StartTime,'yyyy-MM-dd HH:mm:ss')}' AND LastModifiedDate < '@{formatDateTime(pipeline().parameters.EndTime,'yyyy-MM-dd HH:mm:ss')}'
```

Schema by Data Factory

Azure Data Factory service infers schema from a MongoDB collection by using the **latest 100 documents** in the collection. If these 100 documents do not contain full schema, some columns may be ignored during the copy operation.

Data type mapping for MongoDB

When copying data from MongoDB, the following mappings are used from MongoDB data types to interim data types used within the service internally. See Schema and data type mappings to learn about how copy activity

maps the source schema and data type to the sink.

MONGODB DATA TYPE	INTERIM SERVICE DATA TYPE
Binary	Byte[]
Boolean	Boolean
Date	DateTime
NumberDouble	Double
NumberInt	Int32
NumberLong	Int64
ObjectID	String
String	String
UUID	Guid
Object	Renormalized into flatten columns with "_" as nested separator

> **NOTE**
>
> To learn about support for arrays using virtual tables, refer to Support for complex types using virtual tables section.
>
> Currently, the following MongoDB data types are not supported: DBPointer, JavaScript, Max/Min key, Regular Expression, Symbol, Timestamp, Undefined.

Support for complex types using virtual tables

The service uses a built-in ODBC driver to connect to and copy data from your MongoDB database. For complex types such as arrays or objects with different types across the documents, the driver re-normalizes data into corresponding virtual tables. Specifically, if a table contains such columns, the driver generates the following virtual tables:

- A **base table**, which contains the same data as the real table except for the complex type columns. The base table uses the same name as the real table that it represents.
- A **virtual table** for each complex type column, which expands the nested data. The virtual tables are named using the name of the real table, a separator "_" and the name of the array or object.

Virtual tables refer to the data in the real table, enabling the driver to access the de-normalized data. You can access the content of MongoDB arrays by querying and joining the virtual tables.

Example

For example, ExampleTable here is a MongoDB table that has one column with an array of Objects in each cell – Invoices, and one column with an array of Scalar types – Ratings.

_ID	CUSTOMER NAME	INVOICES	SERVICE LEVEL	RATINGS
1111	ABC	[{invoice_id:"123", item:"toaster", price:"456", discount:"0.2"}, {invoice_id:"124", item:"oven", price: "1235", discount: "0.2"}]	Silver	[5,6]
2222	XYZ	[{invoice_id:"135", item:"fridge", price: "12543", discount: "0.0"}]	Gold	[1,2]

The driver would generate multiple virtual tables to represent this single table. The first virtual table is the base table named "ExampleTable", shown in the example. The base table contains all the data of the original table, but the data from the arrays has been omitted and is expanded in the virtual tables.

_ID	CUSTOMER NAME	SERVICE LEVEL
1111	ABC	Silver
2222	XYZ	Gold

The following tables show the virtual tables that represent the original arrays in the example. These tables contain the following:

- A reference back to the original primary key column corresponding to the row of the original array (via the _id column)
- An indication of the position of the data within the original array
- The expanded data for each element within the array

Table "ExampleTable_Invoices":

_ID	EXAMPLETABLE_INVOICES_DIM1_IDX	INVOICE_ID	ITEM	PRICE	DISCOUNT
1111	0	123	toaster	456	0.2
1111	1	124	oven	1235	0.2
2222	0	135	fridge	12543	0.0

Table "ExampleTable_Ratings":

_ID	EXAMPLETABLE_RATINGS_DIM1_IDX	EXAMPLETABLE_RATINGS
1111	0	5
1111	1	6
2222	0	1

_ID	EXAMPLETABLE_RATINGS_DIM1_IDX	EXAMPLETABLE_RATINGS
2222	1	2

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from or to MongoDB Atlas using Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from and to a MongoDB Atlas database. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This MongoDB Atlas connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Specifically, this MongoDB Atlas connector supports **versions up to 4.2**.

Prerequisites

If you use Azure Integration Runtime for copy, make sure you add the effective region's Azure Integration Runtime IPs to the MongoDB Atlas IP Access List.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

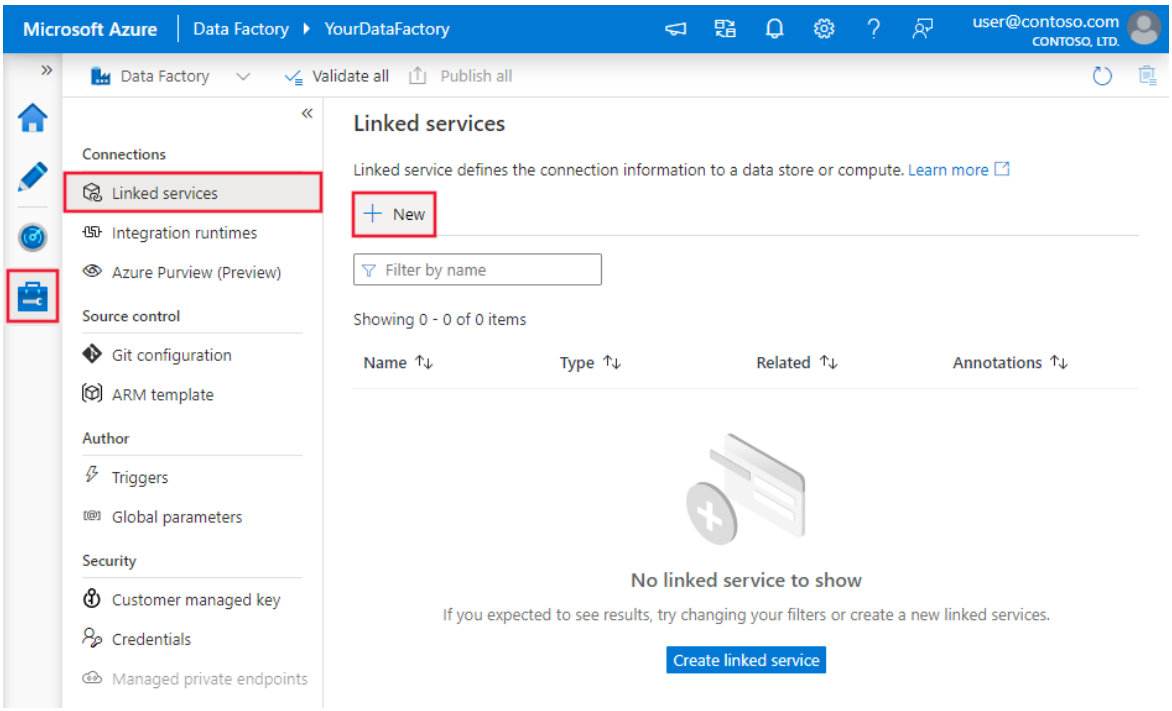
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to MongoDB Atlas using UI

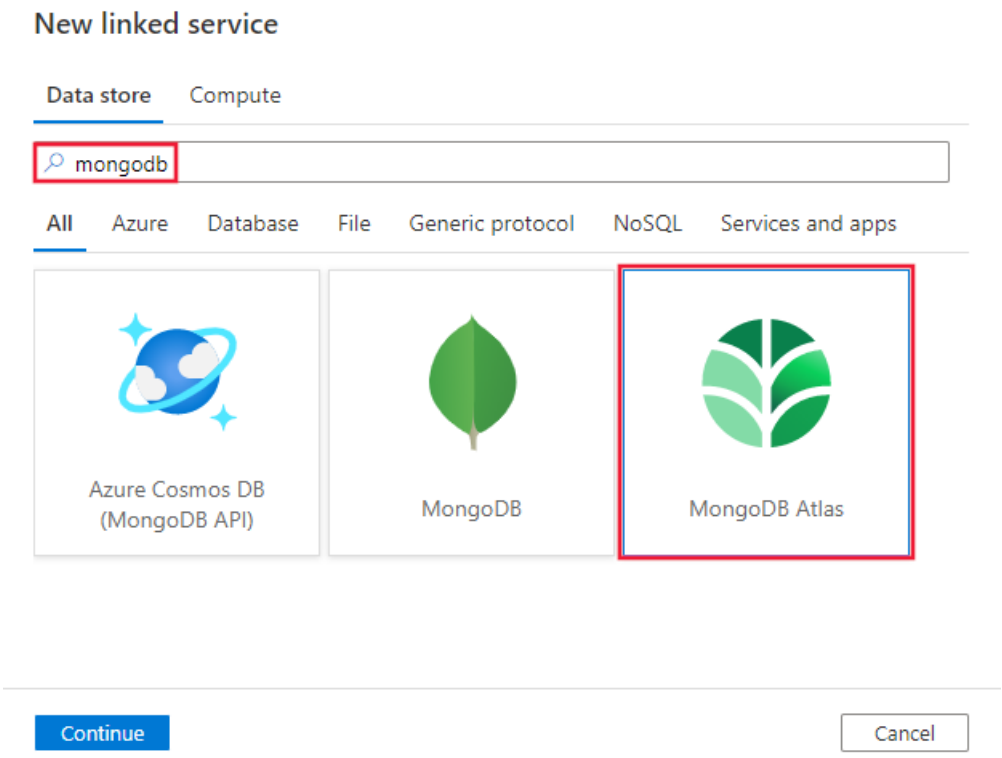
Use the following steps to create a linked service to MongoDB Atlas in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

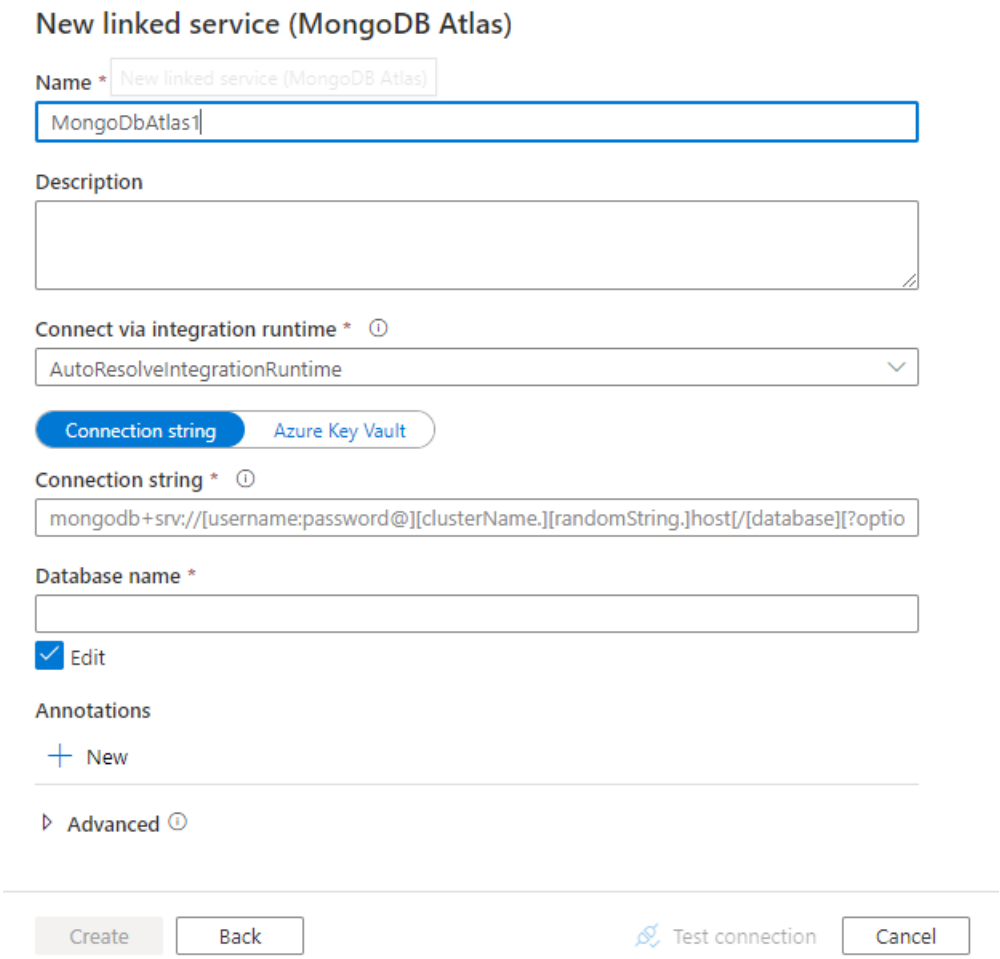
   - Azure Data Factory
   - Azure Synapse



2. Search for MongoDB and select the MongoDB Atlas connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to MongoDB Atlas connector.

Linked service properties

The following properties are supported for MongoDB Atlas linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **MongoDbAtlas**	Yes
connectionString	Specify the MongoDB Atlas connection string e.g. `mongodb+srv://<username>:<password>@<clustername>.<randomString>.<hostName>/<dbname>?<otherProperties>`. You can also put a connection string in Azure Key Vault. Refer to Store credentials in Azure Key Vault with more details.	Yes
database	Name of the database that you want to access.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "MongoDbAtlasLinkedService",
    "properties": {
        "type": "MongoDbAtlas",
        "typeProperties": {
            "connectionString": "mongodb+srv://<username>:<password>@<clustername>.<randomString>.
<hostName>/<dbname>?<otherProperties>",
            "database": "myDatabase"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties that are available for defining datasets, see Datasets and linked services. The following properties are supported for MongoDB Atlas dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **MongoDbAtlasCollection**	Yes
collectionName	Name of the collection in MongoDB Atlas database.	Yes

Example:

```
{
    "name": "MongoDbAtlasDataset",
    "properties": {
        "type": "MongoDbAtlasCollection",
        "typeProperties": {
            "collectionName": "<Collection name>"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<MongoDB Atlas linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by MongoDB Atlas source and sink.

MongoDB Atlas as source

The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **MongoDbAtlasSource**	Yes
filter	Specifies selection filter using query operators. To return all documents in a collection, omit this parameter or pass an empty document ({}).	No
cursorMethods.project	Specifies the fields to return in the documents for projection. To return all fields in the matching documents, omit this parameter.	No
cursorMethods.sort	Specifies the order in which the query returns matching documents. Refer to cursor.sort().	No
cursorMethods.limit	Specifies the maximum number of documents the server returns. Refer to cursor.limit().	No
cursorMethods.skip	Specifies the number of documents to skip and from where MongoDB Atlas begins to return results. Refer to cursor.skip().	No
batchSize	Specifies the number of documents to return in each batch of the response from MongoDB Atlas instance. In most cases, modifying the batch size will not affect the user or the application. Cosmos DB limits each batch cannot exceed 40MB in size, which is the sum of the batchSize number of documents' size, so decrease this value if your document size being large.	No (the default is **100**)

TIP

The service supports consuming BSON document in **Strict mode**. Make sure your filter query is in Strict mode instead of Shell mode. More description can be found at MongoDB manual.

Example:

```
"activities":[
    {
        "name": "CopyFromMongoDbAtlas",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<MongoDB Atlas input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "MongoDbAtlasSource",
                "filter": "{datetimeData: {$gte: ISODate(\"2018-12-11T00:00:00.000Z\"),$lt: ISODate(\"2018-
12-12T00:00:00.000Z\")}, _id: ObjectId(\"5acd7c3d0000000000000000\") }",
                "cursorMethods": {
                    "project": "{ _id : 1, name : 1, age: 1, datetimeData: 1 }",
                    "sort": "{ age : 1 }",
                    "skip": 3,
                    "limit": 3
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

MongoDB Atlas as sink

The following properties are supported in the Copy Activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the Copy Activity sink must be set to **MongoDbAtlasSink**.	Yes
writeBehavior	Describes how to write data to MongoDB Atlas. Allowed values: **insert** and **upsert**. The behavior of **upsert** is to replace the document if a document with the same `_id` already exists; otherwise, insert the document. **Note**: The service automatically generates an `_id` for a document if an `_id` isn't specified either in the original document or by column mapping. This means that you must ensure that, for **upsert** to work as expected, your document has an ID.	No (the default is **insert**)

PROPERTY	DESCRIPTION	REQUIRED
writeBatchSize	The **writeBatchSize** property controls the size of documents to write in each batch. You can try increasing the value for **writeBatchSize** to improve performance and decreasing the value if your document size being large.	No (the default is **10,000**)
writeBatchTimeout	The wait time for the batch insert operation to finish before it times out. The allowed value is timespan.	No (the default is **00:30:00** - 30 minutes)

TIP

To import JSON documents as-is, refer to Import or export JSON documents section; to copy from tabular-shaped data, refer to Schema mapping.

Example

```
"activities":[
    {
        "name": "CopyToMongoDBAtlas",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Document DB output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "MongoDbAtlasSink",
                "writeBehavior": "upsert"
            }
        }
    }
]
```

Import and Export JSON documents

You can use this MongoDB Atlas connector to easily:

- Copy documents between two MongoDB Atlas collections as-is.
- Import JSON documents from various sources to MongoDB Atlas, including from Azure Cosmos DB, Azure Blob storage, Azure Data Lake Store, and other supported file-based stores.
- Export JSON documents from a MongoDB Atlas collection to various file-based stores.

To achieve such schema-agnostic copy, skip the "structure" (also called *schema*) section in dataset and schema mapping in copy activity.

Schema mapping

To copy data from MongoDB Atlas to tabular sink or reversed, refer to schema mapping.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from MySQL using Azure Data Factory or Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from a MySQL database. It builds on the copy activity overview article that presents a general overview of copy activity.

> **NOTE**
> To copy data from or to Azure Database for MySQL service, use the specialized Azure Database for MySQL connector.

Supported capabilities

This MySQL connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

Specifically, this MySQL connector supports MySQL **version 5.6, 5.7 and 8.0**.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

The Integration Runtime provides a built-in MySQL driver starting from version 3.7, therefore you don't need to manually install any driver.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

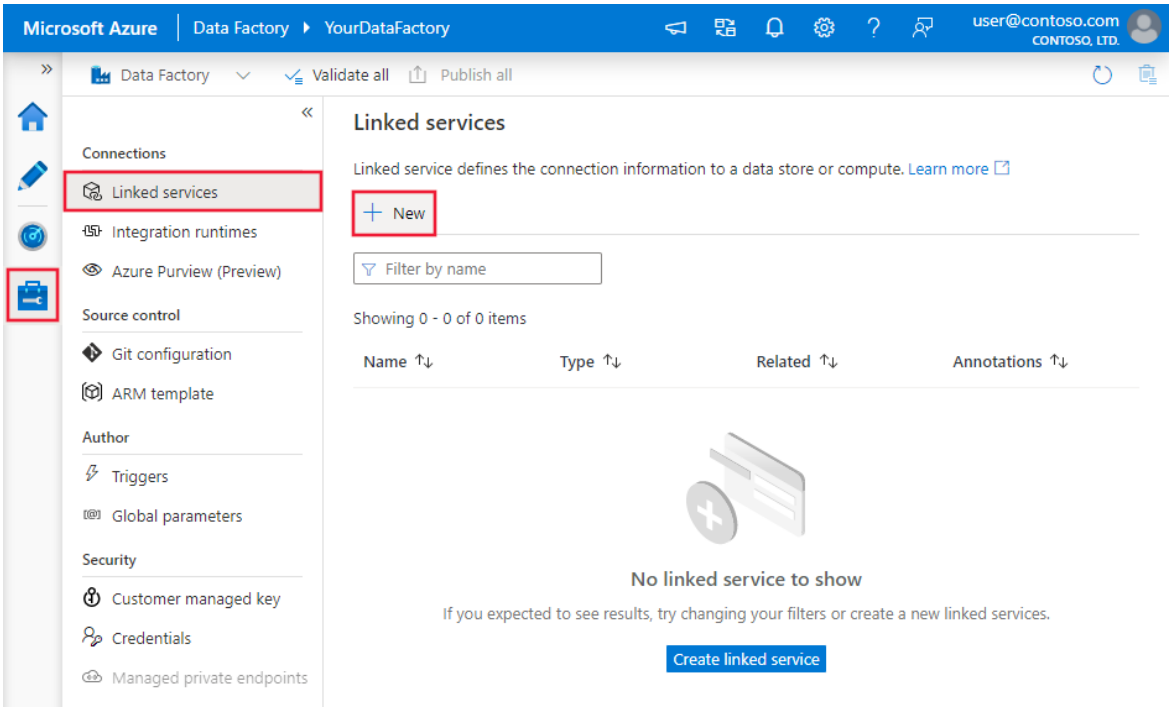
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to MySQL using UI

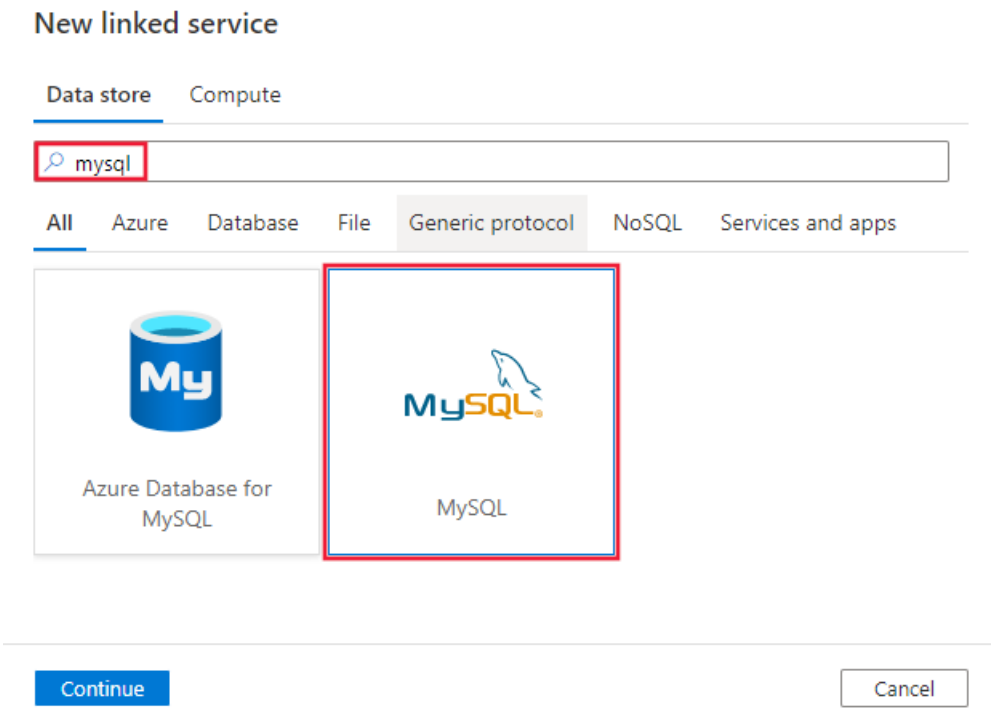
Use the following steps to create a linked service to MySQL in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

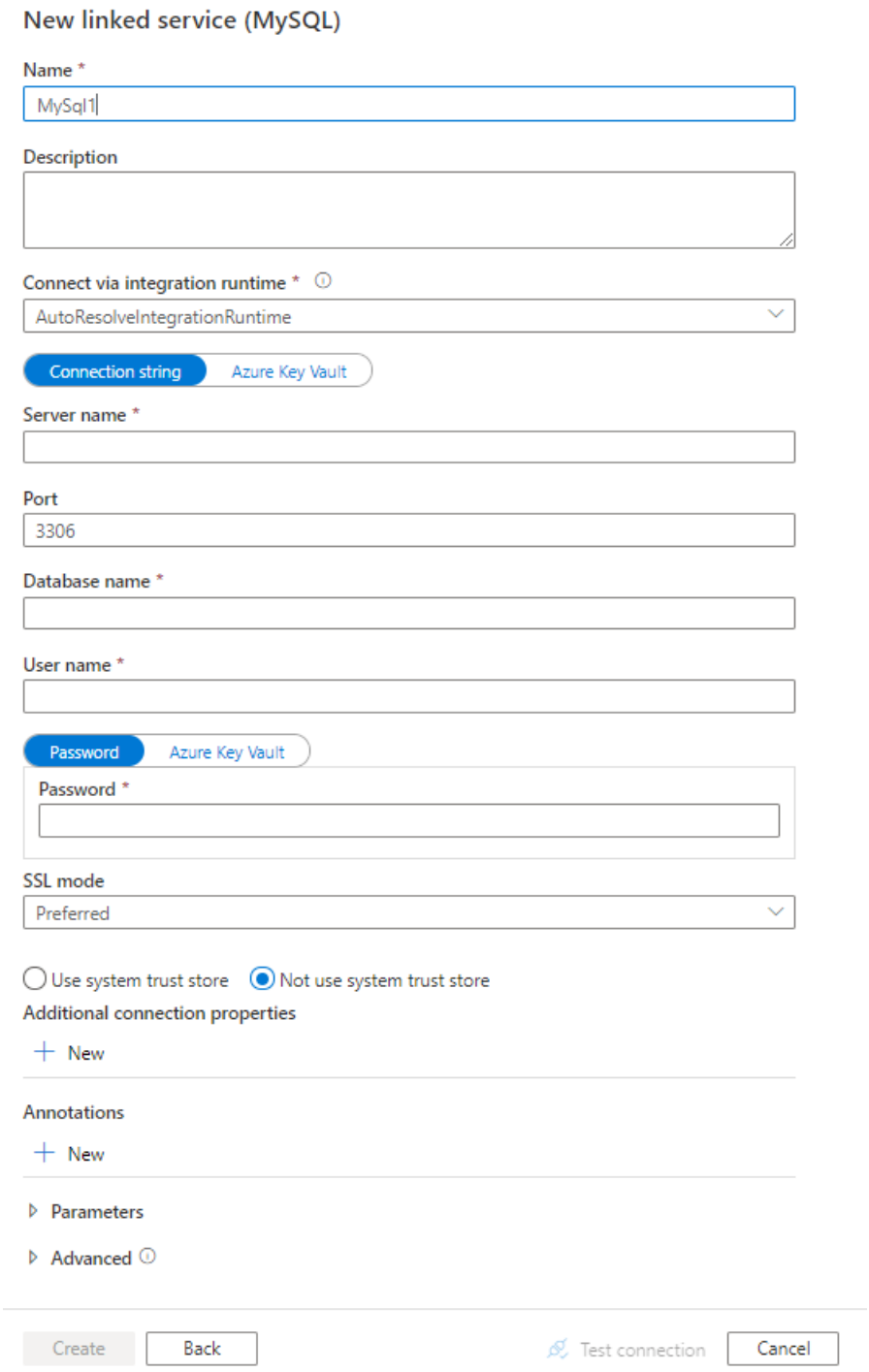
   - Azure Data Factory
   - Azure Synapse



2. Search for MySQL and select the MySQL connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to MySQL connector.

Linked service properties

The following properties are supported for MySQL linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **MySql**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	Specify information needed to connect to the Azure Database for MySQL instance. You can also put password in Azure Key Vault and pull the `password` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

A typical connection string is `Server=<server>;Port=<port>;Database=<database>;UID=<username>;PWD=<password>`.
More properties you can set per your case:

PROPERTY	DESCRIPTION	OPTIONS	REQUIRED
SSLMode	This option specifies whether the driver uses TLS encryption and verification when connecting to MySQL. E.g., `SSLMode=<0/1/2/3/4>`.	DISABLED (0) / PREFERRED (1) **(Default)** / REQUIRED (2) / VERIFY_CA (3) / VERIFY_IDENTITY (4)	No
SSLCert	The full path and name of a .pem file containing the SSL certificate used for proving the identity of the client. To specify a private key for encrypting this certificate before sending it to the server, use the `SSLKey` property.		Yes, if using two-way SSL verification.
SSLKey	The full path and name of a file containing the private key used for encrypting the client-side certificate during two-way SSL verification.		Yes, if using two-way SSL verification.
UseSystemTrustStore	This option specifies whether to use a CA certificate from the system trust store, or from a specified PEM file. E.g. `UseSystemTrustStore=<0/1>;`	Enabled (1) / Disabled (0) **(Default)**	No

Example:

```
{
    "name": "MySQLLinkedService",
    "properties": {
        "type": "MySql",
        "typeProperties": {
            "connectionString": "Server=<server>;Port=<port>;Database=<database>;UID=<username>;PWD=
<password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "MySQLLinkedService",
    "properties": {
        "type": "MySql",
        "typeProperties": {
            "connectionString": "Server=<server>;Port=<port>;Database=<database>;UID=<username>;",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

If you were using MySQL linked service with the following payload, it is still supported as-is, while you are suggested to use the new one going forward.

Previous payload:

```
{
    "name": "MySQLLinkedService",
    "properties": {
        "type": "MySql",
        "typeProperties": {
            "server": "<server>",
            "database": "<database>",
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by MySQL dataset.

To copy data from MySQL, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **MySqlTable**	Yes
tableName	Name of the table in the MySQL database.	No (if "query" in activity source is specified)

Example

```
{
    "name": "MySQLDataset",
    "properties":
    {
        "type": "MySqlTable",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<MySQL linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

If you were using `RelationalTable` typed dataset, it is still supported as-is, while you are suggested to use the new one going forward.

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by MySQL source.

MySQL as source

To copy data from MySQL, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **MySqlSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromMySQL",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<MySQL input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "MySqlSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

If you were using `RelationalSource` typed source, it is still supported as-is, while you are suggested to use the new one going forward.

Data type mapping for MySQL

When copying data from MySQL, the following mappings are used from MySQL data types to interim data types used by the service internally. See Schema and data type mappings to learn about how copy activity maps the source schema and data type to the sink.

MYSQL DATA TYPE	INTERIM SERVICE DATA TYPE
`bigint`	`Int64`
`bigint unsigned`	`Decimal`
`bit(1)`	`Boolean`

MYSQL DATA TYPE	INTERIM SERVICE DATA TYPE
`bit(M), M>1`	`Byte[]`
`blob`	`Byte[]`
`bool`	`Int16`
`char`	`String`
`date`	`Datetime`
`datetime`	`Datetime`
`decimal`	`Decimal, String`
`double`	`Double`
`double precision`	`Double`
`enum`	`String`
`float`	`Single`
`int`	`Int32`
`int unsigned`	`Int64`
`integer`	`Int32`
`integer unsigned`	`Int64`
`long varbinary`	`Byte[]`
`long varchar`	`String`
`longblob`	`Byte[]`
`longtext`	`String`
`mediumblob`	`Byte[]`
`mediumint`	`Int32`
`mediumint unsigned`	`Int64`
`mediumtext`	`String`
`numeric`	`Decimal`

MYSQL DATA TYPE	INTERIM SERVICE DATA TYPE
`real`	`Double`
`set`	`String`
`smallint`	`Int16`
`smallint unsigned`	`Int32`
`text`	`String`
`time`	`TimeSpan`
`timestamp`	`Datetime`
`tinyblob`	`Byte[]`
`tinyint`	`Int16`
`tinyint unsigned`	`Int16`
`tinytext`	`String`
`varchar`	`String`
`year`	`Int`

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Netezza by using Azure Data Factory or Synapse Analytics

8/19/2022 • 8 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory or Synapse Analytics pipelines to copy data from Netezza. The article builds on Copy Activity, which presents a general overview of Copy Activity.

> **TIP**
> For data migration scenario from Netezza to Azure, learn more from Migrate data from on-premises Netezza server to Azure.

Supported capabilities

This Netezza connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that Copy Activity supports as sources and sinks, see Supported data stores and formats.

Netezza connector supports parallel copying from source. See the Parallel copy from Netezza section for details.

The service provides a built-in driver to enable connectivity. You don't need to manually install any driver to use this connector.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Get started

You can create a pipeline that uses a copy activity by using the .NET SDK, the Python SDK, Azure PowerShell, the

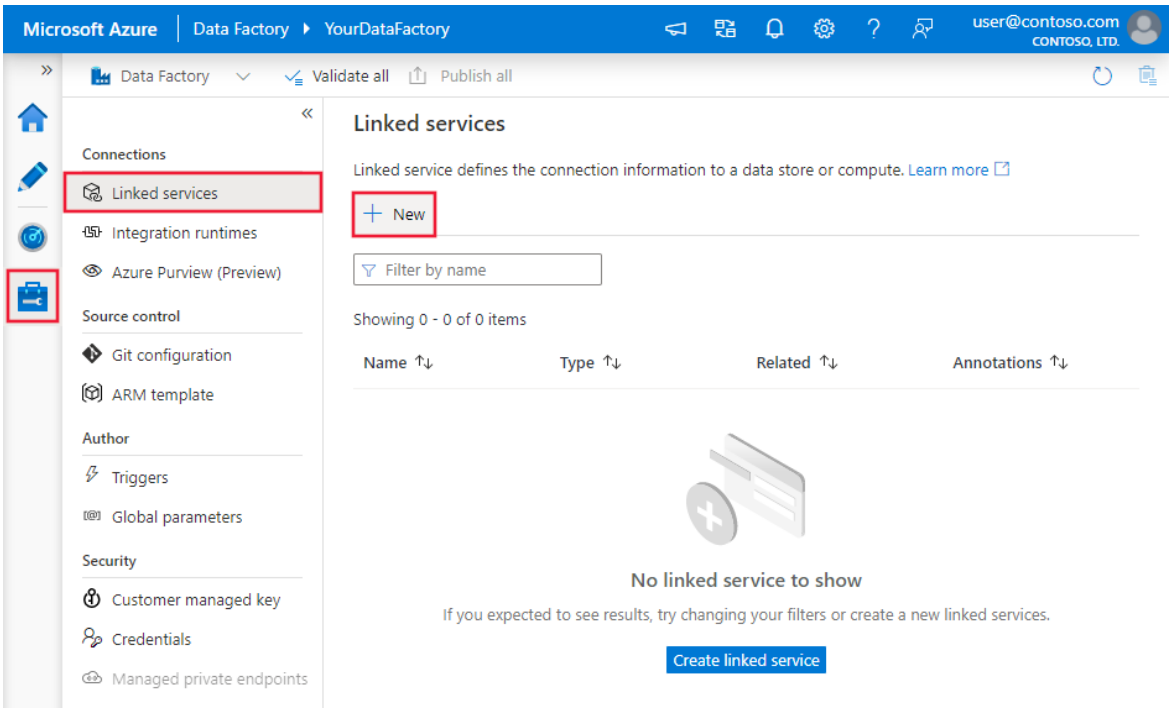
REST API, or an Azure Resource Manager template. See the Copy Activity tutorialfor step-by-step instructions to create a pipeline with a copy activity.

Create a linked service to Netezza using UI

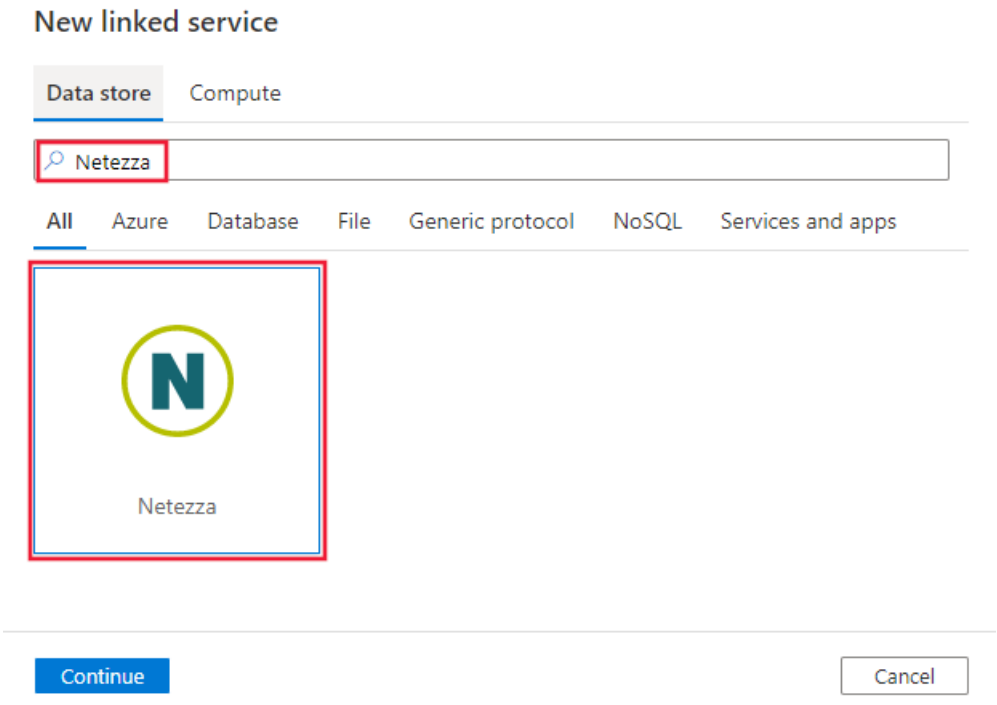
Use the following steps to create a linked service to Netezza in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

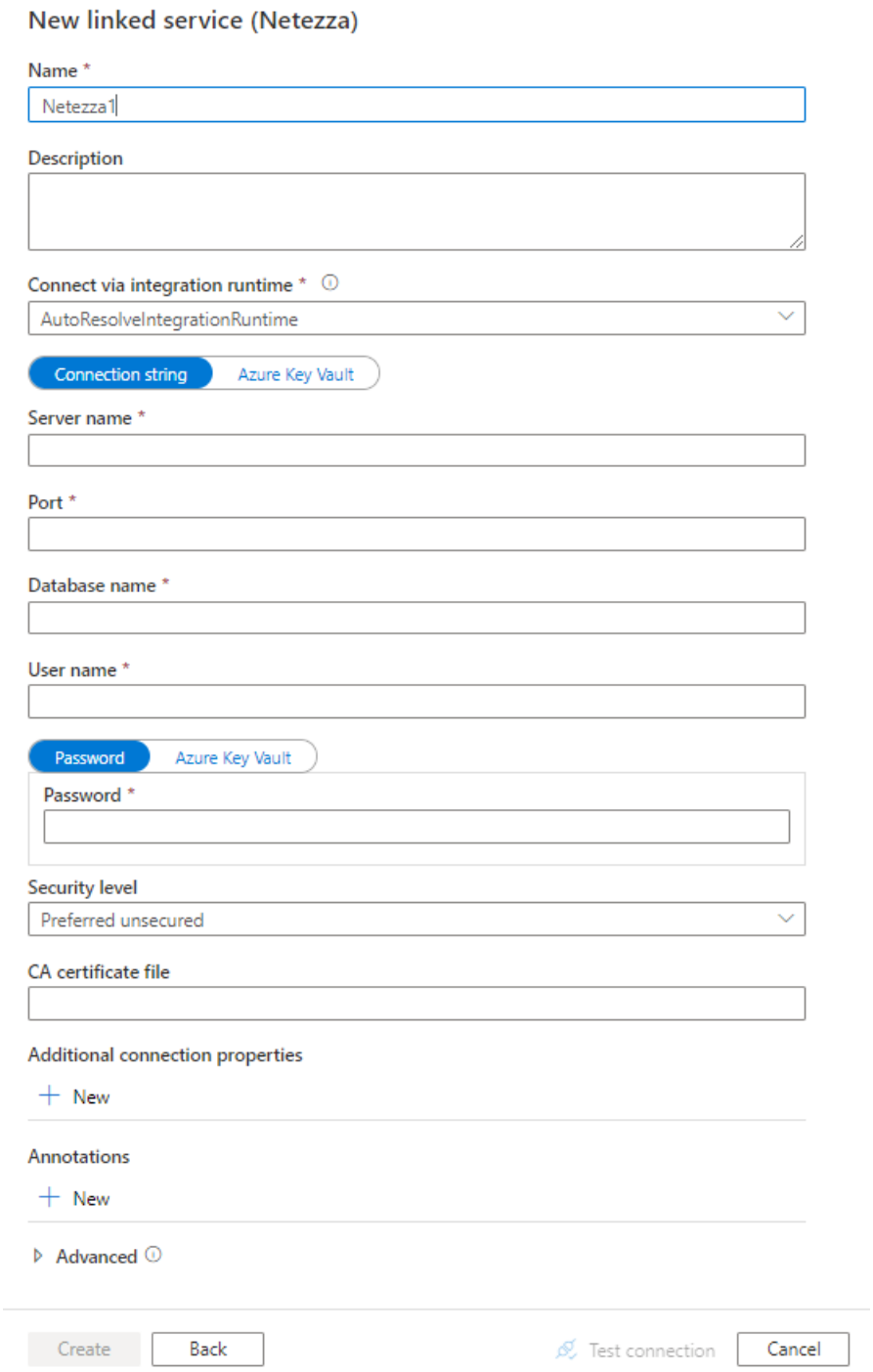
   - Azure Data Factory
   - Azure Synapse



2. Search for Netezza and select the Netezza connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties you can use to define entities that are specific to the Netezza connector.

Linked service properties

The following properties are supported for the Netezza linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **Netezza**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	An ODBC connection string to connect to Netezza. You can also put password in Azure Key Vault and pull the `pwd` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to use to connect to the data store. Learn more from Prerequisites section. If not specified, the default Azure Integration Runtime is used.	No

A typical connection string is `Server=<server>;Port=<port>;Database=<database>;UID=<user name>;PWD=<password>`.
The following table describes more properties that you can set:

PROPERTY	DESCRIPTION	REQUIRED
SecurityLevel	The level of security that the driver uses for the connection to the data store. The driver supports SSL connections with one-way authentication using SSL version 3. Example: `SecurityLevel=preferredSecured`. Supported values are: - **Only unsecured (onlyUnSecured)**: The driver doesn't use SSL. - **Preferred unsecured (preferredUnSecured) (default)**: If the server provides a choice, the driver doesn't use SSL. - **Preferred secured (preferredSecured)**: If the server provides a choice, the driver uses SSL. - **Only secured (onlySecured)**: The driver doesn't connect unless an SSL connection is available.	No
CaCertFile	The full path to the SSL certificate that's used by the server. Example: `CaCertFile=<cert path>;`	Yes, if SSL is enabled

Example

```
{
    "name": "NetezzaLinkedService",
    "properties": {
        "type": "Netezza",
        "typeProperties": {
            "connectionString": "Server=<server>;Port=<port>;Database=<database>;UID=<user name>;PWD=
<password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "NetezzaLinkedService",
    "properties": {
        "type": "Netezza",
        "typeProperties": {
            "connectionString": "Server=<server>;Port=<port>;Database=<database>;UID=<user name>;",
            "pwd": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

This section provides a list of properties that the Netezza dataset supports.

For a full list of sections and properties that are available for defining datasets, see Datasets.

To copy data from Netezza, set the **type** property of the dataset to **NetezzaTable**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **NetezzaTable**	Yes
schema	Name of the schema.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)

PROPERTY	DESCRIPTION	REQUIRED
tableName	Name of the table with schema. This property is supported for backward compatibility. Use `schema` and `table` for new workload.	No (if "query" in activity source is specified)

Example

```
{
    "name": "NetezzaDataset",
    "properties": {
        "type": "NetezzaTable",
        "linkedServiceName": {
            "referenceName": "<Netezza linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {}
    }
}
```

Copy Activity properties

This section provides a list of properties that the Netezza source supports.

For a full list of sections and properties that are available for defining activities, see Pipelines.

Netezza as source

> **TIP**
> To load data from Netezza efficiently by using data partitioning, learn more from Parallel copy from Netezza section.

To copy data from Netezza, set the **source** type in Copy Activity to **NetezzaSource**. The following properties are supported in the Copy Activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the Copy Activity source must be set to **NetezzaSource**.	Yes
query	Use the custom SQL query to read data. Example: `"SELECT * FROM MyTable"`	No (if "tableName" in dataset is specified)
partitionOptions	Specifies the data partitioning options used to load data from Netezza. Allow values are: **None** (default), **DataSlice**, and **DynamicRange**. When a partition option is enabled (that is, not `None`), the degree of parallelism to concurrently load data from a Netezza database is controlled by `parallelCopies` setting on the copy activity.	No

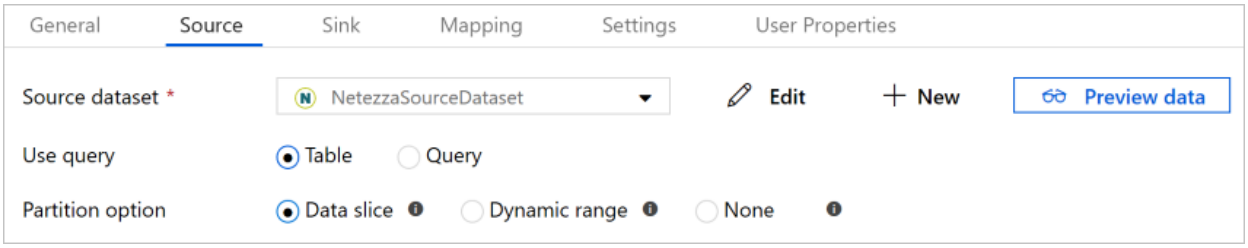
PROPERTY	DESCRIPTION	REQUIRED
partitionSettings	Specify the group of the settings for data partitioning. Apply when partition option isn't `None`.	No
partitionColumnName	Specify the name of the source column **in integer type** that will be used by range partitioning for parallel copy. If not specified, the primary key of the table is autodetected and used as the partition column. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfRangePartitionColumnName` in WHERE clause. See example in Parallel copy from Netezza section.	No
partitionUpperBound	The maximum value of the partition column to copy data out. Apply when partition option is `DynamicRange`. If you use query to retrieve source data, hook `?AdfRangePartitionUpbound` in the WHERE clause. For an example, see the Parallel copy from Netezza section.	No
partitionLowerBound	The minimum value of the partition column to copy data out. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfRangePartitionLowbound` in the WHERE clause. For an example, see the Parallel copy from Netezza section.	No

Example:

```
"activities":[
    {
        "name": "CopyFromNetezza",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Netezza input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "NetezzaSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Parallel copy from Netezza

The Data Factory Netezza connector provides built-in data partitioning to copy data from Netezza in parallel. You can find data partitioning options on the **Source** table of the copy activity.



When you enable partitioned copy, the service runs parallel queries against your Netezza source to load data by partitions. The parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your Netezza database.

You are suggested to enable parallel copy with data partitioning especially when you load large amount of data from your Netezza database. The following are suggested configurations for different scenarios. When copying data into file-based data store, it's recommanded to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

SCENARIO	SUGGESTED SETTINGS
Full load from large table.	**Partition option**: Data Slice. During execution, the service automatically partitions the data based on Netezza's built-in data slices, and copies data by partitions.

SCENARIO	SUGGESTED SETTINGS
Load large amount of data by using a custom query.	**Partition option**: Data Slice. **Query**: `SELECT * FROM <TABLENAME> WHERE mod(datasliceid, ?AdfPartitionCount) = ?AdfDataSliceCondition AND <your_additional_where_clause>` . During execution, the service replaces `?AdfPartitionCount` (with parallel copy number set on copy activity) and `?AdfDataSliceCondition` with the data slice partition logic, and sends to Netezza.
Load large amount of data by using a custom query, having an integer column with evenly distributed value for range partitioning.	**Partition options**: Dynamic range partition. **Query**: `SELECT * FROM <TABLENAME> WHERE ?AdfRangePartitionColumnName <= ? AdfRangePartitionUpbound AND ? AdfRangePartitionColumnName >= ? AdfRangePartitionLowbound AND <your_additional_where_clause>` . **Partition column**: Specify the column used to partition data. You can partition against the column with integer data type. **Partition upper bound** and **partition lower bound**: Specify if you want to filter against the partition column to retrieve data only between the lower and upper range. During execution, the service replaces `?AdfRangePartitionColumnName`, `?AdfRangePartitionUpbound`, and `?AdfRangePartitionLowbound` with the actual column name and value ranges for each partition, and sends to Netezza. For example, if your partition column "ID" set with the lower bound as 1 and the upper bound as 80, with parallel copy set as 4, the service retrieves data by 4 partitions. Their IDs are between [1,20], [21, 40], [41, 60], and [61, 80], respectively.

Example: query with data slice partition

```
"source": {
    "type": "NetezzaSource",
    "query":"SELECT * FROM <TABLENAME> WHERE mod(datasliceid, ?AdfPartitionCount) = ?AdfDataSliceCondition
AND <your_additional_where_clause>",
    "partitionOption": "DataSlice"
}
```

Example: query with dynamic range partition

```
"source": {
    "type": "NetezzaSource",
    "query":"SELECT * FROM <TABLENAME> WHERE ?AdfRangePartitionColumnName <= ?AdfRangePartitionUpbound AND ?
AdfRangePartitionColumnName >= ?AdfRangePartitionLowbound AND <your_additional_where_clause>",
    "partitionOption": "DynamicRange",
    "partitionSettings": {
        "partitionColumnName": "<dynamic_range_partition_column_name>",
        "partitionUpperBound": "<upper_value_of_partition_column>",
        "partitionLowerBound": "<lower_value_of_partition_column>"
    }
}
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores that Copy Activity supports as sources and sinks, see Supported data stores and formats.

Copy data from an OData source by using Azure Data Factory or Synapse Analytics

8/19/2022 • 8 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from an OData source. The article builds on Copy Activity, which presents a general overview of Copy Activity.

Supported capabilities

This OData connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see Supported data stores.

Specifically, this OData connector supports:

- OData version 3.0 and 4.0.
- Copying data by using one of the following authentications: **Anonymous**, **Basic**, **Windows**, and **Azure Active Directory service principal**.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK

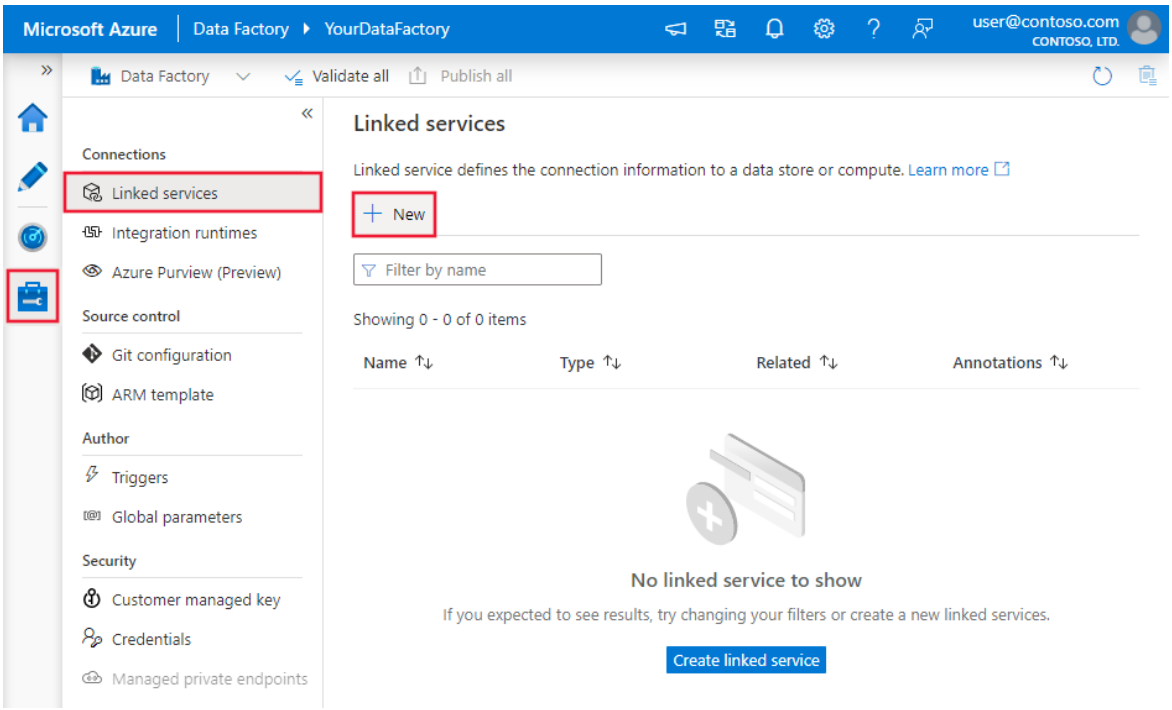
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to an OData store using UI

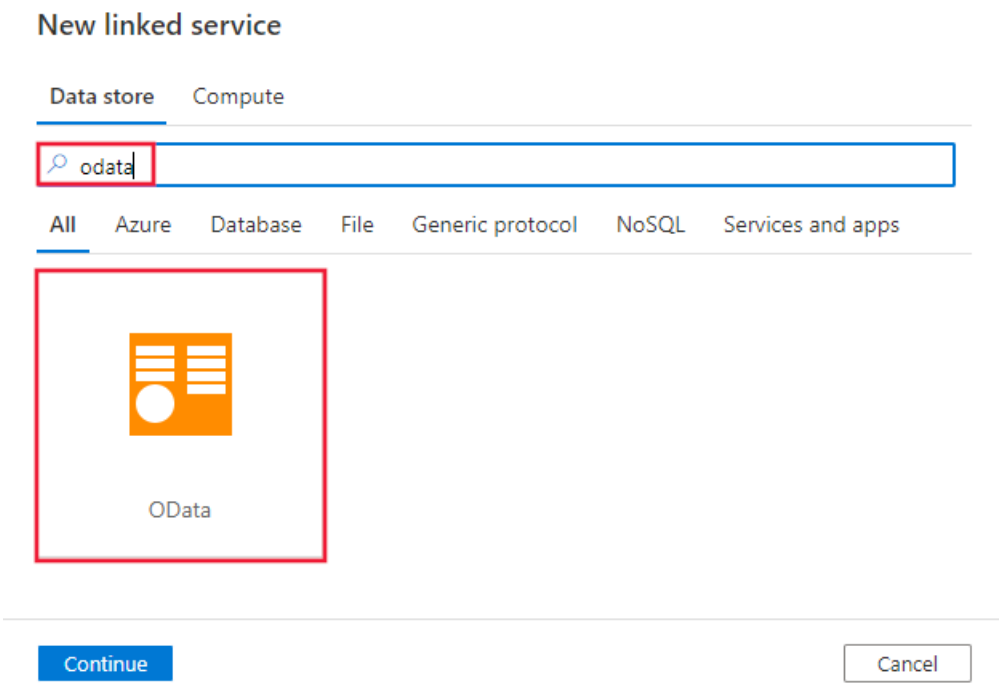
Use the following steps to create a linked service to an OData store in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then select New:

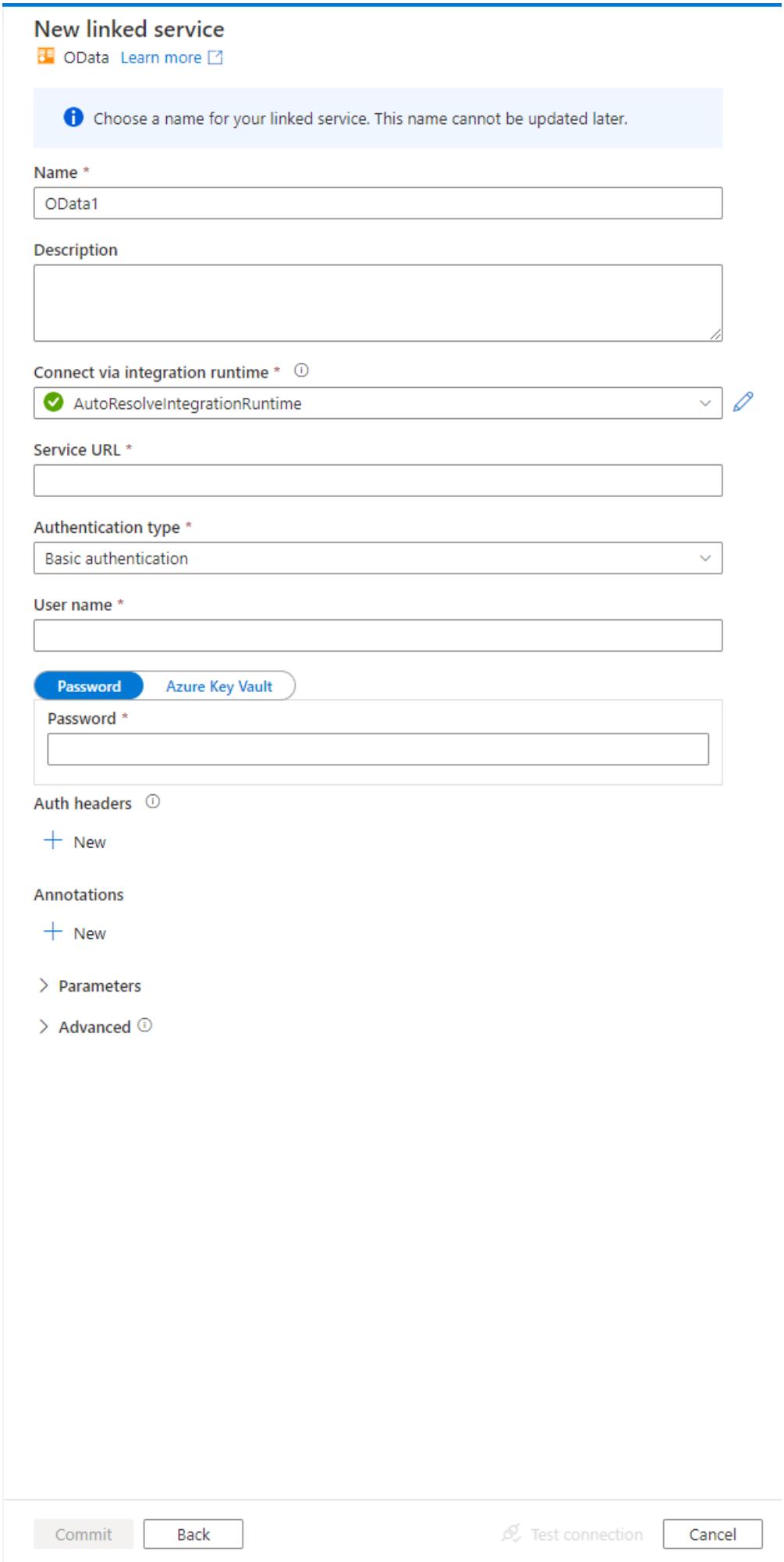
   - Azure Data Factory
   - Azure Synapse



2. Search for OData and select the OData connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties you can use to define Data Factory entities that are specific to an OData connector.

Linked service properties

The following properties are supported for an OData linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **OData**.	Yes
url	The root URL of the OData service.	Yes
authenticationType	The type of authentication used to connect to the OData source. Allowed values are **Anonymous**, **Basic**, **Windows**, and **AadServicePrincipal**. User-based OAuth isn't supported. You can additionally configure authentication headers in `authHeader` property.	Yes
authHeaders	Additional HTTP request headers for authentication. For example, to use API key authentication, you can select authentication type as "Anonymous" and specify API key in the header.	No
userName	Specify **userName** if you use Basic or Windows authentication.	No
password	Specify **password** for the user account you specified for **userName**. Mark this field as a **SecureString** type to store it securely. You also can reference a secret stored in Azure Key Vault.	No
servicePrincipalId	Specify the Azure Active Directory application's client ID.	No
aadServicePrincipalCredentialType	Specify the credential type to use for service principal authentication. Allowed values are: `ServicePrincipalKey` or `ServicePrincipalCert`.	No
servicePrincipalKey	Specify the Azure Active Directory application's key. Mark this field as a **SecureString** to store it securely, or reference a secret stored in Azure Key Vault.	No

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalEmbeddedCert	Specify the base64 encoded certificate of your application registered in Azure Active Directory, and ensure the certificate content type is **PKCS #12**. Mark this field as a **SecureString** to store it securely, or reference a secret stored in Azure Key Vault.	No
servicePrincipalEmbeddedCertPassword	Specify the password of your certificate if your certificate is secured with a password. Mark this field as a **SecureString** to store it securely, or reference a secret stored in Azure Key Vault.	No
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. Retrieve it by hovering the mouse in the top-right corner of the Azure portal.	No
aadResourceId	Specify the Azure AD resource you are requesting for authorization.	No
azureCloudType	For service principal authentication, specify the type of Azure cloud environment to which your Azure Active Directory application is registered. Allowed values are **AzurePublic**, **AzureChina**, **AzureUsGovernment**, and **AzureGermany**. By default, the service's cloud environment is used.	No
connectVia	The Integration Runtime to use to connect to the data store. Learn more from Prerequisites section. If not specified, the default Azure Integration Runtime is used.	No

Example 1: Using Anonymous authentication

```
{
    "name": "ODataLinkedService",
    "properties": {
        "type": "OData",
        "typeProperties": {
            "url": "https://services.odata.org/OData/OData.svc",
            "authenticationType": "Anonymous"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 2: Using Basic authentication

```
{
    "name": "ODataLinkedService",
    "properties": {
        "type": "OData",
        "typeProperties": {
            "url": "<endpoint of OData source>",
            "authenticationType": "Basic",
            "userName": "<user name>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 3: Using Windows authentication

```
{
    "name": "ODataLinkedService",
    "properties": {
        "type": "OData",
        "typeProperties": {
            "url": "<endpoint of OData source>",
            "authenticationType": "Windows",
            "userName": "<domain>\\<user>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 4: Using service principal key authentication

```
{
    "name": "ODataLinkedService",
    "properties": {
        "type": "OData",
        "typeProperties": {
            "url": "<endpoint of OData source>",
            "authenticationType": "AadServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "aadServicePrincipalCredentialType": "ServicePrincipalKey",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>",
            "aadResourceId": "<AAD resource URL>"
        }
    },
    "connectVia": {
        "referenceName": "<name of Integration Runtime>",
        "type": "IntegrationRuntimeReference"
    }
}
```

Example 5: Using service principal cert authentication

```
{
    "name": "ODataLinkedService",
    "properties": {
        "type": "OData",
        "typeProperties": {
            "url": "<endpoint of OData source>",
            "authenticationType": "AadServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "aadServicePrincipalCredentialType": "ServicePrincipalCert",
            "servicePrincipalEmbeddedCert": {
                "type": "SecureString",
                "value": "<base64 encoded string of (.pfx) certificate data>"
            },
            "servicePrincipalEmbeddedCertPassword": {
                "type": "SecureString",
                "value": "<password of your certificate>"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>",
            "aadResourceId": "<AAD resource e.g. https://tenant.sharepoint.com>"
        }
    },
    "connectVia": {
        "referenceName": "<name of Integration Runtime>",
        "type": "IntegrationRuntimeReference"
    }
}
```

Example 6: Using API key authentication

```
{
    "name": "ODataLinkedService",
    "properties": {
        "type": "OData",
        "typeProperties": {
            "url": "<endpoint of OData source>",
            "authenticationType": "Anonymous",
            "authHeader": {
                "APIKey": {
                    "type": "SecureString",
                    "value": "<API key>"
                }
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

This section provides a list of properties that the OData dataset supports.

For a full list of sections and properties that are available for defining datasets, see Datasets and linked services.

To copy data from OData, set the **type** property of the dataset to **ODataResource**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the dataset must be set to **ODataResource**.	Yes
path	The path to the OData resource.	Yes

Example

```
{
    "name": "ODataDataset",
    "properties":
    {
        "type": "ODataResource",
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<OData linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties":
        {
            "path": "Products"
        }
    }
}
```

Copy Activity properties

This section provides a list of properties that the OData source supports.

For a full list of sections and properties that are available for defining activities, see Pipelines.

OData as source

To copy data from OData, the following properties are supported in the Copy Activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the Copy Activity source must be set to **ODataSource**.	Yes
query	OData query options for filtering data. Example: `"$select=Name,Description&$top=5"`. **Note**: The OData connector copies data from the combined URL: `[URL specified in linked service]/[path specified in dataset]?[query specified in copy activity source]`. For more information, see OData URL components.	No
httpRequestTimeout	The timeout (the **TimeSpan** value) for the HTTP request to get a response. This value is the timeout to get a response, not the timeout to read response data. If not specified, the default value is **00:30:00** (30 minutes).	No

Example

```
"activities":[
    {
        "name": "CopyFromOData",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<OData input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "ODataSource",
                "query": "$select=Name,Description&$top=5"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

If you were using `RelationalSource` typed source, it is still supported as-is, while you are suggested to use the new one going forward.

Data type mapping for OData

When you copy data from OData, the following mappings are used between OData data types and interim data types used within the service internally. To learn how Copy Activity maps the source schema and data type to the sink, see Schema and data type mappings.

ODATA DATA TYPE	INTERIM SERVICE DATA TYPE
Edm.Binary	Byte[]
Edm.Boolean	Bool
Edm.Byte	Byte[]
Edm.DateTime	DateTime
Edm.Decimal	Decimal
Edm.Double	Double
Edm.Single	Single
Edm.Guid	Guid
Edm.Int16	Int16
Edm.Int32	Int32
Edm.Int64	Int64
Edm.SByte	Int16
Edm.String	String
Edm.Time	TimeSpan
Edm.DateTimeOffset	DateTimeOffset

> **NOTE**
> OData complex data types (such as **Object**) aren't supported.

Copy data from Project Online

Project Online requires user-based OAuth, which is not supported by Azure Data Factory. To copy data from Project Online, you can use the OData connector and an access token obtained from tools like Postman.

Caution

The access token expires in 1 hour by default, you need to get a new access token when it expires.

1. Use **Postman** to get the access token:

   a. Navigate to **Authorization** tab on the Postman Website.

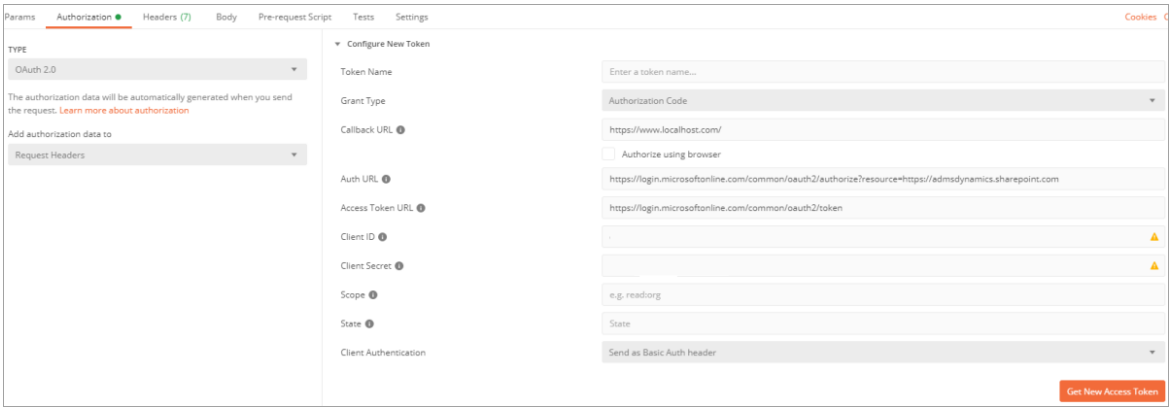
   b. In the **Type** box, select **OAuth 2.0**, and in the **Add authorization data to** box, select **Request Headers**.

   c. Fill the following information in the **Configure New Token** page to get a new access token:

   - **Grant type**: Select **Authorization Code**.
   - **Callback URL**: Enter `https://www.localhost.com/`.
   - **Auth URL**: Enter `https://login.microsoftonline.com/common/oauth2/authorize?resource=https://<your tenant name>.sharepoint.com`. Replace `<your tenant name>` with your own tenant name.
   - **Access Token URL**: Enter `https://login.microsoftonline.com/common/oauth2/token`.
   - **Client ID**: Enter your Azure Active Directory service principal ID.
   - **Client Secret**: Enter your service principal secret.
   - **Client Authentication**: Select **Send as Basic Auth header**.

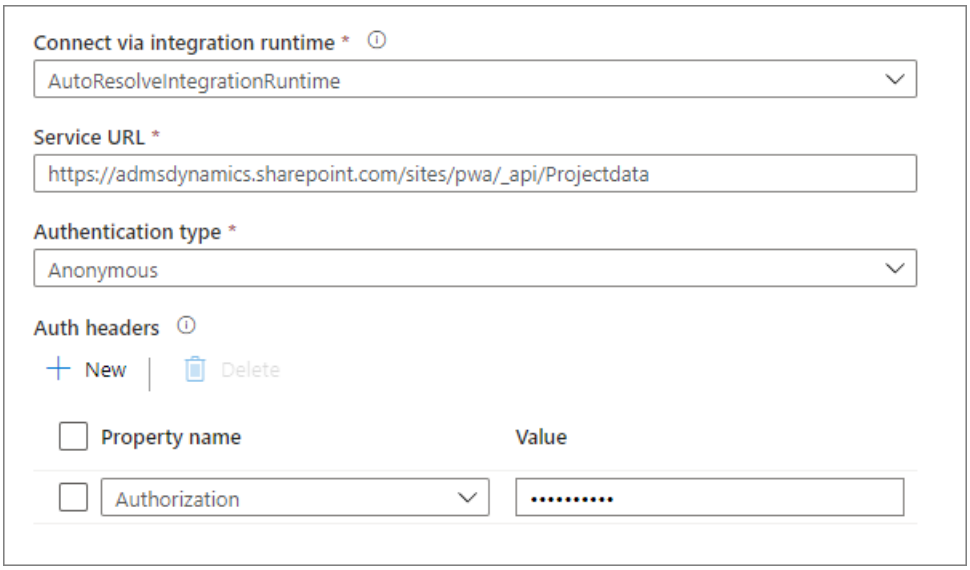
   d. You will be asked to sign in with your username and password.

   e. Once you get your access token, please copy and save it for the next step.



2. Create the OData linked service:

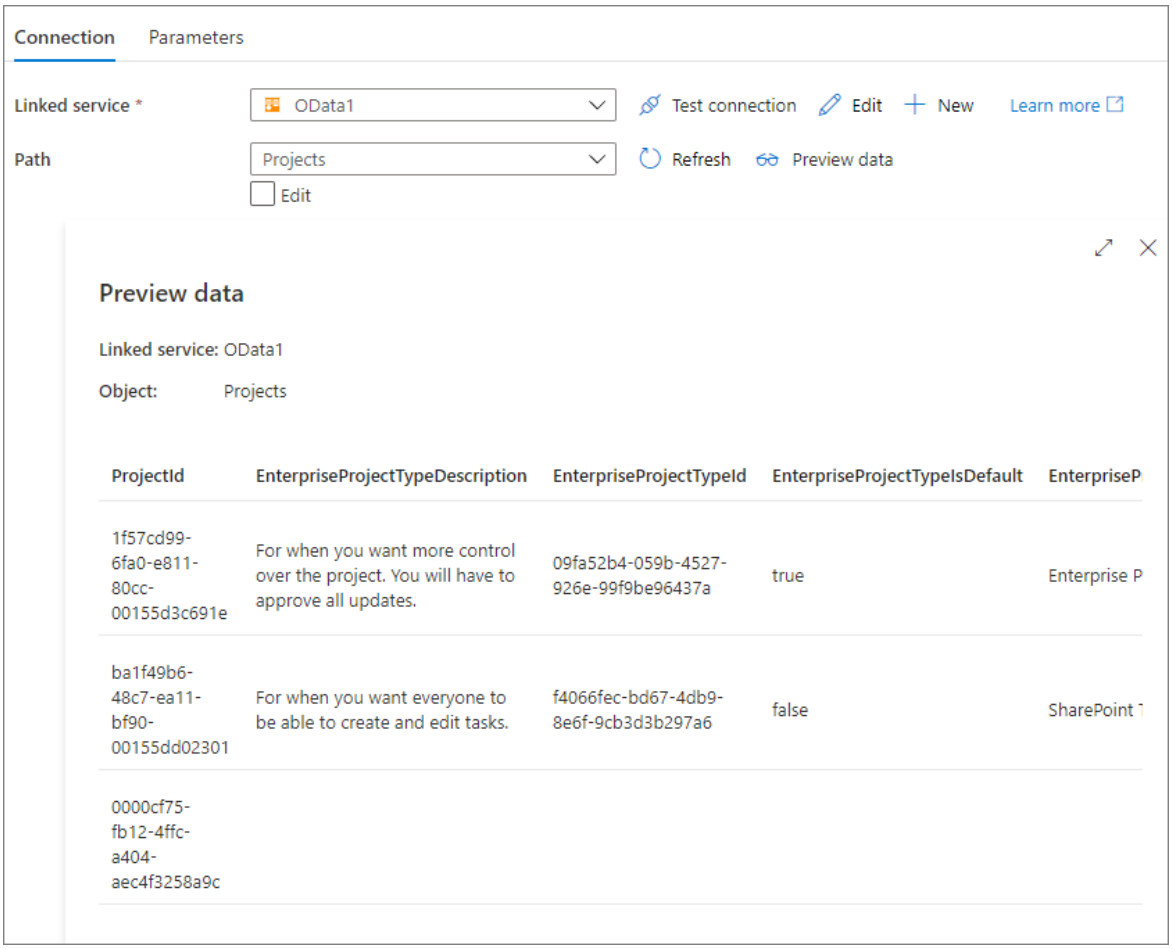
- **Service URL**: Enter `https://<your tenant name>.sharepoint.com/sites/pwa/_api/Projectdata`. Replace `<your tenant name>` with your own tenant name.
- **Authentication type**: Select **Anonymous**.
- **Auth headers**:
  - **Property name**: Choose **Authorization**.
  - **Value**: Enter `Bearer <access token from step 1>`.
- Test the linked service.



3. Create the OData dataset:

   a. Create the dataset with the OData linked service created in step 2.

   b. Preview data.



Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores that Copy Activity supports as sources and sinks, see Supported data stores and formats.

Copy data from and to ODBC data stores using Azure Data Factory or Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory to copy data from and to an ODBC data store. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This ODBC connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	②
Lookup activity	②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Specifically, this ODBC connector supports copying data from/to **any ODBC-compatible data stores** using **Basic** or **Anonymous** authentication. A **64-bit ODBC driver** is required. For ODBC sink, the service support ODBC version 2.0 standard.

Prerequisites

To use this ODBC connector, you need to:

- Set up a Self-hosted Integration Runtime. See Self-hosted Integration Runtime article for details.
- Install the 64-bit ODBC driver for the data store on the Integration Runtime machine.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

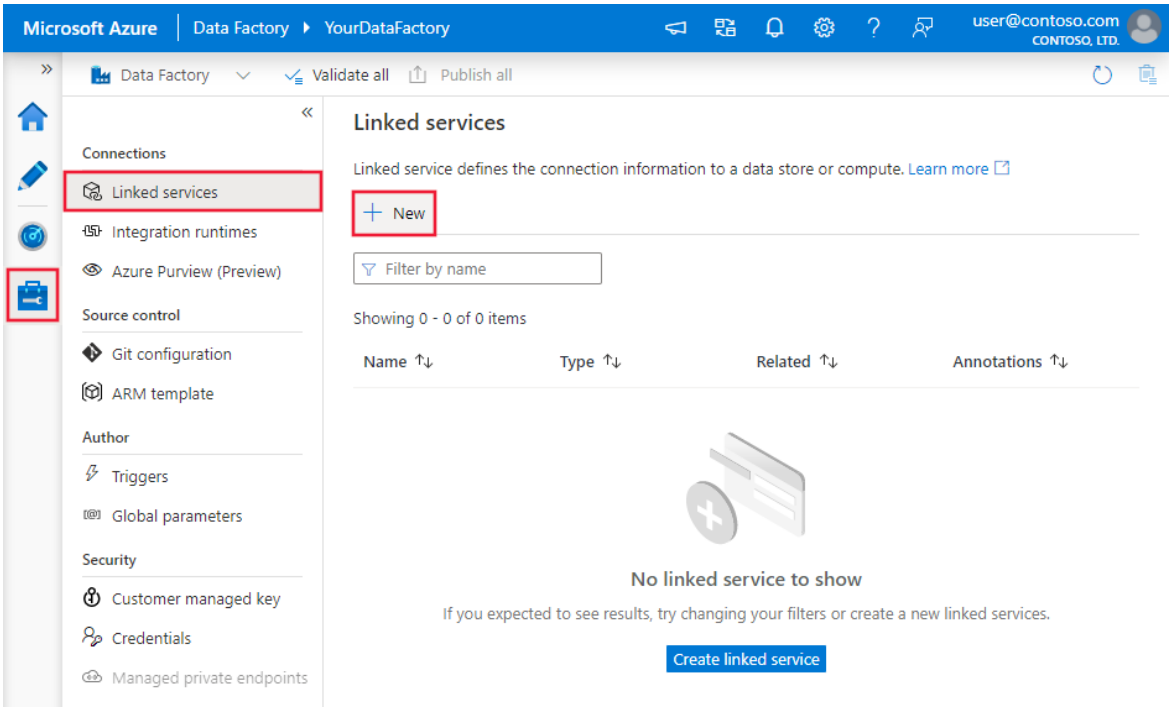
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to an ODBC data store using UI

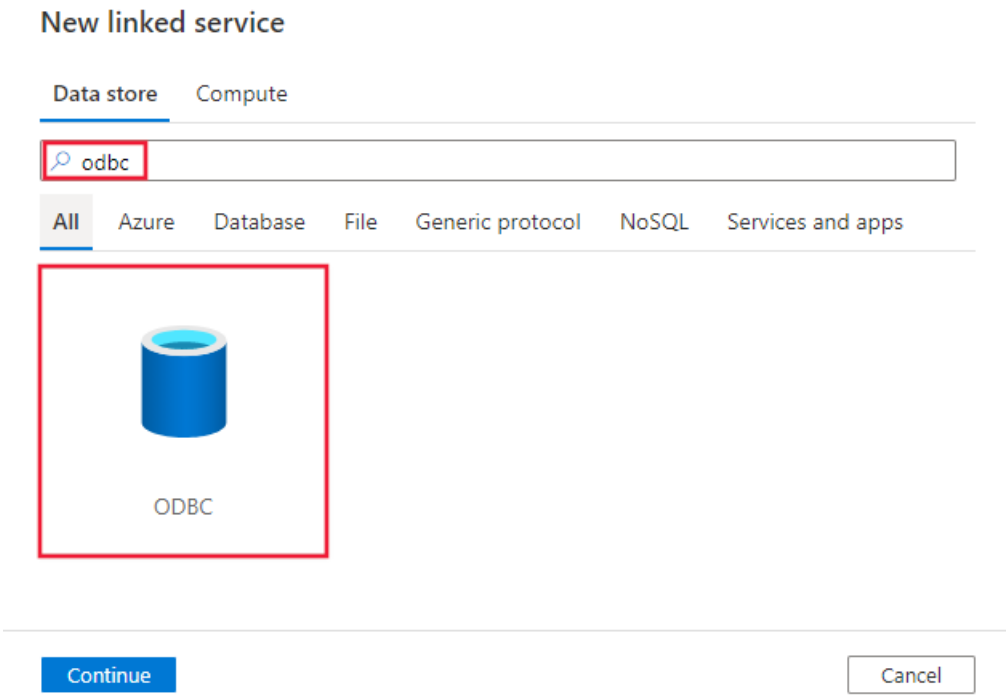
Use the following steps to create a linked service to an ODBC data store in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

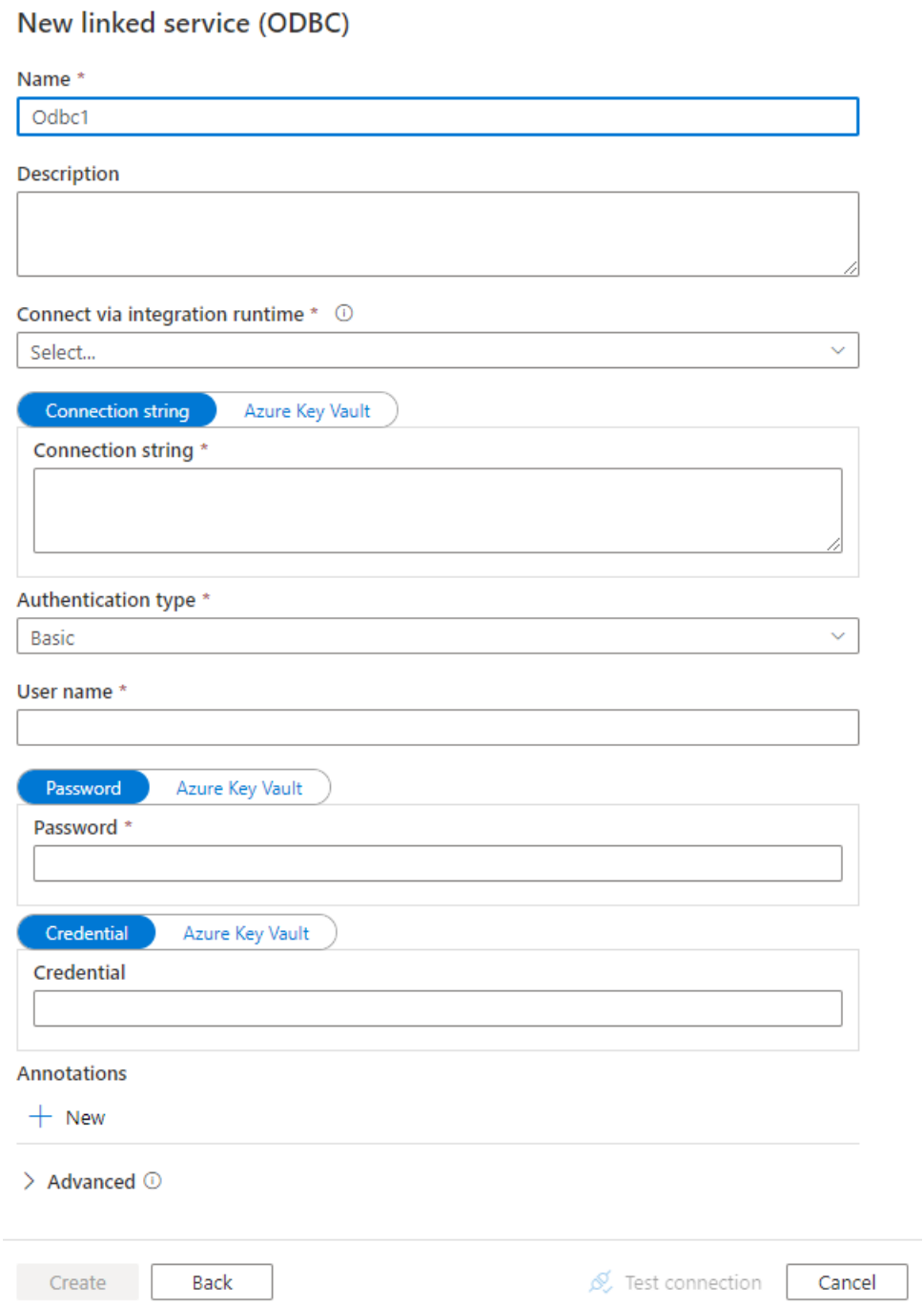
   - Azure Data Factory
   - Azure Synapse



2. Search for ODBC and select the ODBC connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to ODBC connector.

Linked service properties

The following properties are supported for ODBC linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Odbc**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	The connection string excluding the credential portion. You can specify the connection string with pattern like `Driver={SQL Server};Server=Server.database.windows.net; Database=TestDatabase;`, or use the system DSN (Data Source Name) you set up on the Integration Runtime machine with `DSN=<name of the DSN on IR machine>;` (you need still specify the credential portion in linked service accordingly). You can also put a password in Azure Key Vault and pull the `password` configuration out of the connection string. Refer to Store credentials in Azure Key Vault with more details.	Yes
authenticationType	Type of authentication used to connect to the ODBC data store. Allowed values are: **Basic** and **Anonymous**.	Yes
userName	Specify user name if you are using Basic authentication.	No
password	Specify password for the user account you specified for the userName. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
credential	The access credential portion of the connection string specified in driver-specific property-value format. Example: `"RefreshToken=<secret refresh token>;"`. Mark this field as a SecureString.	No
connectVia	The Integration Runtime to be used to connect to the data store. A Self-hosted Integration Runtime is required as mentioned in Prerequisites.	Yes

Example 1: using Basic authentication

```
{
    "name": "ODBCLinkedService",
    "properties": {
        "type": "Odbc",
        "typeProperties": {
            "connectionString": "<connection string>",
            "authenticationType": "Basic",
            "userName": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 2: using Anonymous authentication

```
{
    "name": "ODBCLinkedService",
    "properties": {
        "type": "Odbc",
        "typeProperties": {
            "connectionString": "<connection string>",
            "authenticationType": "Anonymous",
            "credential": {
                "type": "SecureString",
                "value": "RefreshToken=<secret refresh token>;"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by ODBC dataset.

To copy data from/to ODBC-compatible data store, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **OdbcTable**	Yes
tableName	Name of the table in the ODBC data store.	No for source (if "query" in activity source is specified); Yes for sink

Example

```
{
    "name": "ODBCDataset",
    "properties": {
        "type": "OdbcTable",
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<ODBC linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "tableName": "<table name>"
        }
    }
}
```

If you were using `RelationalTable` typed dataset, it is still supported as-is, while you are suggested to use the new one going forward.

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by ODBC source.

ODBC as source

To copy data from ODBC-compatible data store, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **OdbcSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromODBC",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<ODBC input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "OdbcSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

If you were using `RelationalSource` typed source, it is still supported as-is, while you are suggested to use the new one going forward.

ODBC as sink

To copy data to ODBC-compatible data store, set the sink type in the copy activity to **OdbcSink**. The following properties are supported in the copy activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to: **OdbcSink**	Yes
writeBatchTimeout	Wait time for the batch insert operation to complete before it times out. Allowed values are: timespan. Example: "00:30:00" (30 minutes).	No
writeBatchSize	Inserts data into the SQL table when the buffer size reaches writeBatchSize. Allowed values are: integer (number of rows).	No (default is 0 - auto detected)
preCopyScript	Specify a SQL query for Copy Activity to execute before writing data into data store in each run. You can use this property to clean up the pre-loaded data.	No

> **NOTE**
>
> For "writeBatchSize", if it's not set (auto-detected), copy activity first detects whether the driver supports batch operations, and set it to 10000 if it does, or set it to 1 if it doesn't. If you explicitly set the value other than 0, copy activity honors the value and fails at runtime if the driver doesn't support batch operations.

Example:

```
"activities":[
    {
        "name": "CopyToODBC",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<ODBC output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "OdbcSink",
                "writeBatchSize": 100000
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Troubleshoot connectivity issues

To troubleshoot connection issues, use the **Diagnostics** tab of **Integration Runtime Configuration Manager**.

1. Launch **Integration Runtime Configuration Manager**.
2. Switch to the **Diagnostics** tab.
3. Under the "Test Connection" section, select the **type** of data store (linked service).
4. Specify the **connection string** that is used to connect to the data store, choose the **authentication** and enter **user name**, **password**, and/or **credentials**.
5. Click **Test connection** to test the connection to the data store.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Microsoft 365 (Office 365) into Azure using Azure Data Factory or Synapse Analytics

8/19/2022 • 8 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Azure Data Factory and Synapse Analytics pipelines integrate with Microsoft Graph data connect, allowing you to bring the rich organizational data in your Microsoft 365 (Office 365) tenant into Azure in a scalable way and build analytics applications and extract insights based on these valuable data assets. Integration with Privileged Access Management provides secured access control for the valuable curated data in Microsoft 365 (Office 365). Please refer to this link for an overview on Microsoft Graph data connect and refer to this link for licensing information.

This article outlines how to use the Copy Activity to copy data from Microsoft 365 (Office 365). It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Microsoft 365 (Office 365) connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②

① Azure integration runtime ② Self-hosted integration runtime

ADF Microsoft 365 (Office 365) connector and Microsoft Graph data connect enables at scale ingestion of different types of datasets from Exchange Email enabled mailboxes, including address book contacts, calendar events, email messages, user information, mailbox settings, and so on. Refer here to see the complete list of datasets available.

For now, within a single copy activity you can only **copy data from Microsoft 365 (Office 365) into Azure Blob Storage, Azure Data Lake Storage Gen1, and Azure Data Lake Storage Gen2 in JSON format** (type setOfObjects). If you want to load Microsoft 365 (Office 365) into other types of data stores or in other formats, you can chain the first copy activity with a subsequent copy activity to further load data into any of the supported ADF destination stores (refer to "supported as a sink" column in the "Supported data stores and formats" table).

IMPORTANT

- The Azure subscription containing the data factory or Synapse workspace and the sink data store must be under the same Azure Active Directory (Azure AD) tenant as Microsoft 365 (Office 365) tenant.
- Ensure the Azure Integration Runtime region used for copy activity as well as the destination is in the same region where the Microsoft 365 (Office 365) tenant users' mailbox is located. Refer here to understand how the Azure IR location is determined. Refer to table here for the list of supported Office regions and corresponding Azure regions.
- Service Principal authentication is the only authentication mechanism supported for Azure Blob Storage, Azure Data Lake Storage Gen1, and Azure Data Lake Storage Gen2 as destination stores.

> **NOTE**
> Please use Azure integration runtime in both source and sink linked services. The self-hosted integration runtime and the managed virtual network integration runtime are not supported.

Prerequisites

To copy data from Microsoft 365 (Office 365) into Azure, you need to complete the following prerequisite steps:

- Your Microsoft 365 (Office 365) tenant admin must complete on-boarding actions as described here.
- Create and configure an Azure AD web application in Azure Active Directory. For instructions, see Create an Azure AD application.
- Make note of the following values, which you will use to define the linked service for Microsoft 365 (Office 365):
  - Tenant ID. For instructions, see Get tenant ID.
  - Application ID and Application key. For instructions, see Get application ID and authentication key.
- Add the user identity who will be making the data access request as the owner of the Azure AD web application (from the Azure AD web application > Settings > Owners > Add owner).
  - The user identity must be in the Microsoft 365 (Office 365) organization you are getting data from and must not be a Guest user.

Approving new data access requests

If this is the first time you are requesting data for this context (a combination of which data table is being access, which destination account is the data being loaded into, and which user identity is making the data access request), you will see the copy activity status as "In Progress", and only when you click into "Details" link under Actions will you see the status as "RequestingConsent". A member of the data access approver group needs to approve the request in the Privileged Access Management before the data extraction can proceed.

Refer here on how the approver can approve the data access request, and refer here for an explanation on the overall integration with Privileged Access Management, including how to set up the data access approver group.

Getting started

> **TIP**
> For a walkthrough of using Microsoft 365 (Office 365) connector, see Load data from Microsoft 365 (Office 365) article.

You can create a pipeline with the copy activity by using one of the following tools or SDKs. Select a link to go to a tutorial with step-by-step instructions to create a pipeline with a copy activity.

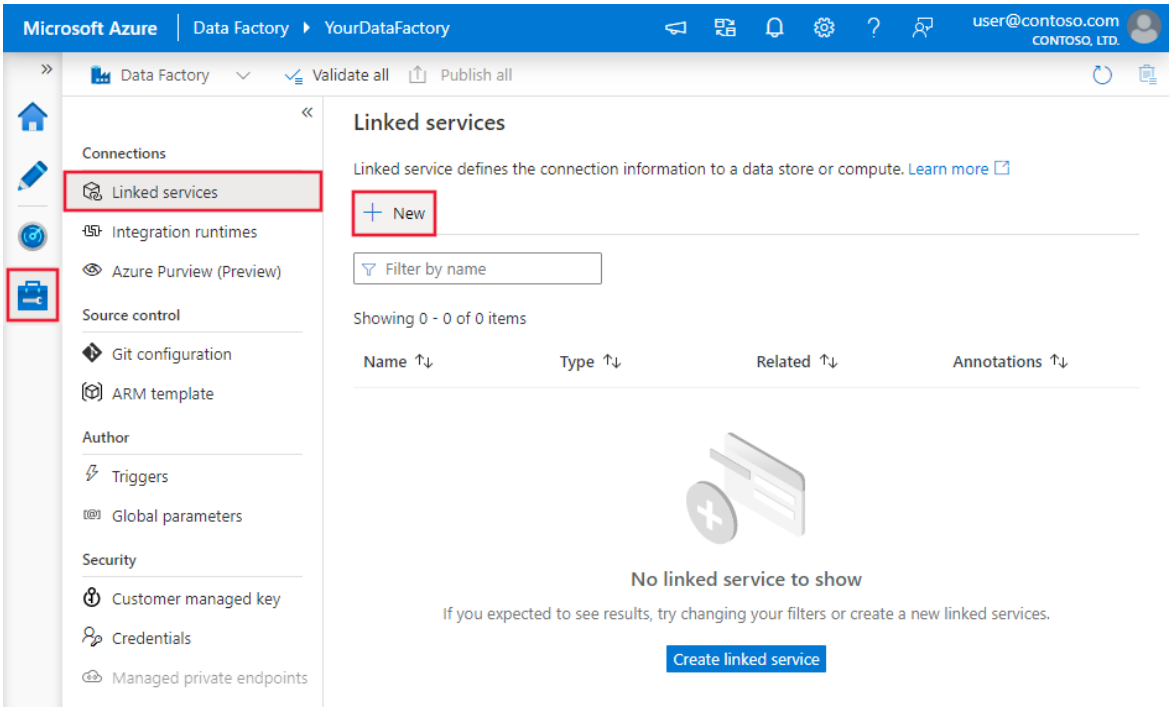
- Azure portal
- .NET SDK
- Python SDK
- Azure PowerShell
- REST API
- Azure Resource Manager template.

Create a linked service to Microsoft 365 (Office 365) using UI

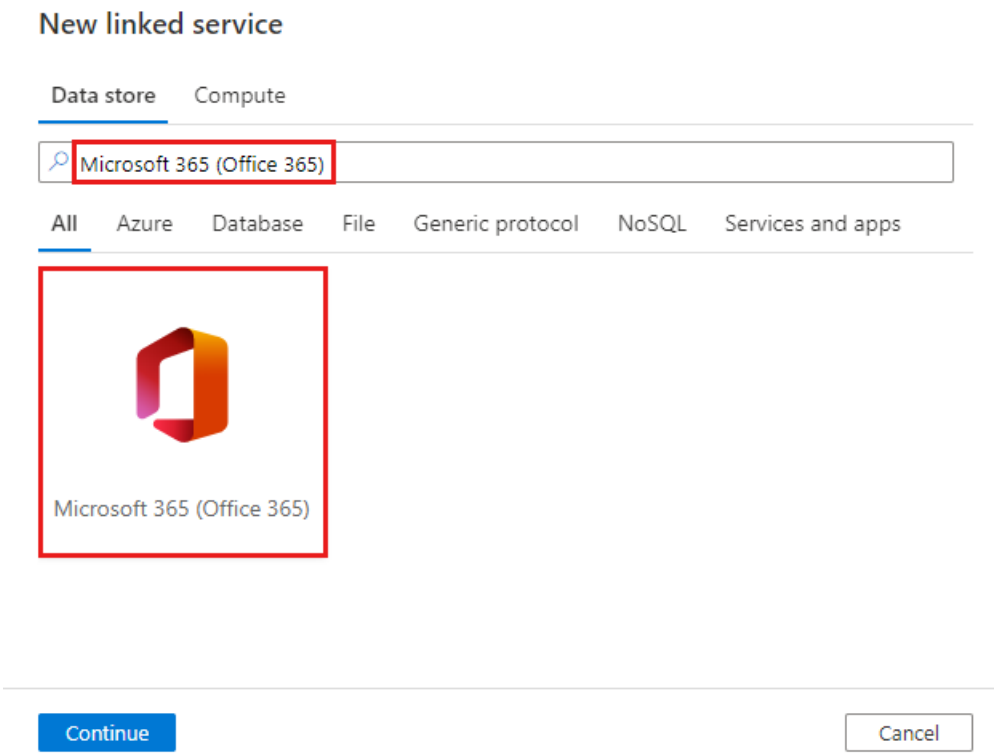
Use the following steps to create a linked service to Microsoft 365 (Office 365) in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

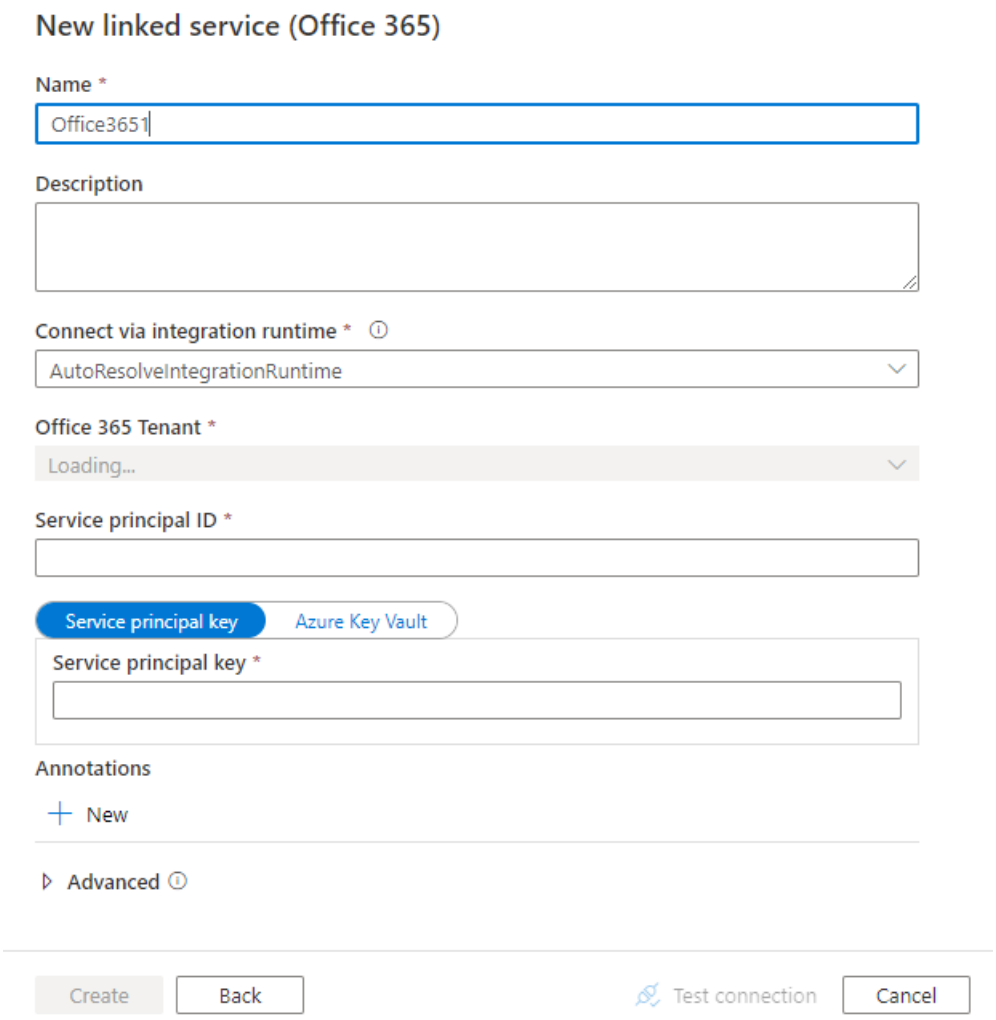
   - Azure Data Factory
   - Azure Synapse



2. Search for Microsoft 365 (Office 365) and select the Microsoft 365 (Office 365) connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Microsoft 365 (Office 365) connector.

Linked service properties

The following properties are supported for Microsoft 365 (Office 365) linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Office365**	Yes
office365TenantId	Azure tenant ID to which the Microsoft 365 (Office 365) account belongs.	Yes
servicePrincipalTenantId	Specify the tenant information under which your Azure AD web application resides.	Yes
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key. Mark this field as a SecureString to store it securely.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectVia	The Integration Runtime to be used to connect to the data store. If not specified, it uses the default Azure Integration Runtime.	No

NOTE

The difference between **office365TenantId** and **servicePrincipalTenantId** and the corresponding value to provide:

- If you are an enterprise developer developing an application against Microsoft 365 (Office 365) data for your own organization's usage, then you should supply the same tenant ID for both properties, which is your organization's AAD tenant ID.
- If you are an ISV developer developing an application for your customers, then office365TenantId will be your customer's (application installer) AAD tenant ID and servicePrincipalTenantId will be your company's AAD tenant ID.

Example:

```
{
    "name": "Office365LinkedService",
    "properties": {
        "type": "Office365",
        "typeProperties": {
            "office365TenantId": "<Microsoft 365 (Office 365) tenant id>",
            "servicePrincipalTenantId": "<AAD app service principal tenant id>",
            "servicePrincipalId": "<AAD app service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<AAD app service principal key>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Microsoft 365 (Office 365) dataset.

To copy data from Microsoft 365 (Office 365), the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **Office365Table**	Yes
tableName	Name of the dataset to extract from Microsoft 365 (Office 365). Refer here for the list of Microsoft 365 (Office 365) datasets available for extraction.	Yes

If you were setting `dateFilterColumn`, `startTime`, `endTime`, and `userScopeFilterUri` in dataset, it is still supported as-is, while you are suggested to use the new model in activity source going forward.

Example

```
{
    "name": "DS_May2019_O365_Message",
    "properties": {
        "type": "Office365Table",
        "linkedServiceName": {
            "referenceName": "<Microsoft 365 (Office 365) linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [],
        "typeProperties": {
            "tableName": "BasicDataSet_v0.Event_v1"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Microsoft 365 (Office 365) source.

Microsoft 365 (Office 365) as source

To copy data from Microsoft 365 (Office 365), the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **Office365Source**	Yes
allowedGroups	Group selection predicate. Use this property to select up to 10 user groups for whom the data will be retrieved. If no groups are specified, then data will be returned for the entire organization.	No
userScopeFilterUri	When `allowedGroups` property is not specified, you can use a predicate expression that is applied on the entire tenant to filter the specific rows to extract from Microsoft 365 (Office 365). The predicate format should match the query format of Microsoft Graph APIs, e.g. `https://graph.microsoft.com/v1.0/users?$filter=Department eq 'Finance'` .	No
dateFilterColumn	Name of the DateTime filter column. Use this property to limit the time range for which Microsoft 365 (Office 365) data is extracted.	Yes if dataset has one or more DateTime columns. Refer here for list of datasets that require this DateTime filter.
startTime	Start DateTime value to filter on.	Yes if `dateFilterColumn` is specified
endTime	End DateTime value to filter on.	Yes if `dateFilterColumn` is specified

PROPERTY	DESCRIPTION	REQUIRED
outputColumns	Array of the columns to copy to sink.	No

Example:

```
"activities": [
    {
        "name": "CopyFromO365ToBlob",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Microsoft 365 (Office 365) input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "Office365Source",
                "dateFilterColumn": "CreatedDateTime",
                "startTime": "2019-04-28T16:00:00.000Z",
                "endTime": "2019-05-05T16:00:00.000Z",
                "userScopeFilterUri": "https://graph.microsoft.com/v1.0/users?$filter=Department eq
'Finance'",
                "outputColumns": [
                    {
                        "name": "Id"
                    },
                    {
                        "name": "CreatedDateTime"
                    },
                    {
                        "name": "LastModifiedDateTime"
                    },
                    {
                        "name": "ChangeKey"
                    },
                    {
                        "name": "Categories"
                    },
                    {
                        "name": "OriginalStartTimeZone"
                    },
                    {
                        "name": "OriginalEndTimeZone"
                    },
                    {
                        "name": "ResponseStatus"
                    },
                    {
                        "name": "iCalUId"
                    },
                    {
                        "name": "ReminderMinutesBeforeStart"
                    },
                    {
                        "name": "IsReminderOn"
                    },
                    {
                        "name": "HasAttachments"
```

```
                    },
                    {
                        "name": "Subject"
                    },
                    {
                        "name": "Body"
                    },
                    {
                        "name": "Importance"
                    },
                    {
                        "name": "Sensitivity"
                    },
                    {
                        "name": "Start"
                    },
                    {
                        "name": "End"
                    },
                    {
                        "name": "Location"
                    },
                    {
                        "name": "IsAllDay"
                    },
                    {
                        "name": "IsCancelled"
                    },
                    {
                        "name": "IsOrganizer"
                    },
                    {
                        "name": "Recurrence"
                    },
                    {
                        "name": "ResponseRequested"
                    },
                    {
                        "name": "ShowAs"
                    },
                    {
                        "name": "Type"
                    },
                    {
                        "name": "Attendees"
                    },
                    {
                        "name": "Organizer"
                    },
                    {
                        "name": "WebLink"
                    },
                    {
                        "name": "Attachments"
                    },
                    {
                        "name": "BodyPreview"
                    },
                    {
                        "name": "Locations"
                    },
                    {
                        "name": "OnlineMeetingUrl"
                    },
                    {
                        "name": "OriginalStart"
                    },
                    {
                        "name": "SeriesMasterId"
```

```
                    }
                ]
            },
            "sink": {
                "type": "BlobSink"
            }
        }
    }
]
```

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from and to Oracle by using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 13 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the copy activity in Azure Data Factory to copy data from and to an Oracle database. It builds on the copy activity overview.

Supported capabilities

This Oracle connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Lookup activity	① ②
Script activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks by the copy activity, see the Supported data stores table.

Specifically, this Oracle connector supports:

- The following versions of an Oracle database:
  - Oracle 19c R1 (19.1) and higher
  - Oracle 18c R1 (18.1) and higher
  - Oracle 12c R1 (12.1) and higher
  - Oracle 11g R1 (11.1) and higher
  - Oracle 10g R1 (10.1) and higher
  - Oracle 9i R2 (9.2) and higher
  - Oracle 8i R3 (8.1.7) and higher
  - Oracle Database Cloud Exadata Service
- Parallel copying from an Oracle source. See the Parallel copy from Oracle section for details.

> **NOTE**
> Oracle proxy server isn't supported.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data

access strategies.

The integration runtime provides a built-in Oracle driver. Therefore, you don't need to manually install a driver when you copy data from and to Oracle.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

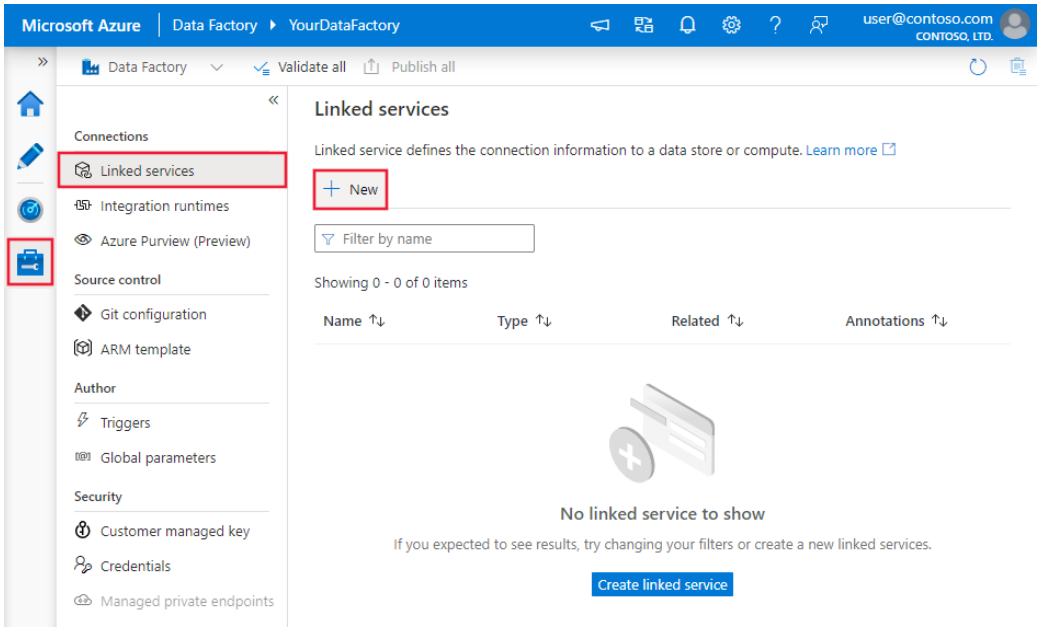
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Oracle using UI

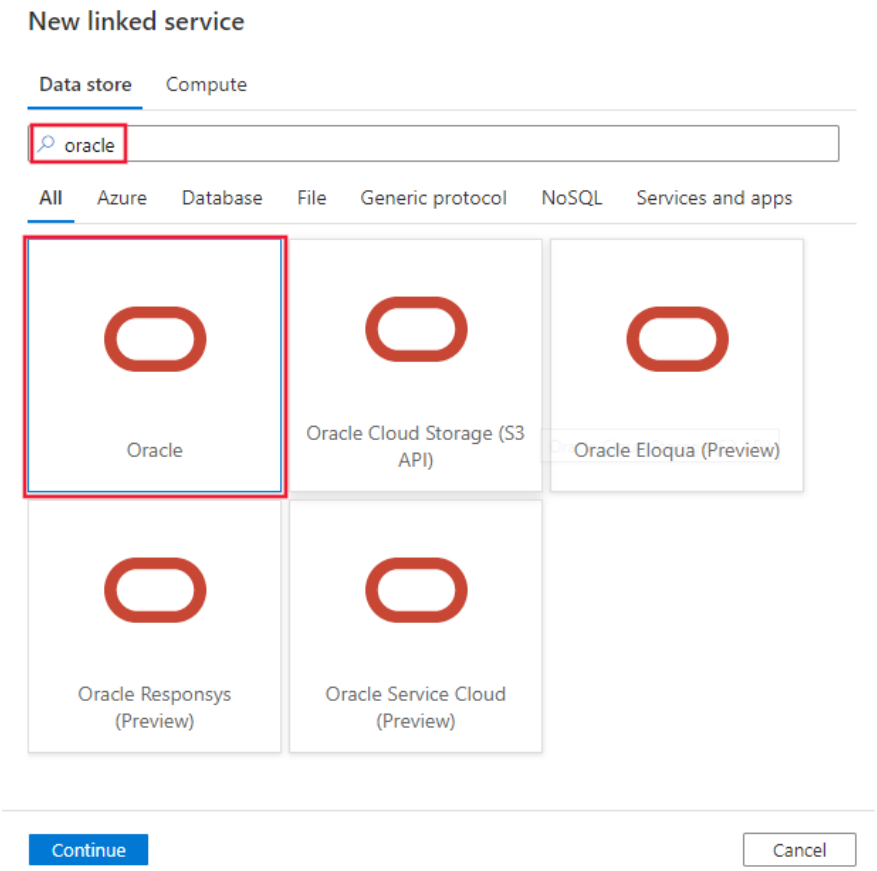
Use the following steps to create a linked service to Oracle in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

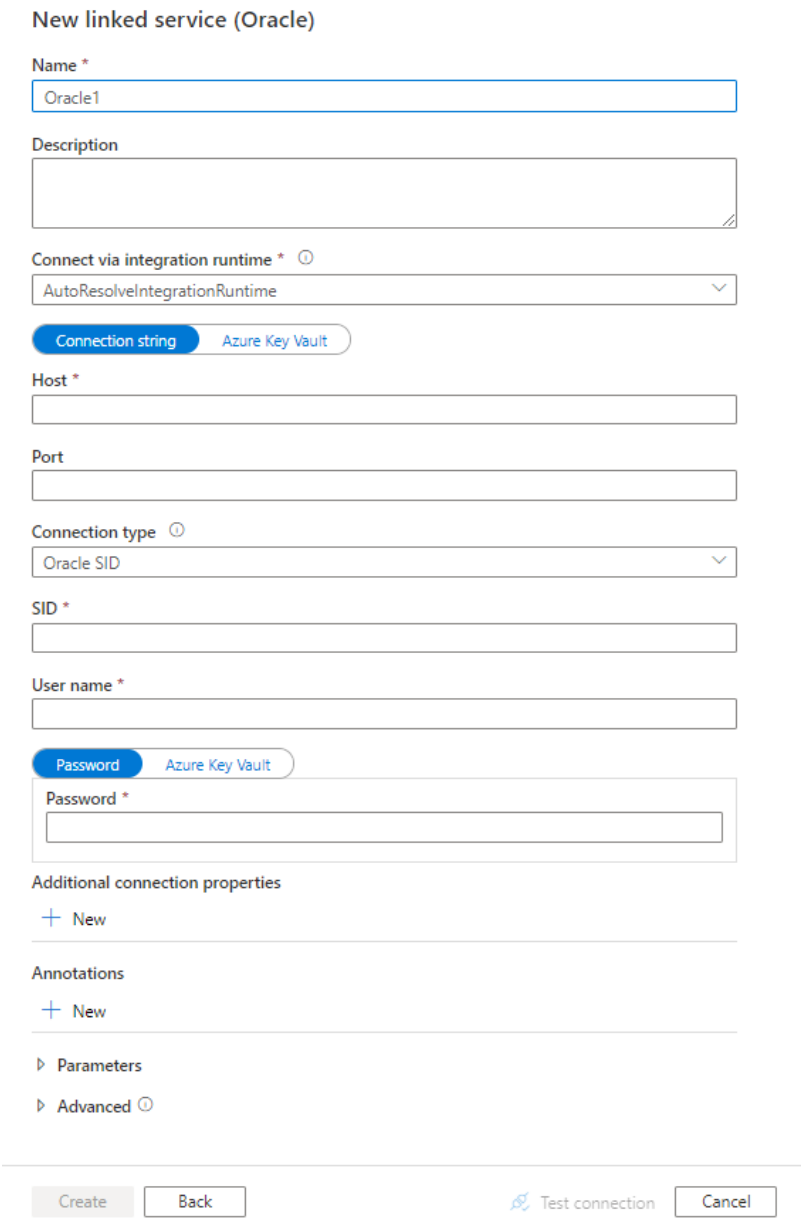
   - Azure Data Factory
   - Azure Synapse



2. Search for Oracle and select the Oracle connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to the Oracle connector.

Linked service properties

The Oracle linked service supports the following properties:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **Oracle**.	Yes

PROPERTY	DESCRIPTION	REQUIRED

connectionString	Specifies the information needed to connect to the Oracle Database instance.
	You can also put a password in Azure Key Vault, and pull the `password` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault with more details.
	
	Supported connection type: You can use **Oracle SID** or **Oracle Service Name** to identify your database:
	- If you use SID:
	`Host=<host>;Port=<port>;Sid=<sid>;User Id=<username>;Password=<password>;`
	- If you use Service Name:
	`Host=<host>;Port=<port>;ServiceName=<servicename>;User Id=<username>;Password=<password>;`
	For advanced Oracle native connection options, you can choose to add an entry in TNSNAMES.ORA file on the Oracle server, and in Oracle linked service, choose to use Oracle Service Name connection type and configure the corresponding service name.
		Yes

connectVia	The integration runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, the default Azure Integration Runtime is used.	No

TIP

If you get an error, "ORA-01025: UPI parameter out of range", and your Oracle version is 8i, add `WireProtocolMode=1` to your connection string. Then try again.

If you have multiple Oracle instances for failover scenario, you can create Oracle linked service and fill in the primary host, port, user name, password, etc., and add a new "**Additional connection properties**" with property name as `AlternateServers` and value as `(HostName=<secondary host>:PortNumber=<secondary port>:ServiceName=<secondary service name>)` - do not miss the brackets and pay attention to the colons (`:`) as separator. As an example, the following value of alternate servers defines two alternate database servers for connection failover:

```
(HostName=AccountingOracleServer:PortNumber=1521:SID=Accounting,HostName=255.201.11.24:PortNumber=1522:ServiceName=ABackup.NA.MyC
```

.

More connection properties you can set in connection string per your case:

PROPERTY	DESCRIPTION	ALLOWED VALUES
ArraySize	The number of bytes the connector can fetch in a single network round trip. E.g., `ArraySize=10485760`. Larger values increase throughput by reducing the number of times to fetch data across the network. Smaller values increase response time, as there is less of a delay waiting for the server to transmit data.	An integer from 1 to 4294967296 (4 GB). Default value is `60000`. The value 1 does not define the number of bytes, but indicates allocating space for exactly one row of data.

To enable encryption on Oracle connection, you have two options:

- To use **Triple-DES Encryption (3DES) and Advanced Encryption Standard (AES)**, on the Oracle server side, go to Oracle Advanced Security (OAS) and configure the encryption settings. For details, see this Oracle documentation. The Oracle Application Development Framework (ADF) connector automatically negotiates the encryption method to use the one you configure in OAS when establishing a connection to Oracle.
- To use **TLS**:

  1. Get the TLS/SSL certificate info. Get the Distinguished Encoding Rules (DER)-encoded certificate information of your TLS/SSL cert, and save the output (----- Begin Certificate ... End Certificate -----) as a text file.

     ```
     openssl x509 -inform DER -in [Full Path to the DER Certificate including the name of the DER Certificate] -text
     ```

     Example: Extract cert info from DERcert.cer, and then save the output to cert.txt.

     ```
     openssl x509 -inform DER -in DERcert.cer -text
     Output:
     -----BEGIN CERTIFICATE-----
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     XXXXXXXXX
     -----END CERTIFICATE-----
     ```

  2. Build the `keystore` or `truststore`. The following command creates the `truststore` file, with or without a password, in PKCS-12 format.

     ```
     openssl pkcs12 -in [Path to the file created in the previous step] -out [Path and name of TrustStore] -passout pass:[Keystore PWD] -nokeys -export
     ```

     Example: Create a PKCS12 `truststore` file, named MyTrustStoreFile, with a password.

     ```
     openssl pkcs12 -in cert.txt -out MyTrustStoreFile -passout pass:ThePWD -nokeys -export
     ```

  3. Place the `truststore` file on the self-hosted IR machine. For example, place the file at C:\MyTrustStoreFile.

  4. In the service, configure the Oracle connection string with `EncryptionMethod=1` and the corresponding `TrustStore` / `TrustStorePassword` value. For example,
     `Host=<host>;Port=<port>;Sid=<sid>;User Id=<username>;Password=<password>;EncryptionMethod=1;TrustStore=C:\\MyTrustStoreFile;TrustStorePassword=<trust_store_password>`
     .

Example:

```
{
    "name": "OracleLinkedService",
    "properties": {
        "type": "Oracle",
        "typeProperties": {
            "connectionString": "Host=<host>;Port=<port>;Sid=<sid>;User Id=<username>;Password=<password>;"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "OracleLinkedService",
    "properties": {
        "type": "Oracle",
        "typeProperties": {
            "connectionString": "Host=<host>;Port=<port>;Sid=<sid>;User Id=<username>;",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

This section provides a list of properties supported by the Oracle dataset. For a full list of sections and properties available for defining datasets, see Datasets.

To copy data from and to Oracle, set the type property of the dataset to `OracleTable`. The following properties are supported.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to `OracleTable`.	Yes
schema	Name of the schema.	No for source, Yes for sink
table	Name of the table/view.	No for source, Yes for sink
tableName	Name of the table/view with schema. This property is supported for backward compatibility. For new workload, use `schema` and `table`.	No for source, Yes for sink

Example:

```
{
    "name": "OracleDataset",
    "properties":
    {
        "type": "OracleTable",
        "schema": [],
        "typeProperties": {
            "schema": "<schema_name>",
            "table": "<table_name>"
        },
        "linkedServiceName": {
            "referenceName": "<Oracle linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

This section provides a list of properties supported by the Oracle source and sink. For a full list of sections and properties available for defining activities, see Pipelines.

Oracle as source

> **TIP**
> To load data from Oracle efficiently by using data partitioning, learn more from Parallel copy from Oracle.

To copy data from Oracle, set the source type in the copy activity to `OracleSource`. The following properties are supported in the copy activity **source** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to `OracleSource`.	Yes
oracleReaderQuery	Use the custom SQL query to read data. An example is `"SELECT * FROM MyTable"`. When you enable partitioned load, you need to hook any corresponding built-in partition parameters in your query. For examples, see the Parallel copy from Oracle section.	No
partitionOptions	Specifies the data partitioning options used to load data from Oracle. Allowed values are: **None** (default), **PhysicalPartitionsOfTable**, and **DynamicRange**. When a partition option is enabled (that is, not `None`), the degree of parallelism to concurrently load data from an Oracle database is controlled by the `parallelCopies` setting on the copy activity.	No
partitionSettings	Specify the group of the settings for data partitioning. Apply when the partition option isn't `None`.	No
partitionNames	The list of physical partitions that needs to be copied. Apply when the partition option is `PhysicalPartitionsOfTable`. If you use a query to retrieve the source data, hook `?AdfTabularPartitionName` in the WHERE clause. For an example, see the Parallel copy from Oracle section.	No
partitionColumnName	Specify the name of the source column **in integer type** that will be used by range partitioning for parallel copy. If not specified, the primary key of the table is auto-detected and used as the partition column. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfRangePartitionColumnName` in the WHERE clause. For an example, see the Parallel copy from Oracle section.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionUpperBound	The maximum value of the partition column to copy data out. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfRangePartitionUpbound` in the WHERE clause. For an example, see the Parallel copy from Oracle section.	No
partitionLowerBound	The minimum value of the partition column to copy data out. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfRangePartitionLowbound` in the WHERE clause. For an example, see the Parallel copy from Oracle section.	No

Example: copy data by using a basic query without partition

```
"activities":[
    {
        "name": "CopyFromOracle",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Oracle input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "OracleSource",
                "oracleReaderQuery": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Oracle as sink

To copy data to Oracle, set the sink type in the copy activity to `OracleSink`. The following properties are supported in the copy activity **sink** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to `OracleSink`.	Yes
writeBatchSize	Inserts data into the SQL table when the buffer size reaches `writeBatchSize`. Allowed values are Integer (number of rows).	No (default is 10,000)

PROPERTY	DESCRIPTION	REQUIRED
writeBatchTimeout	The wait time for the batch insert operation to complete before it times out. Allowed values are Timespan. An example is 00:30:00 (30 minutes).	No
preCopyScript	Specify a SQL query for the copy activity to run before writing data into Oracle in each run. You can use this property to clean up the preloaded data.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToOracle",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Oracle output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "OracleSink"
            }
        }
    }
]
```

Parallel copy from Oracle

The Oracle connector provides built-in data partitioning to copy data from Oracle in parallel. You can find data partitioning options on the **Source** tab of the copy activity.

When you enable partitioned copy, the service runs parallel queries against your Oracle source to load data by partitions. The parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your Oracle database.

You are suggested to enable parallel copy with data partitioning especially when you load large amount of data

from your Oracle database. The following are suggested configurations for different scenarios. When copying data into file-based data store, it's recommanded to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

SCENARIO	SUGGESTED SETTINGS
Full load from large table, with physical partitions.	**Partition option**: Physical partitions of table. During execution, the service automatically detects the physical partitions, and copies data by partitions.
Full load from large table, without physical partitions, while with an integer column for data partitioning.	**Partition options**: Dynamic range partition. **Partition column**: Specify the column used to partition data. If not specified, the primary key column is used.
Load a large amount of data by using a custom query, with physical partitions.	**Partition option**: Physical partitions of table. **Query**: `SELECT * FROM <TABLENAME> PARTITION("?AdfTabularPartitionName") WHERE <your_additional_where_clause>` . **Partition name**: Specify the partition name(s) to copy data from. If not specified, the service automatically detects the physical partitions on the table you specified in the Oracle dataset. During execution, the service replaces `?AdfTabularPartitionName` with the actual partition name, and sends to Oracle.
Load a large amount of data by using a custom query, without physical partitions, while with an integer column for data partitioning.	**Partition options**: Dynamic range partition. **Query**: `SELECT * FROM <TABLENAME> WHERE ?AdfRangePartitionColumnName <= ?AdfRangePartitionUpbound AND ?AdfRangePartitionColumnName >= ?AdfRangePartitionLowbound AND <your_additional_where_clause>` . **Partition column**: Specify the column used to partition data. You can partition against the column with integer data type. **Partition upper bound** and **partition lower bound**: Specify if you want to filter against partition column to retrieve data only between the lower and upper range. During execution, the service replaces `?AdfRangePartitionColumnName`, `?AdfRangePartitionUpbound`, and `?AdfRangePartitionLowbound` with the actual column name and value ranges for each partition, and sends to Oracle. For example, if your partition column "ID" is set with the lower bound as 1 and the upper bound as 80, with parallel copy set as 4, the service retrieves data by 4 partitions. Their IDs are between [1,20], [21, 40], [41, 60], and [61, 80], respectively.

TIP

When copying data from a non-partitioned table, you can use "Dynamic range" partition option to partition against an integer column. If your source data doesn't have such type of column, you can leverage ORA_HASH function in source query to generate a column and use it as partition column.

Example: query with physical partition

```
"source": {
    "type": "OracleSource",
    "query":"SELECT * FROM <TABLENAME> PARTITION(\"?AdfTabularPartitionName\") WHERE
<your_additional_where_clause>",
    "partitionOption": "PhysicalPartitionsOfTable",
    "partitionSettings": {
        "partitionNames": [
            "<partitionA_name>",
            "<partitionB_name>"
        ]
    }
}
```

Example: query with dynamic range partition

```
"source": {
    "type": "OracleSource",
    "query":"SELECT * FROM <TABLENAME> WHERE ?AdfRangePartitionColumnName <= ?AdfRangePartitionUpbound AND ?
AdfRangePartitionColumnName >= ?AdfRangePartitionLowbound AND <your_additional_where_clause>",
    "partitionOption": "DynamicRange",
    "partitionSettings": {
        "partitionColumnName": "<partition_column_name>",
        "partitionUpperBound": "<upper_value_of_partition_column>",
        "partitionLowerBound": "<lower_value_of_partition_column>"
    }
}
```

Data type mapping for Oracle

When you copy data from and to Oracle, the following interim data type mappings are used within the service. To learn about how the copy activity maps the source schema and data type to the sink, see Schema and data type mappings.

ORACLE DATA TYPE	INTERIM DATA TYPE
BFILE	Byte[]
BLOB	Byte[] (only supported on Oracle 10g and higher)
CHAR	String
CLOB	String
DATE	DateTime
FLOAT	Decimal, String (if precision > 28)
INTEGER	Decimal, String (if precision > 28)
LONG	String
LONG RAW	Byte[]
NCHAR	String
NCLOB	String
NUMBER (p,s)	Decimal, String (if p > 28)
NUMBER without precision and scale	Double
NVARCHAR2	String

ORACLE DATA TYPE	INTERIM DATA TYPE
RAW	Byte[]
ROWID	String
TIMESTAMP	DateTime
TIMESTAMP WITH LOCAL TIME ZONE	String
TIMESTAMP WITH TIME ZONE	String
UNSIGNED INTEGER	Number
VARCHAR2	String
XML	String

> **NOTE**
> The data types INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND aren't supported.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from Oracle Cloud Storage using Azure Data Factory or Synapse Analytics

8/19/2022 • 9 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to copy data from Oracle Cloud Storage. To learn more, read the introductory articles for Azure Data Factory and Synapse Analytics.

Supported capabilities

This Oracle Cloud Storage connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②
GetMetadata activity	① ②
Delete activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

Specifically, this Oracle Cloud Storage connector supports copying files as is or parsing files with the supported file formats and compression codecs. It takes advantage of Oracle Cloud Storage's S3-compatible interoperability.

Prerequisites

To copy data from Oracle Cloud Storage, please refer here for the prerequisites and required permission.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

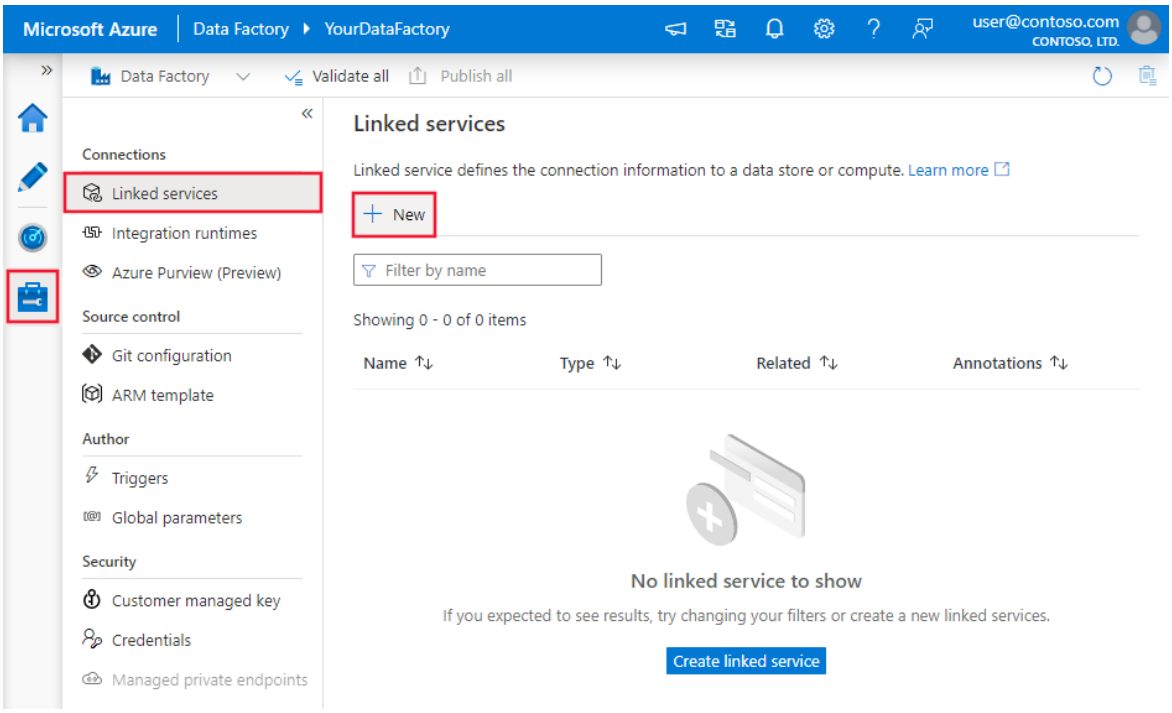
Create a linked service to Oracle Cloud Storage using UI

Use the following steps to create a linked service to Oracle Cloud Storage in the Azure portal UI.

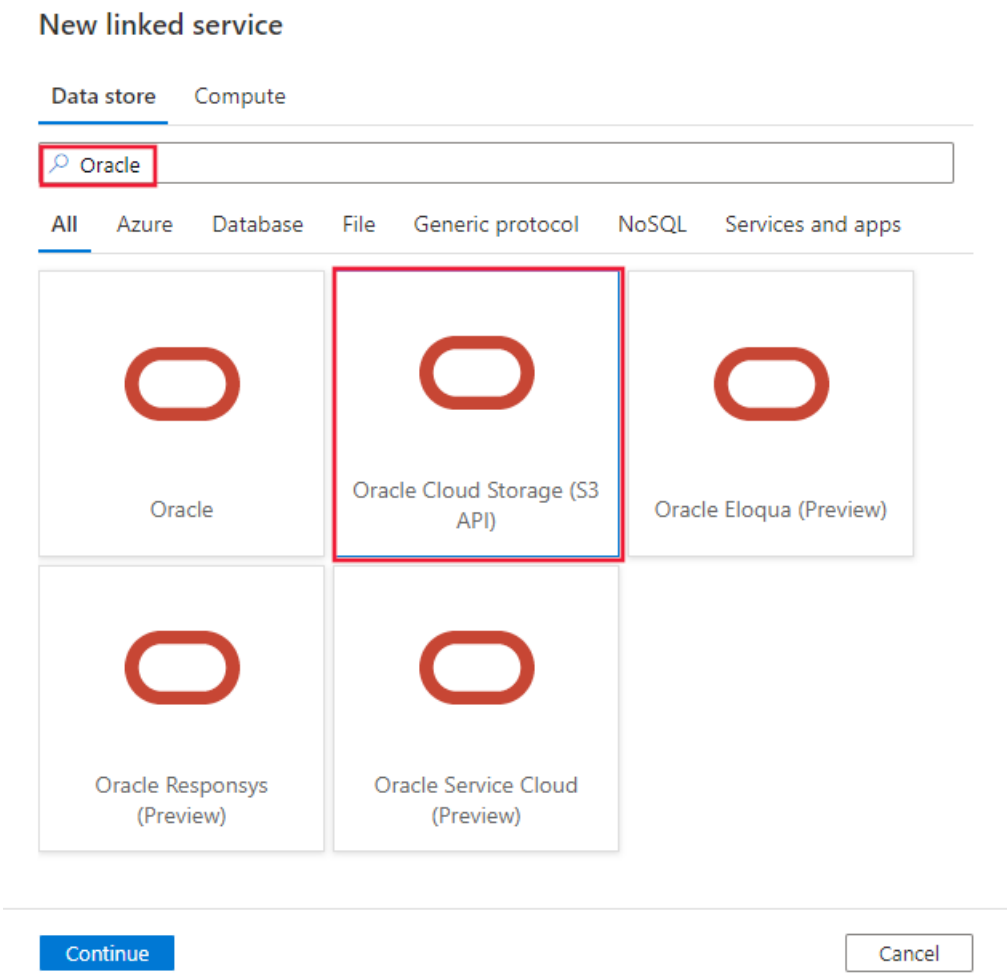
1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services,

then click New:

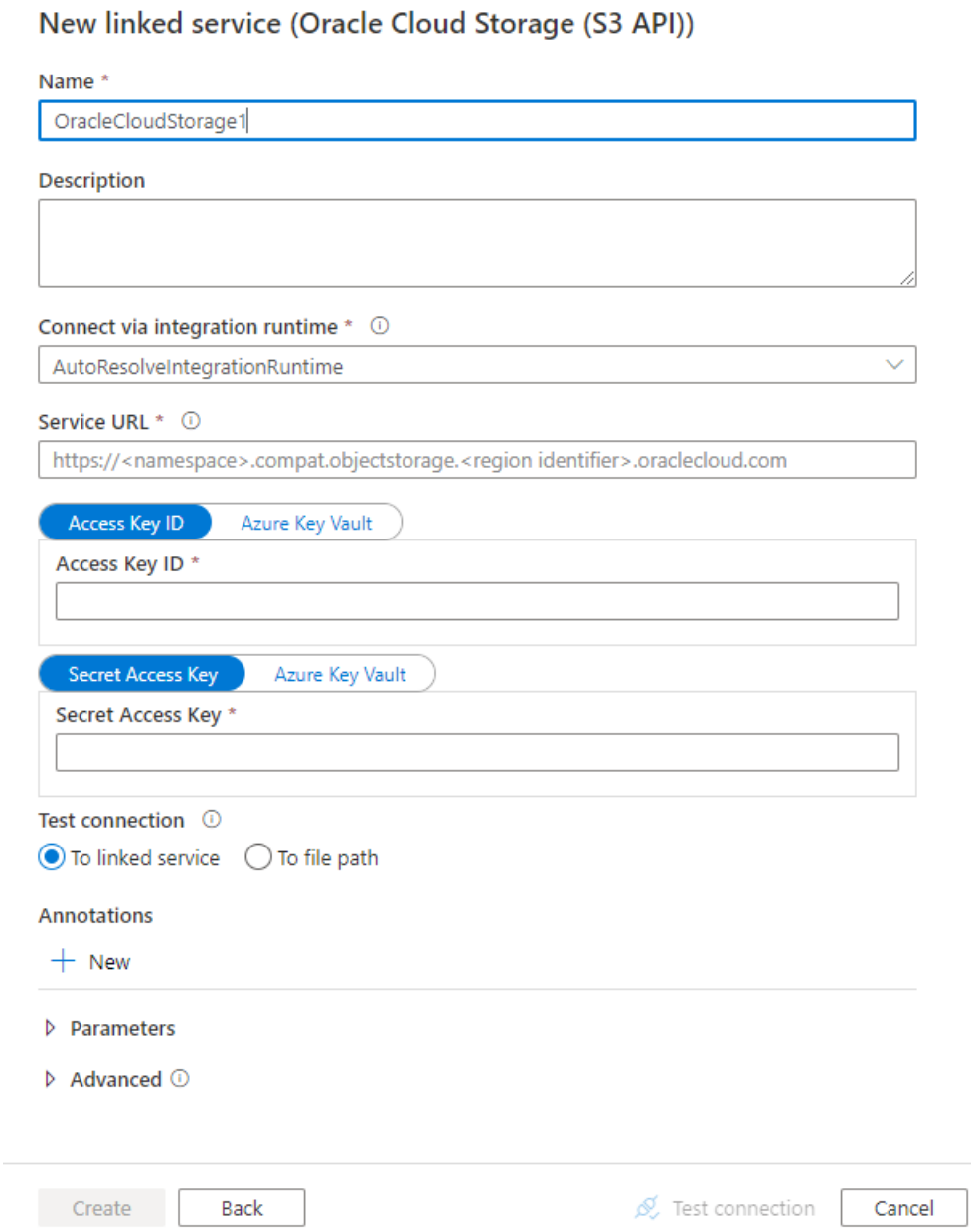
- Azure Data Factory
- Azure Synapse



2. Search for Oracle and select the Oracle Cloud Storage connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to Oracle Cloud Storage.

Linked service properties

The following properties are supported for Oracle Cloud Storage linked services:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **OracleCloudStorage**.	Yes
accessKeyId	ID of the secret access key. To find the access key and secret, see Prerequisites.	Yes

PROPERTY	DESCRIPTION	REQUIRED
secretAccessKey	The secret access key itself. Mark this field as **SecureString** to store it securely, or reference a secret stored in Azure Key Vault.	Yes
serviceUrl	Specify the custom endpoint as `https://<namespace>.compat.objectstorage.<region identifier>.oraclecloud.com`. Refer here for more details	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

Here's an example:

```
{
    "name": "OracleCloudStorageLinkedService",
    "properties": {
        "type": "OracleCloudStorage",
        "typeProperties": {
            "accessKeyId": "<access key id>",
            "secretAccessKey": {
                "type": "SecureString",
                "value": "<secret access key>"
            },
            "serviceUrl": "https://<namespace>.compat.objectstorage.<region identifier>.oraclecloud.com"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for Oracle Cloud Storage under `location` settings in a format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property under `location` in the dataset must be set to **OracleCloudStorageLocation**.	Yes
bucketName	The Oracle Cloud Storage bucket name.	Yes
folderPath	The path to folder under the given bucket. If you want to use a wildcard to filter the folder, skip this setting and specify that in activity source settings.	No
fileName	The file name under the given bucket and folder path. If you want to use a wildcard to filter the files, skip this setting and specify that in activity source settings.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<Oracle Cloud Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "OracleCloudStorageLocation",
                "bucketName": "bucketname",
                "folderPath": "folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties that the Oracle Cloud Storage source supports.

Oracle Cloud Storage as a source type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format

- ORC format
- Parquet format
- XML format

The following properties are supported for Oracle Cloud Storage under `storeSettings` settings in a format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property under `storeSettings` must be set to **OracleCloudStorageReadSettings**.	Yes
Locate the files to copy:		
OPTION 1: static path	Copy from the given bucket or folder/file path specified in the dataset. If you want to copy all files from a bucket or folder, additionally specify `wildcardFileName` as `*`.	
OPTION 2: Oracle Cloud Storage prefix - prefix	Prefix for the Oracle Cloud Storage key name under the given bucket configured in the dataset to filter source Oracle Cloud Storage files. Oracle Cloud Storage keys whose names start with `bucket_in_dataset/this_prefix` are selected. It utilizes Oracle Cloud Storage's service-side filter, which provides better performance than a wildcard filter.	No
OPTION 3: wildcard - wildcardFolderPath	The folder path with wildcard characters under the given bucket configured in a dataset to filter source folders. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your folder name has a wildcard or this escape character inside. See more examples in Folder and file filter examples.	No
OPTION 3: wildcard - wildcardFileName	The file name with wildcard characters under the given bucket and folder path (or wildcard folder path) to filter source files. Allowed wildcards are: `*` (matches zero or more characters) and `?` (matches zero or single character). Use `^` to escape if your file name has a wildcard or this escape character inside. See more examples in Folder and file filter examples.	Yes

PROPERTY	DESCRIPTION	REQUIRED
OPTION 3: a list of files - fileListPath	Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset. When you're using this option, do not specify the file name in the dataset. See more examples in File list examples.	No
Additional settings:		
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when **recursive** is set to **true** and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are **true** (default) and **false**. This property doesn't apply when you configure `fileListPath`.	No
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	Files are filtered based on the attribute: last modified. The files will be selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z". The properties can be **NULL**, which means no file attribute filter will be applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is **NULL**, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is **NULL**, the files whose last modified attribute is less than the datetime value will be selected. This property doesn't apply when you configure `fileListPath`.	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No

PROPERTY	DESCRIPTION	REQUIRED
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromOracleCloudStorage",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "OracleCloudStorageReadSettings",
                    "recursive": true,
                    "wildcardFolderPath": "myfolder*A",
                    "wildcardFileName": "*.csv"
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

BUCKET	KEY	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)

BUCKET	KEY	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
bucket	`Folder*/*`	false	bucket FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
bucket	`Folder*/*`	true	bucket FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv
bucket	`Folder*/*.csv`	false	bucket FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
bucket	`Folder*/*.csv`	true	bucket FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This section describes the resulting behavior of using a file list path in the Copy activity source.

Assume that you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	CONFIGURATION
bucket FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In dataset:** - Bucket: `bucket` - Folder path: `FolderA` **In copy activity source:** - File list path: `bucket/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line, with the relative path to the path configured in the dataset.

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity.

Delete activity properties

To learn details about the properties, check Delete activity.

Next steps

For a list of data stores that the Copy activity supports as sources and sinks, see Supported data stores.

Copy data from Oracle Eloqua using Azure Data Factory or Synapse Analytics (Preview)

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Oracle Eloqua. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and provide feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Oracle Eloqua connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

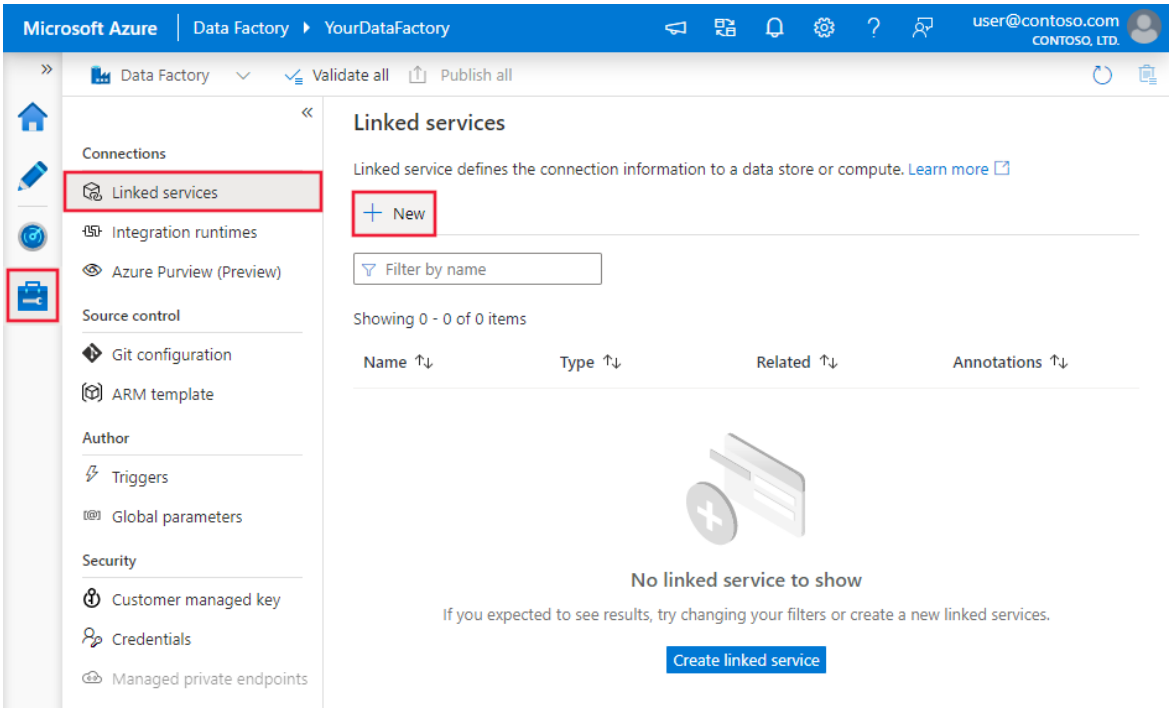
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Oracle Eloqua using UI

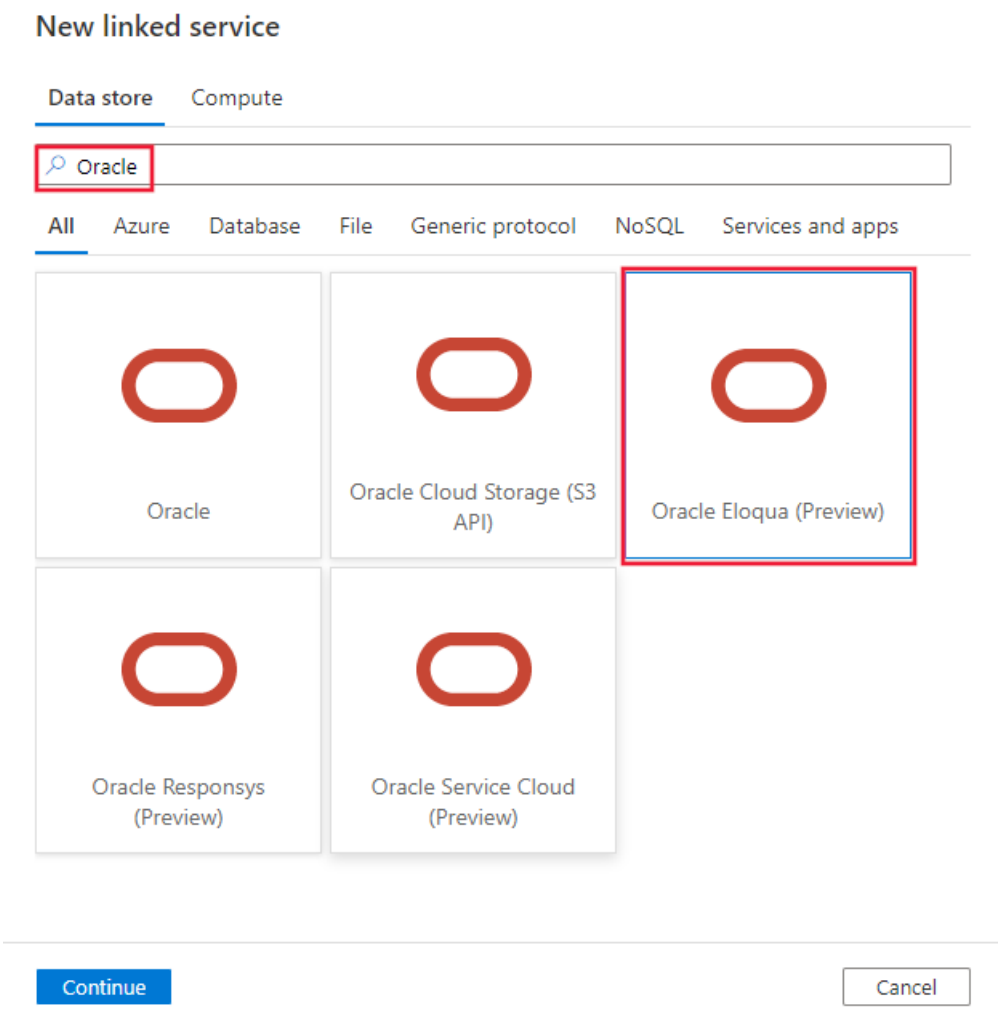
Use the following steps to create a linked service to Oracle Eloqua in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

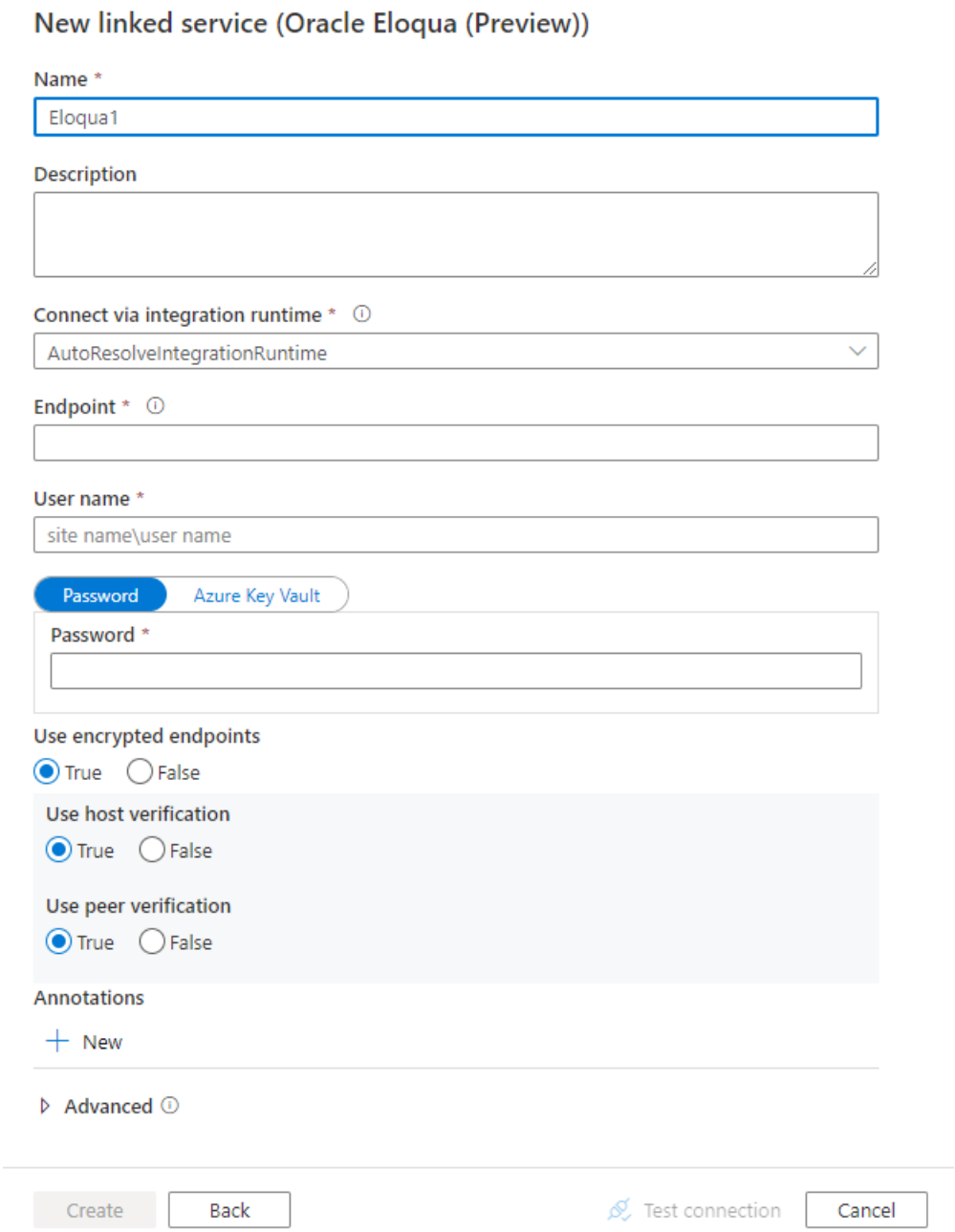
- Azure Data Factory
- Azure Synapse



2. Search for Oracle and select the Oracle Eloqua connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Oracle Eloqua connector.

Linked service properties

The following properties are supported for Oracle Eloqua linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Eloqua**	Yes

PROPERTY	DESCRIPTION	REQUIRED
endpoint	The endpoint of the Eloqua server. Eloqua supports multiple data centers, to determine your endpoint, login to https://login.eloqua.com with your credential, then copy the **base URL** portion from the redirected URL with the pattern of `xxx.xxx.eloqua.com`.	Yes
username	The site name and user name of your Eloqua account in the form: `SiteName\Username` e.g. `Eloqua\Alice`.	Yes
password	The password corresponding to the user name. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
    "name": "EloquaLinkedService",
    "properties": {
        "type": "Eloqua",
        "typeProperties": {
            "endpoint" : "<base URL e.g. xxx.xxx.eloqua.com>",
            "username" : "<site name>\\<user name e.g. Eloqua\\Alice>",
            "password": {
                 "type": "SecureString",
                 "value": "<password>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Oracle Eloqua dataset.

To copy data from Oracle Eloqua, set the type property of the dataset to **EloquaObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **EloquaObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "EloquaDataset",
    "properties": {
        "type": "EloquaObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Eloqua linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Oracle Eloqua source.

Eloqua as source

To copy data from Oracle Eloqua, set the source type in the copy activity to **EloquaSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **EloquaSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM Accounts"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromEloqua",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Eloqua input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "EloquaSource",
                "query": "SELECT * FROM Accounts"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of supported data stores in the service, see supported data stores.

Copy data from Oracle Responsys using Azure Data Factory or Synapse Analytics (Preview)

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Oracle Responsys. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Oracle Responsys connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

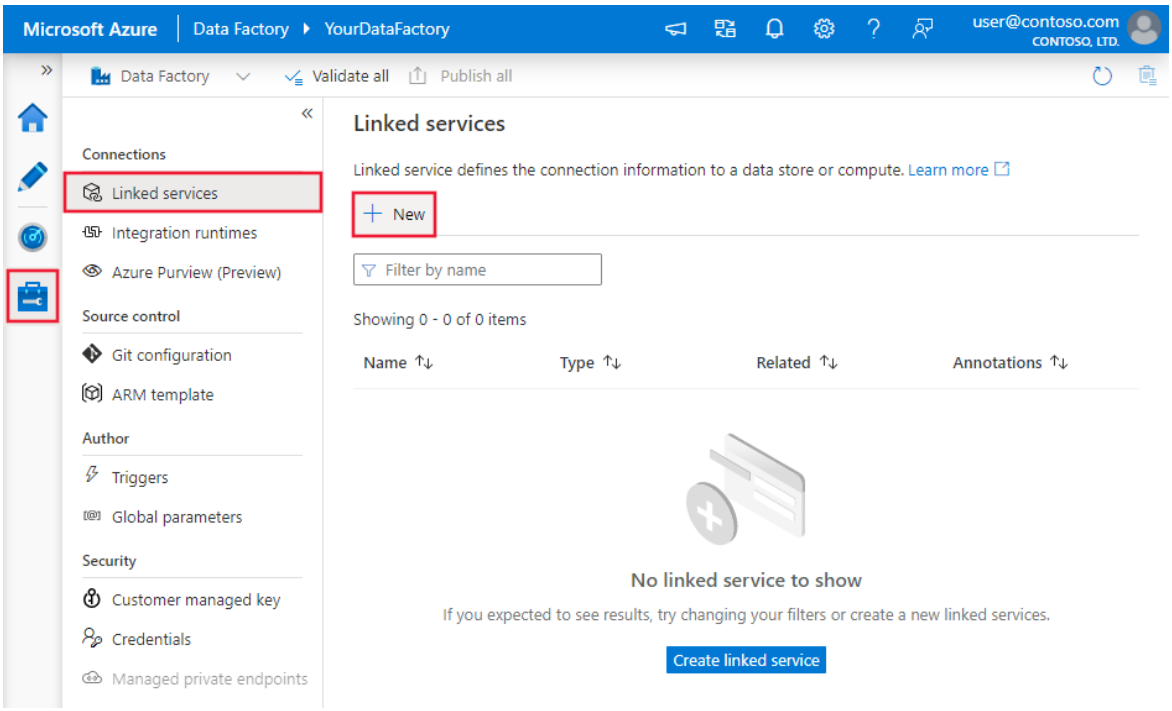
You can create a pipeline with copy activity using .NET SDK, Python SDK, Azure PowerShell, REST API, or Azure Resource Manager template. See Copy activity tutorial for step-by-step instructions to create a pipeline with a copy activity.

Create a linked service to Oracle Responsys using UI

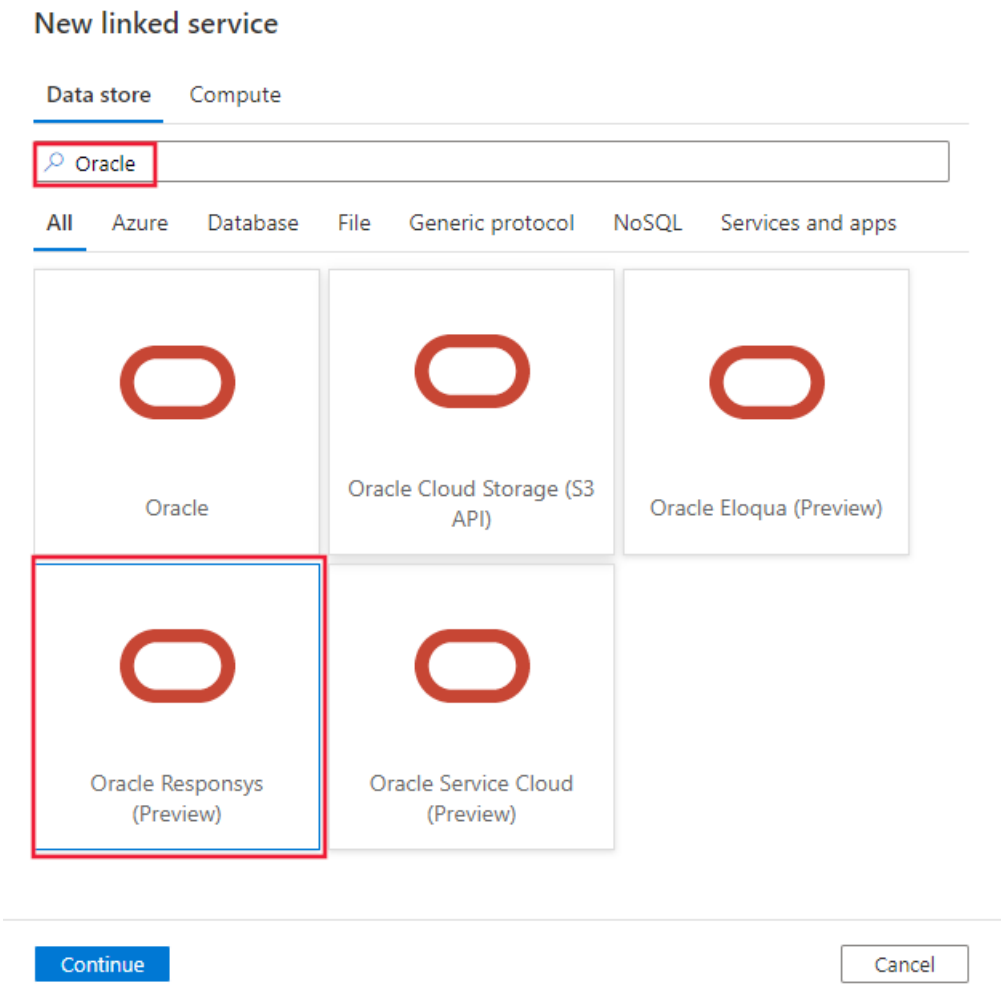
Use the following steps to create a linked service to Oracle Responsys in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

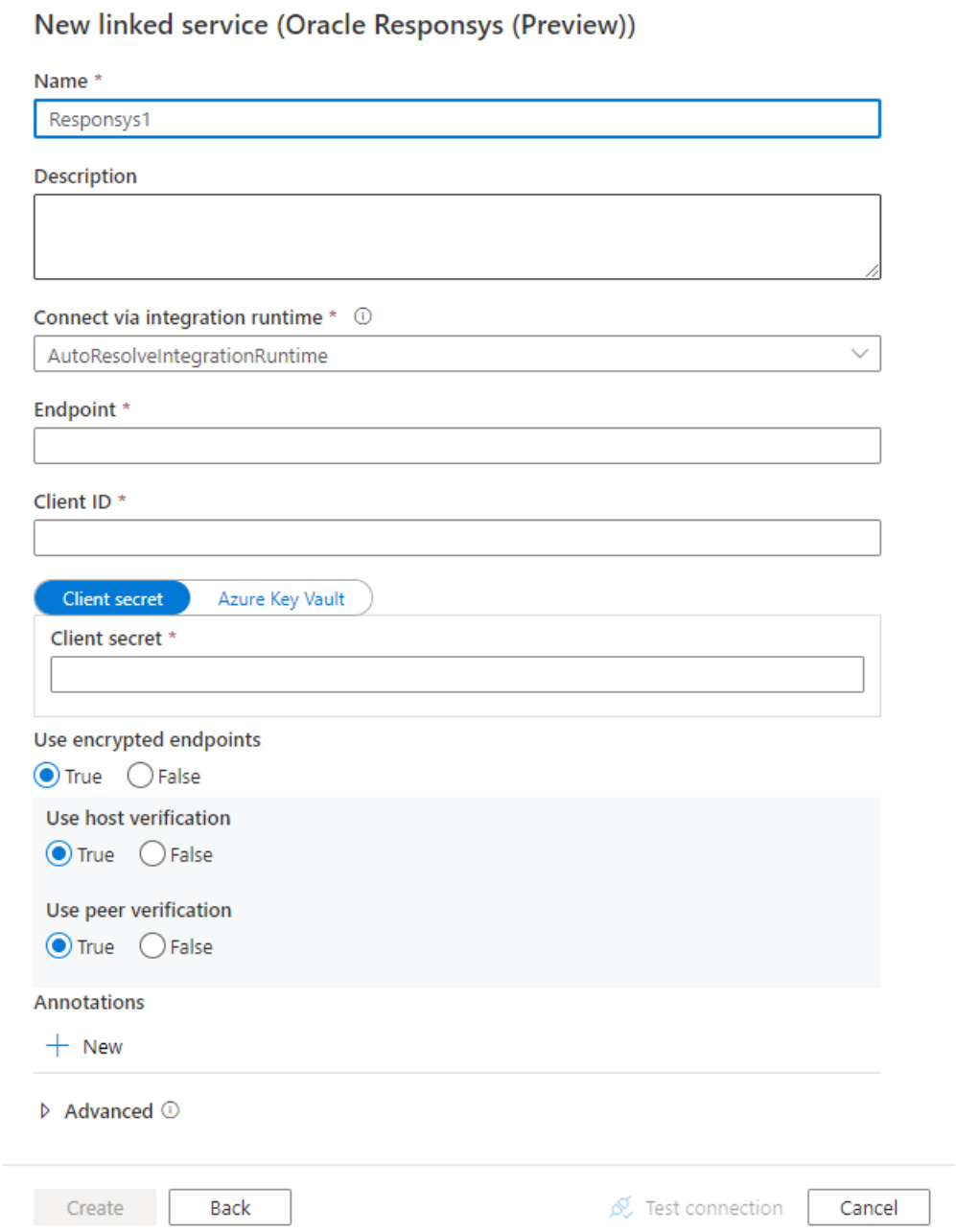
   - Azure Data Factory
   - Azure Synapse



2. Search for Oracle and select the Oracle Responsys connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Oracle Responsys connector.

Linked service properties

The following properties are supported for Oracle Responsys linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Responsys**	Yes
endpoint	The endpoint of the Respopnsys server	Yes
clientId	The client ID associated with the Responsys application.	Yes

PROPERTY	DESCRIPTION	REQUIRED
clientSecret	The client secret associated with the Responsys application. You can choose to mark this field as a SecureString to store it securely in the service, or store password in Azure Key Vault and let the service copy activity pull from there when performing data copy - learn more from Store credentials in Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
    "name": "OracleResponsysLinkedService",
    "properties": {
        "type": "Responsys",
        "typeProperties": {
            "endpoint" : "<endpoint>",
            "clientId" : "<clientId>",
            "clientSecret": {
                 "type": "SecureString",
                 "value": "<clientSecret>"
            },
            "useEncryptedEndpoints" : true,
            "useHostVerification" : true,
            "usePeerVerification" : true
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Oracle Responsys dataset.

To copy data from Oracle Responsys, set the type property of the dataset to **ResponsysObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **ResponsysObject**	Yes

PROPERTY	DESCRIPTION	REQUIRED
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "OracleResponsysDataset",
    "properties": {
        "type": "ResponsysObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Oracle Responsys linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Oracle Responsys source.

Oracle Responsys as source

To copy data from Oracle Responsys, set the source type in the copy activity to **ResponsysSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **ResponsysSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromOracleResponsys",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Oracle Responsys input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "ResponsysSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Oracle Service Cloud using Azure Data Factory or Synapse Analytics (Preview)

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Oracle Service Cloud. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and provide feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Oracle Service Cloud connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

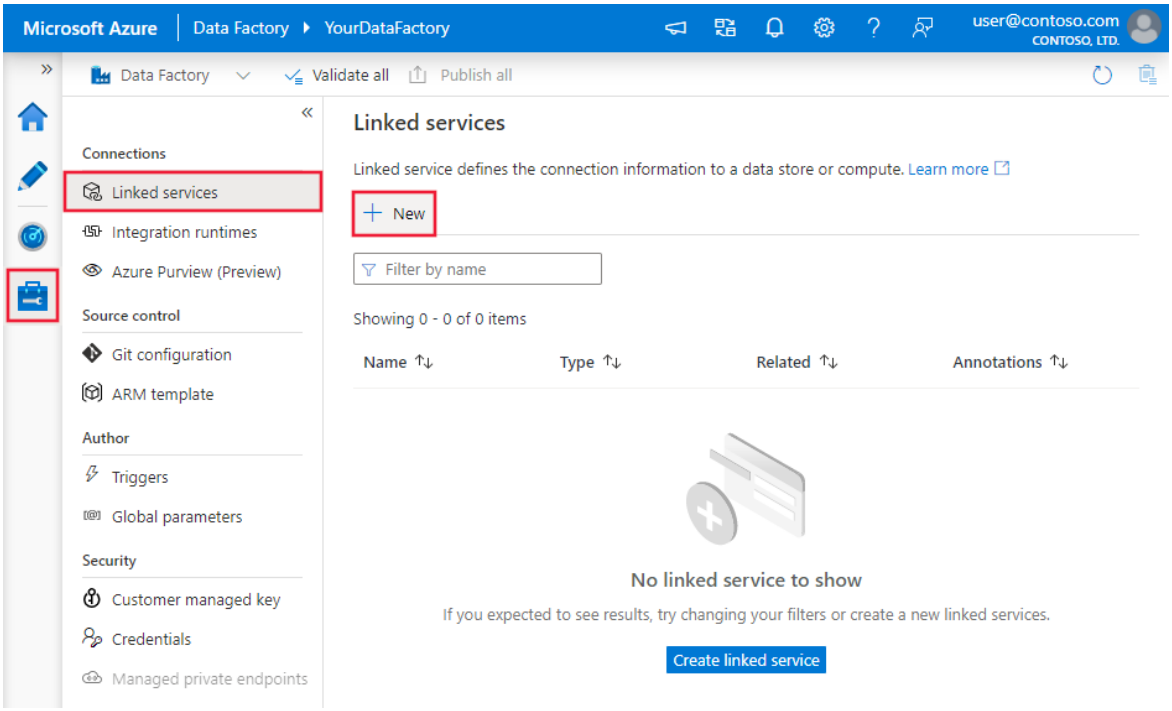
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Oracle Service Cloud using UI

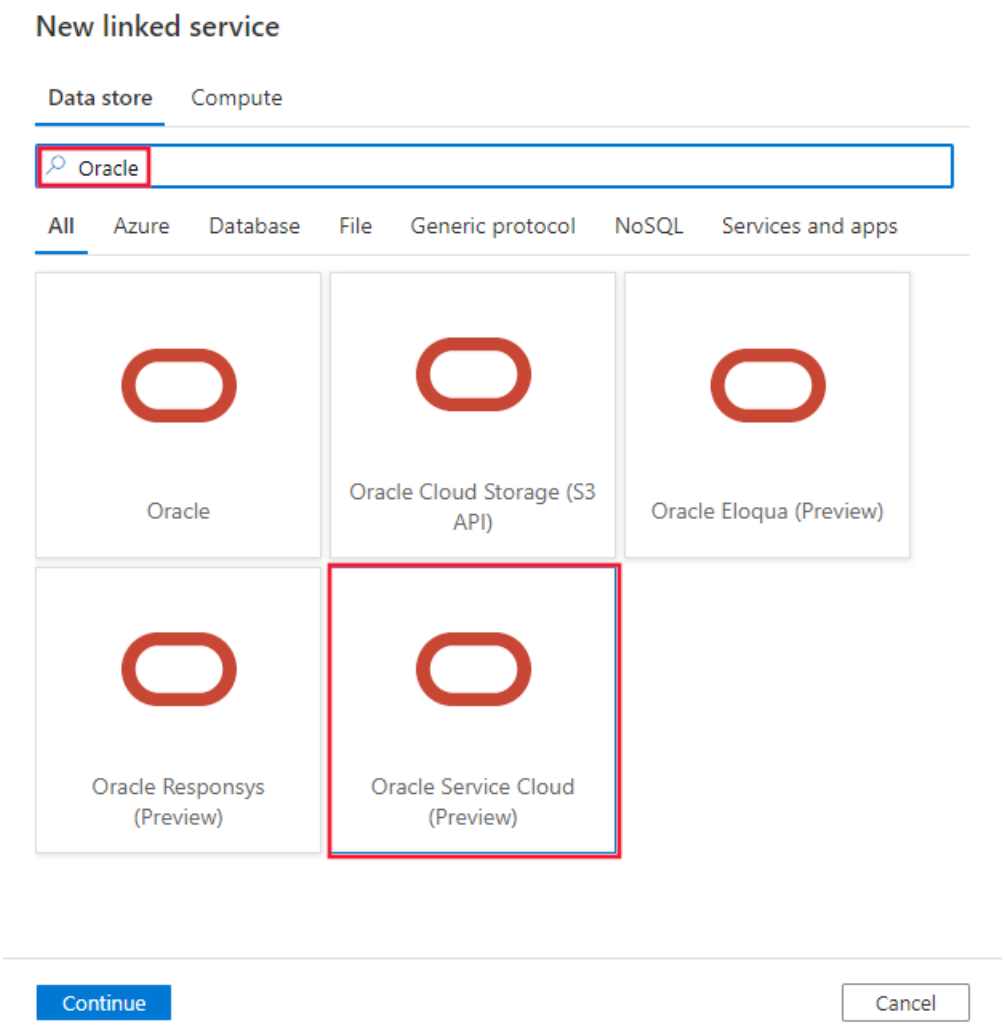
Use the following steps to create a linked service to Oracle Service Cloud in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

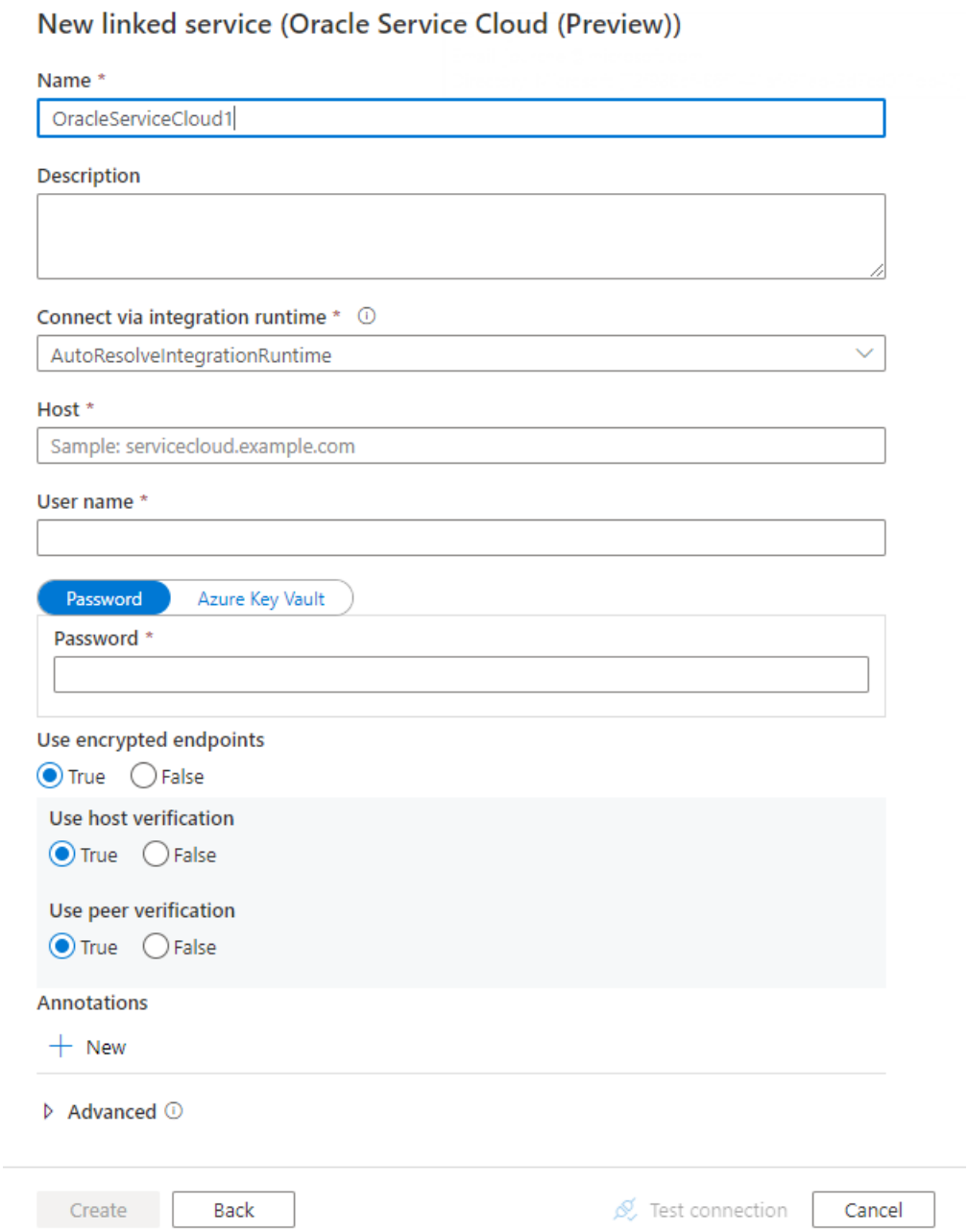
- Azure Data Factory
- Azure Synapse



2. Search for Oracle and select the Oracle Service Cloud connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Oracle Service Cloud connector.

Linked service properties

The following properties are supported for Oracle Service Cloud linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **OracleServiceCloud**	Yes
host	The URL of the Oracle Service Cloud instance.	Yes
username	The user name that you use to access Oracle Service Cloud server.	Yes

PROPERTY	DESCRIPTION	REQUIRED
password	The password corresponding to the user name that you provided in the username key. You can choose to mark this field as a SecureString to store it securely in the service, or store password in Azure Key Vault and let the service copy activity pull from there when performing data copy - learn more from Store credentials in Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
    "name": "OracleServiceCloudLinkedService",
    "properties": {
        "type": "OracleServiceCloud",
        "typeProperties": {
            "host" : "<host>",
            "username" : "<username>",
            "password": {
                 "type": "SecureString",
                 "value": "<password>"
            },
            "useEncryptedEndpoints" : true,
            "useHostVerification" : true,
            "usePeerVerification" : true,
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Oracle Service Cloud dataset.

To copy data from Oracle Service Cloud, set the type property of the dataset to **OracleServiceCloudObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **OracleServiceCloudObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "OracleServiceCloudDataset",
    "properties": {
        "type": "OracleServiceCloudObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<OracleServiceCloud linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Oracle Service Cloud source.

Oracle Service Cloud as source

To copy data from Oracle Service Cloud, set the source type in the copy activity to **OracleServiceCloudSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **OracleServiceCloudSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromOracleServiceCloud",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<OracleServiceCloud input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "OracleServiceCloudSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

ORC format in Azure Data Factory and Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Follow this article when you want to **parse the ORC files or write the data into ORC format**.

ORC format is supported for the following connectors: Amazon S3, Amazon S3 Compatible Storage, Azure Blob, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure Files, File System, FTP, Google Cloud Storage, HDFS, HTTP, Oracle Cloud Storage and SFTP.

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the ORC dataset.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **Orc**.	Yes
location	Location settings of the file(s). Each file-based connector has its own location type and supported properties under `location`. **See details in connector article -> Dataset properties section**.	Yes
compressionCodec	The compression codec to use when writing to ORC files. When reading from ORC files, Data Factories automatically determine the compression codec based on the file metadata. Supported types are **none**, **zlib**, **snappy** (default), and **lzo**. Note currently Copy activity doesn't support LZO when read/write ORC files.	No

Below is an example of ORC dataset on Azure Blob Storage:

```
{
    "name": "OrcDataset",
    "properties": {
        "type": "Orc",
        "linkedServiceName": {
            "referenceName": "<Azure Blob Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "container": "containername",
                "folderPath": "folder/subfolder",
            }
        }
    }
}
```

Note the following points:

- Complex data types (e.g. MAP, LIST, STRUCT) are currently supported only in Data Flows, not in Copy Activity. To use complex types in data flows, do not import the file schema in the dataset, leaving schema blank in the dataset. Then, in the Source transformation, import the projection.
- White space in column name is not supported.

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the ORC source and sink.

ORC as source

The following properties are supported in the copy activity ***source*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **OrcSource**.	Yes
storeSettings	A group of properties on how to read data from a data store. Each file-based connector has its own supported read settings under `storeSettings`. **See details in connector article -> Copy activity properties section**.	No

ORC as sink

The following properties are supported in the copy activity ***sink*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to **OrcSink**.	Yes
formatSettings	A group of properties. Refer to **ORC write settings** table below.	No

PROPERTY	DESCRIPTION	REQUIRED
storeSettings	A group of properties on how to write data to a data store. Each file-based connector has its own supported write settings under `storeSettings`. **See details in connector article -> Copy activity properties section**.	No

Supported **ORC write settings** under `formatSettings`:

PROPERTY	DESCRIPTION	REQUIRED
type	The type of formatSettings must be set to **OrcWriteSettings**.	Yes
maxRowsPerFile	When writing data into a folder, you can choose to write to multiple files and specify the max rows per file.	No
fileNamePrefix	Applicable when `maxRowsPerFile` is configured. Specify the file name prefix when writing data to multiple files, resulted in this pattern: `<fileNamePrefix>_00000.<fileExtension>`. If not specified, file name prefix will be auto generated. This property does not apply when source is file-based store or partition-option-enabled data store.	No

Mapping data flow properties

In mapping data flows, you can read and write to ORC format in the following data stores: Azure Blob Storage, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2 and SFTP, and you can read ORC format in Amazon S3.

You can point to ORC files either using ORC dataset or using an inline dataset.

Source properties

The below table lists the properties supported by an ORC source. You can edit these properties in the **Source options** tab.

When using inline dataset, you will see additional file settings, which are the same as the properties described in dataset properties section.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Format	Format must be `orc`	yes	`orc`	format

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Wild card paths	All files matching the wildcard path will be processed. Overrides the folder and file path set in the dataset.	no	String[]	wildcardPaths
Partition root path	For file data that is partitioned, you can enter a partition root path in order to read partitioned folders as columns	no	String	partitionRootPath
List of files	Whether your source is pointing to a text file that lists files to process	no	`true` or `false`	fileList
Column to store file name	Create a new column with the source file name and path	no	String	rowUrlColumn
After completion	Delete or move the files after processing. File path starts from the container root	no	Delete: `true` or `false` Move: `[<from>, <to>]`	purgeFiles moveFiles
Filter by last modified	Choose to filter files based upon when they were last altered	no	Timestamp	modifiedAfter modifiedBefore
Allow no files found	If true, an error is not thrown if no files are found	no	`true` or `false`	ignoreNoFilesFound

Source example

The associated data flow script of an ORC source configuration is:

```
source(allowSchemaDrift: true,
    validateSchema: false,
    rowUrlColumn: 'fileName',
    format: 'orc') ~> OrcSource
```

Sink properties

The below table lists the properties supported by an ORC sink. You can edit these properties in the **Settings** tab.

When using inline dataset, you will see additional file settings, which are the same as the properties described in dataset properties section.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Format	Format must be `orc`	yes	`orc`	format
Clear the folder	If the destination folder is cleared prior to write	no	`true` or `false`	truncate
File name option	The naming format of the data written. By default, one file per partition in format `part-#####-tid-<guid>`	no	Pattern: String Per partition: String[] As data in column: String Output to single file: `['<fileName>']`	filePattern partitionFileNames rowUrlColumn partitionFileNames

Sink example

The associated data flow script of an ORC sink configuration is:

```
OrcSource sink(
    format: 'orc',
    filePattern:'output[n].orc',
    truncate: true,
    allowSchemaDrift: true,
    validateSchema: false,
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> OrcSink
```

Using Self-hosted Integration Runtime

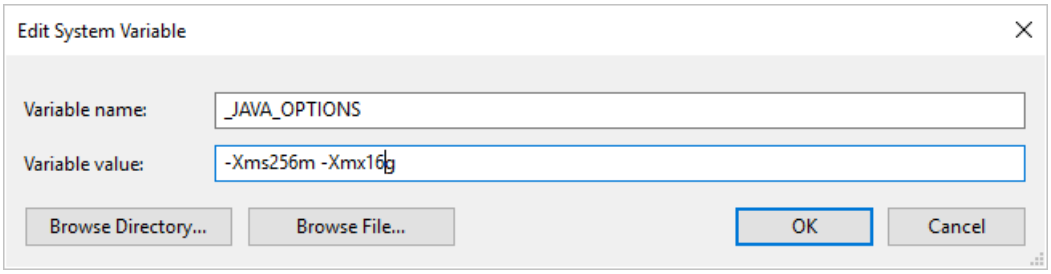
> **IMPORTANT**
> For copy empowered by Self-hosted Integration Runtime e.g. between on-premises and cloud data stores, if you are not copying ORC files **as-is**, you need to install the **64-bit JRE 8 (Java Runtime Environment) or OpenJDK** and **Microsoft Visual C++ 2010 Redistributable Package** on your IR machine. Check the following paragraph with more details.

For copy running on Self-hosted IR with ORC file serialization/deserialization, the service locates the Java runtime by firstly checking the registry `(SOFTWARE\JavaSoft\Java Runtime Environment\{Current Version}\JavaHome)` for JRE, if not found, secondly checking system variable `JAVA_HOME` for OpenJDK.

- **To use JRE**: The 64-bit IR requires 64-bit JRE. You can find it from here.
- **To use OpenJDK**: It's supported since IR version 3.13. Package the jvm.dll with all other required assemblies of OpenJDK into Self-hosted IR machine, and set system environment variable JAVA_HOME accordingly.
- **To install Visual C++ 2010 Redistributable Package**: Visual C++ 2010 Redistributable Package is not installed with self-hosted IR installations. You can find it from here.

TIP

If you copy data to/from ORC format using Self-hosted Integration Runtime and hit error saying "An error occurred when invoking java, message: **java.lang.OutOfMemoryError:Java heap space**", you can add an environment variable `_JAVA_OPTIONS` in the machine that hosts the Self-hosted IR to adjust the min/max heap size for JVM to empower such copy, then rerun the pipeline.



Example: set variable `_JAVA_OPTIONS` with value `-Xms256m -Xmx16g`. The flag `Xms` specifies the initial memory allocation pool for a Java Virtual Machine (JVM), while `Xmx` specifies the maximum memory allocation pool. This means that JVM will be started with `Xms` amount of memory and will be able to use a maximum of `Xmx` amount of memory. By default, the service uses min 64 MB and max 1G.

Next steps

- Copy activity overview
- Lookup activity
- GetMetadata activity

Parquet format in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Follow this article when you want to **parse the Parquet files or write the data into Parquet format**.

Parquet format is supported for the following connectors:

- Amazon S3
- Amazon S3 Compatible Storage
- Azure Blob
- Azure Data Lake Storage Gen1
- Azure Data Lake Storage Gen2
- Azure Files
- File System
- FTP
- Google Cloud Storage
- HDFS
- HTTP
- Oracle Cloud Storage
- SFTP

For a list of supported features for all available connectors, visit the Connectors Overview article.

Using Self-hosted Integration Runtime

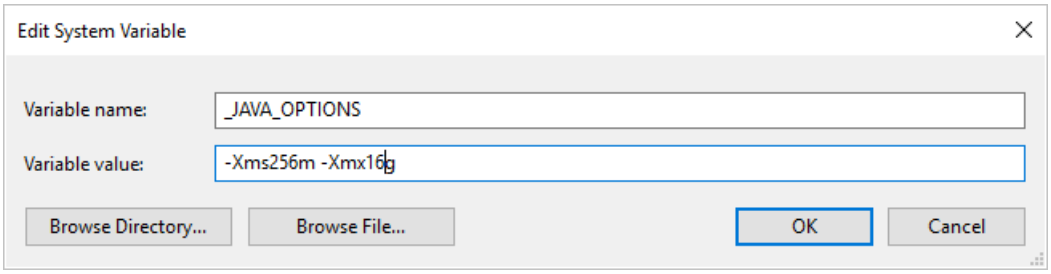
> **IMPORTANT**
> For copy empowered by Self-hosted Integration Runtime e.g. between on-premises and cloud data stores, if you are not copying Parquet files **as-is**, you need to install the **64-bit JRE 8 (Java Runtime Environment) or OpenJDK** on your IR machine. Check the following paragraph with more details.

For copy running on Self-hosted IR with Parquet file serialization/deserialization, the service locates the Java runtime by firstly checking the registry `(SOFTWARE\JavaSoft\Java Runtime Environment\{Current Version}\JavaHome)` for JRE, if not found, secondly checking system variable `JAVA_HOME` for OpenJDK.

- **To use JRE**: The 64-bit IR requires 64-bit JRE. You can find it from here.
- **To use OpenJDK**: It's supported since IR version 3.13. Package the jvm.dll with all other required assemblies of OpenJDK into Self-hosted IR machine, and set system environment variable JAVA_HOME accordingly, and then restart Self-hosted IR for taking effect immediately.

TIP

If you copy data to/from Parquet format using Self-hosted Integration Runtime and hit error saying "An error occurred when invoking java, message: **java.lang.OutOfMemoryError:Java heap space**", you can add an environment variable `_JAVA_OPTIONS` in the machine that hosts the Self-hosted IR to adjust the min/max heap size for JVM to empower such copy, then rerun the pipeline.



Example: set variable `_JAVA_OPTIONS` with value `-Xms256m -Xmx16g`. The flag `Xms` specifies the initial memory allocation pool for a Java Virtual Machine (JVM), while `Xmx` specifies the maximum memory allocation pool. This means that JVM will be started with `Xms` amount of memory and will be able to use a maximum of `Xmx` amount of memory. By default, the service uses min 64 MB and max 1G.

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the Parquet dataset.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **Parquet**.	Yes
location	Location settings of the file(s). Each file-based connector has its own location type and supported properties under `location`. **See details in connector article -> Dataset properties section**.	Yes
compressionCodec	The compression codec to use when writing to Parquet files. When reading from Parquet files, Data Factories automatically determine the compression codec based on the file metadata. Supported types are "**none**", "**gzip**", "**snappy**" (default), and "**lzo**". Note currently Copy activity doesn't support LZO when read/write Parquet files.	No

NOTE

White space in column name is not supported for Parquet files.

Below is an example of Parquet dataset on Azure Blob Storage:

```
{
    "name": "ParquetDataset",
    "properties": {
        "type": "Parquet",
        "linkedServiceName": {
            "referenceName": "<Azure Blob Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "container": "containername",
                "folderPath": "folder/subfolder",
            },
            "compressionCodec": "snappy"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the Parquet source and sink.

Parquet as source

The following properties are supported in the copy activity ***source*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **ParquetSource**.	Yes
storeSettings	A group of properties on how to read data from a data store. Each file-based connector has its own supported read settings under `storeSettings`. **See details in connector article -> Copy activity properties section**.	No

Parquet as sink

The following properties are supported in the copy activity ***sink*** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to **ParquetSink**.	Yes
formatSettings	A group of properties. Refer to **Parquet write settings** table below.	No
storeSettings	A group of properties on how to write data to a data store. Each file-based connector has its own supported write settings under `storeSettings`. **See details in connector article -> Copy activity properties section**.	No

Supported **Parquet write settings** under `formatSettings`:

PROPERTY	DESCRIPTION	REQUIRED
type	The type of formatSettings must be set to **ParquetWriteSettings**.	Yes
maxRowsPerFile	When writing data into a folder, you can choose to write to multiple files and specify the max rows per file.	No
fileNamePrefix	Applicable when `maxRowsPerFile` is configured. Specify the file name prefix when writing data to multiple files, resulted in this pattern: `<fileNamePrefix>_00000.<fileExtension>`. If not specified, file name prefix will be auto generated. This property does not apply when source is file-based store or partition-option-enabled data store.	No

Mapping data flow properties

In mapping data flows, you can read and write to parquet format in the following data stores: Azure Blob Storage, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2 and SFTP, and you can read parquet format in Amazon S3.

Source properties

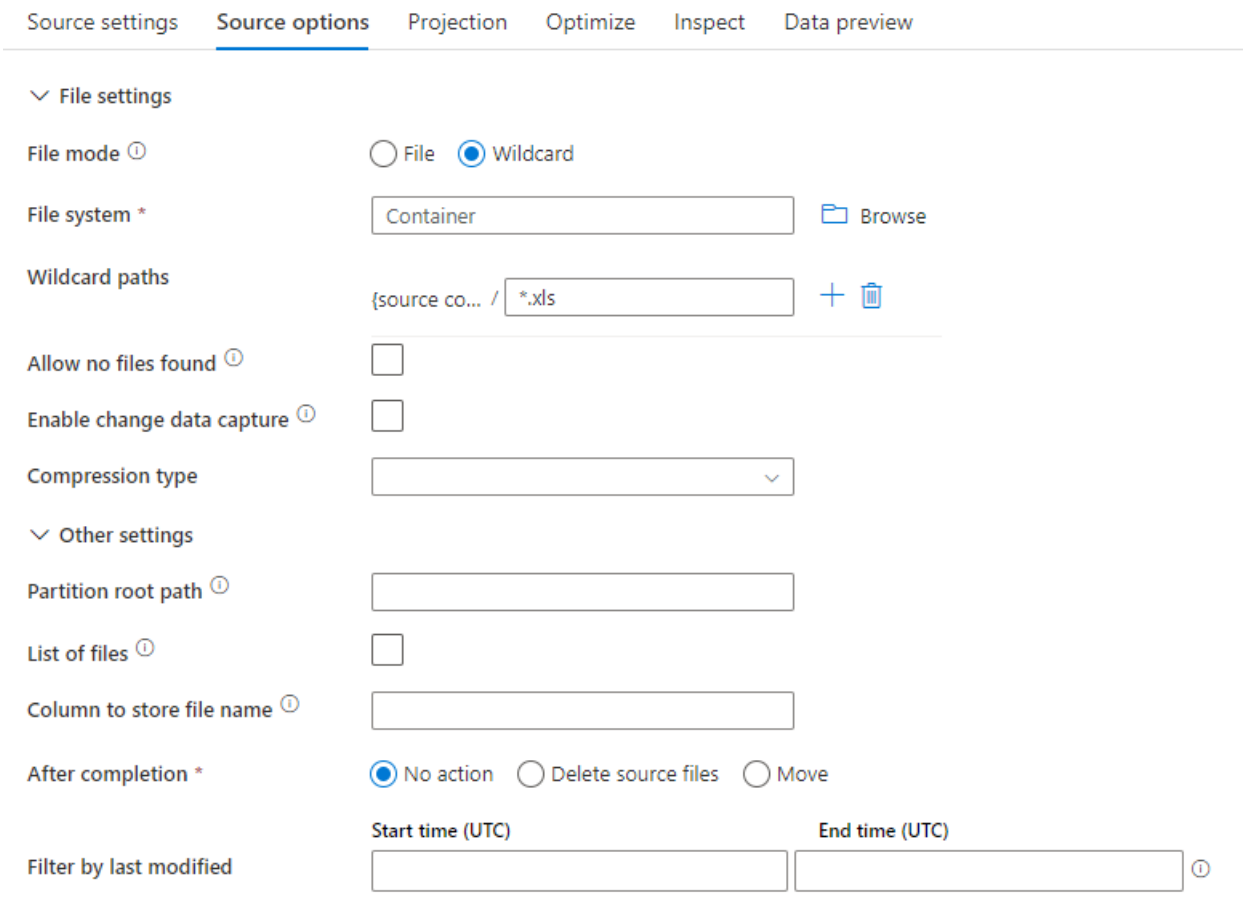
The below table lists the properties supported by a parquet source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Format	Format must be `parquet`	yes	`parquet`	format
Wild card paths	All files matching the wildcard path will be processed. Overrides the folder and file path set in the dataset.	no	String[]	wildcardPaths
Partition root path	For file data that is partitioned, you can enter a partition root path in order to read partitioned folders as columns	no	String	partitionRootPath
List of files	Whether your source is pointing to a text file that lists files to process	no	`true` or `false`	fileList

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Column to store file name	Create a new column with the source file name and path	no	String	rowUrlColumn
After completion	Delete or move the files after processing. File path starts from the container root	no	Delete: `true` or `false` Move: `[<from>, <to>]`	purgeFiles moveFiles
Filter by last modified	Choose to filter files based upon when they were last altered	no	Timestamp	modifiedAfter modifiedBefore
Allow no files found	If true, an error is not thrown if no files are found	no	`true` or `false`	ignoreNoFilesFound

Source example

The below image is an example of a parquet source configuration in mapping data flows.



The associated data flow script is:

```
source(allowSchemaDrift: true,
    validateSchema: false,
    rowUrlColumn: 'fileName',
    format: 'parquet') ~> ParquetSource
```

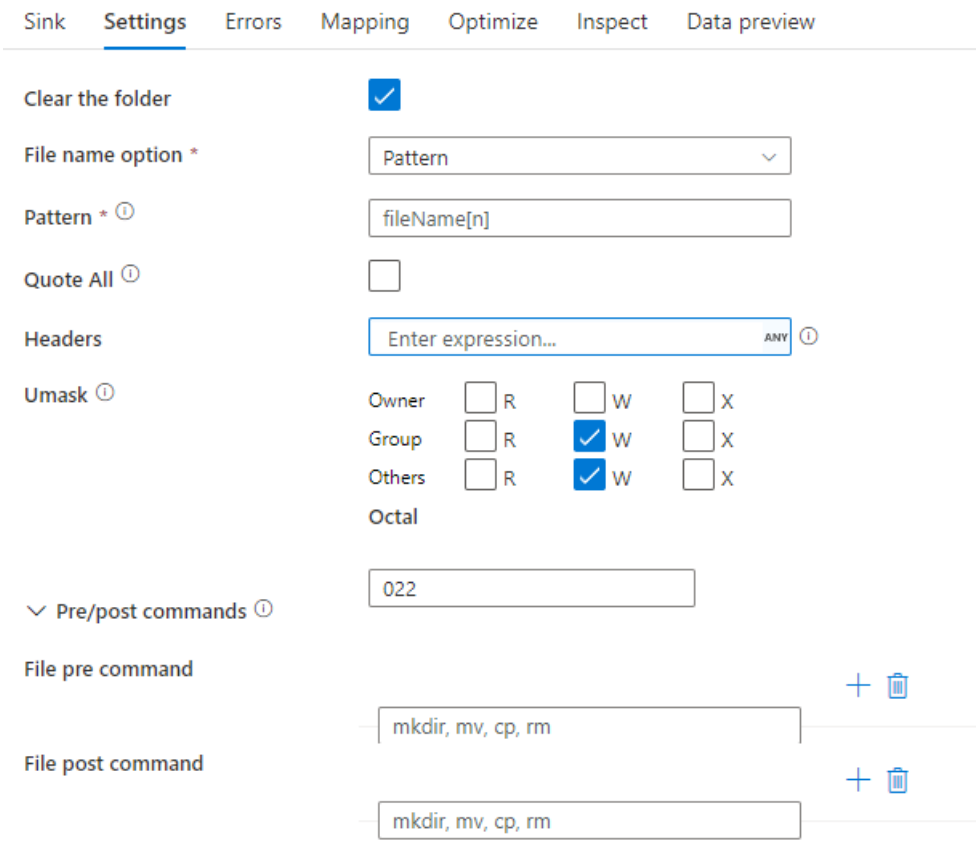
Sink properties

The below table lists the properties supported by a parquet sink. You can edit these properties in the **Settings** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Format	Format must be `parquet`	yes	`parquet`	format
Clear the folder	If the destination folder is cleared prior to write	no	`true` or `false`	truncate
File name option	The naming format of the data written. By default, one file per partition in format `part-#####-tid-<guid>`	no	Pattern: String Per partition: String[] As data in column: String Output to single file: `['<fileName>']`	filePattern partitionFileNames rowUrlColumn partitionFileNames

Sink example

The below image is an example of a parquet sink configuration in mapping data flows.



The associated data flow script is:

```
ParquetSource sink(
    format: 'parquet',
    filePattern:'output[n].parquet',
    truncate: true,
    allowSchemaDrift: true,
    validateSchema: false,
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> ParquetSink
```

Data type support

Parquet complex data types (e.g. MAP, LIST, STRUCT) are currently supported only in Data Flows, not in Copy Activity. To use complex types in data flows, do not import the file schema in the dataset, leaving schema blank in the dataset. Then, in the Source transformation, import the projection.

Next steps

- Copy activity overview
- Mapping data flow
- Lookup activity
- GetMetadata activity

Copy data from PayPal using Azure Data Factory or Synapse Analytics (Preview)

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from PayPal. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This PayPal connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

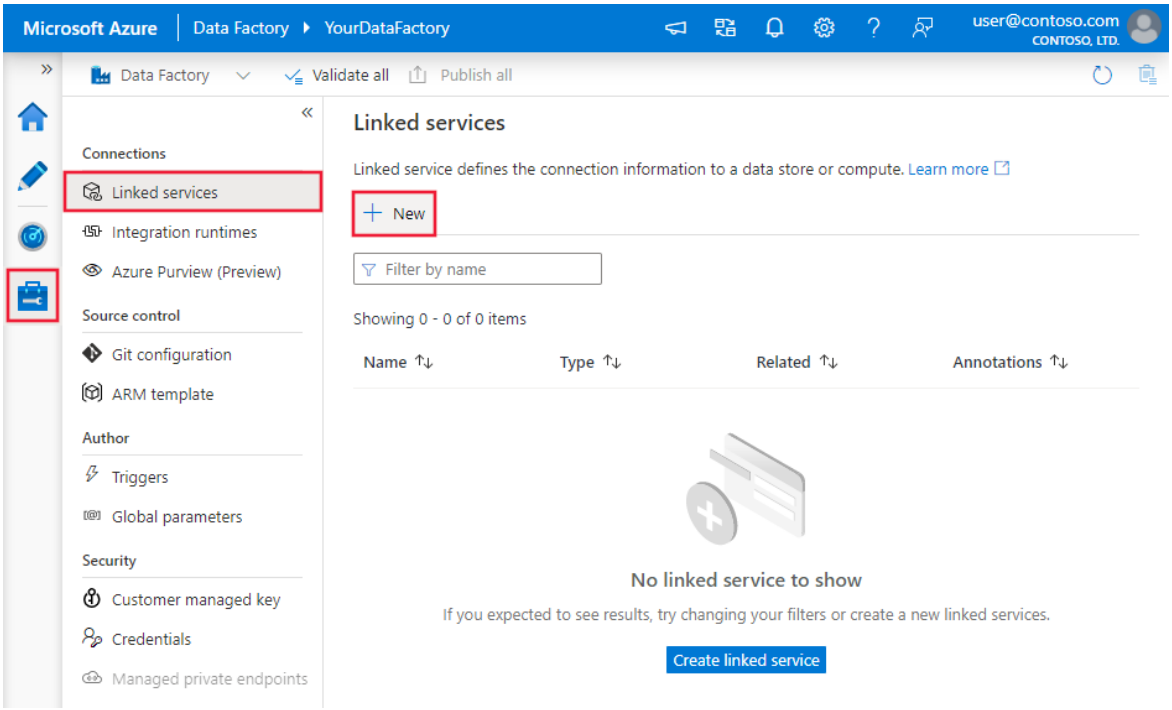
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to PayPal using UI

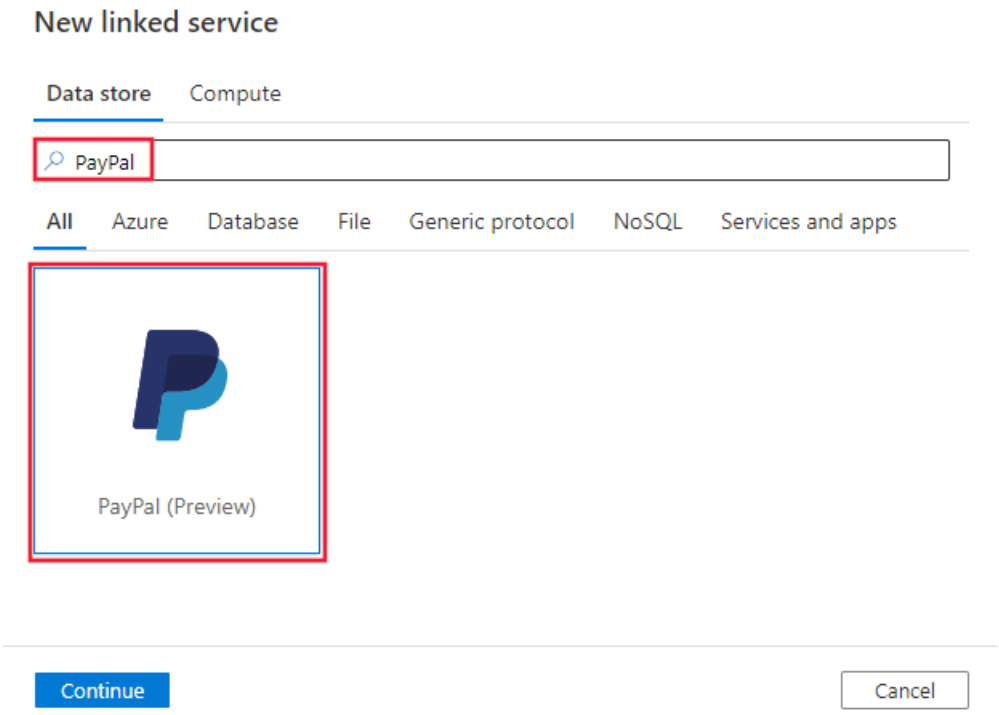
Use the following steps to create a linked service to PayPal in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

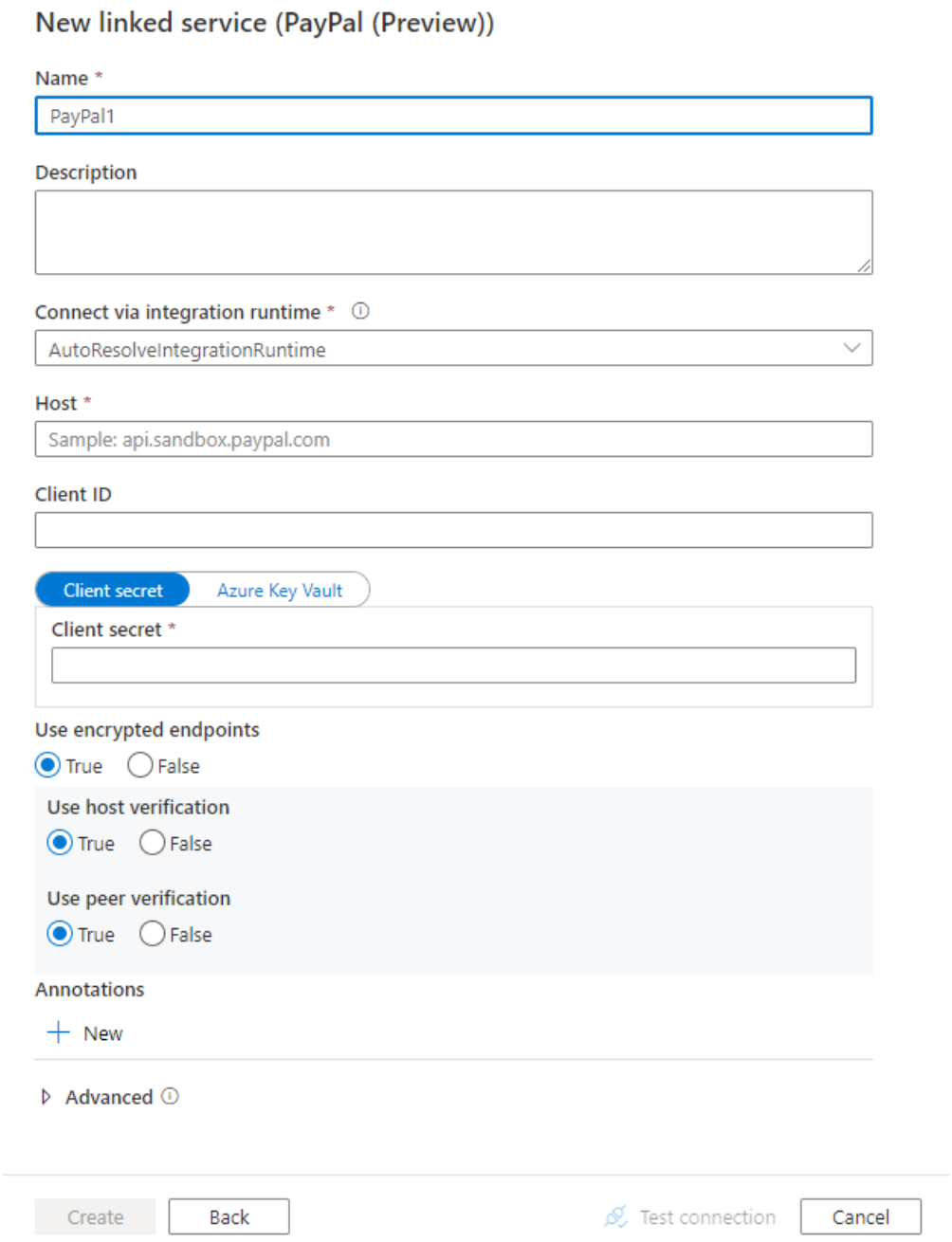
- Azure Data Factory
- Azure Synapse



2. Search for PayPal and select the PayPal connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to PayPal connector.

Linked service properties

The following properties are supported for PayPal linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **PayPal**	Yes
host	The URL of the PayPal instance. (that is, api.sandbox.paypal.com)	Yes

PROPERTY	DESCRIPTION	REQUIRED
clientId	The client ID associated with your PayPal application.	Yes
clientSecret	The client secret associated with your PayPal application. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
    "name": "PayPalLinkedService",
    "properties": {
        "type": "PayPal",
        "typeProperties": {
            "host" : "api.sandbox.paypal.com",
            "clientId" : "<clientId>",
            "clientSecret": {
                 "type": "SecureString",
                 "value": "<clientSecret>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by PayPal dataset.

To copy data from PayPal, set the type property of the dataset to **PayPalObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **PayPalObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "PayPalDataset",
    "properties": {
        "type": "PayPalObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<PayPal linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by PayPal source.

PayPal as source

To copy data from PayPal, set the source type in the copy activity to **PayPalSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **PayPalSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM Payment_Experience"` .	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromPayPal",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<PayPal input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "PayPalSource",
                "query": "SELECT * FROM Payment_Experience"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Phoenix using Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Phoenix. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Phoenix connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK

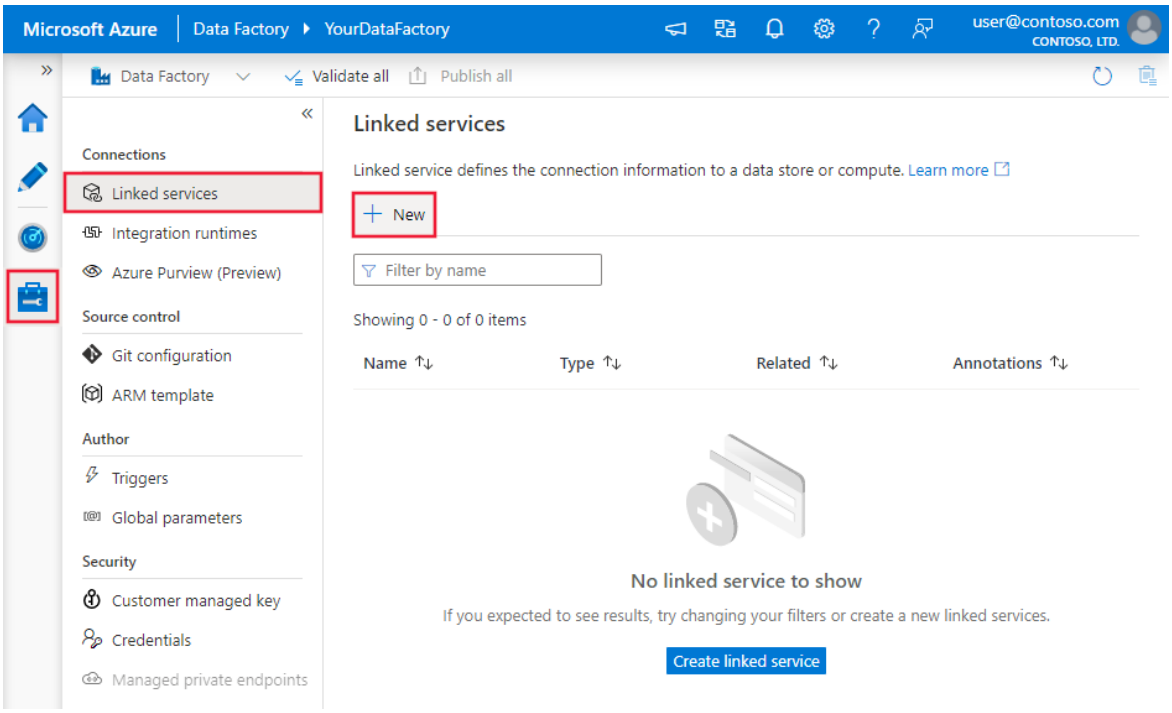
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Phoenix using UI

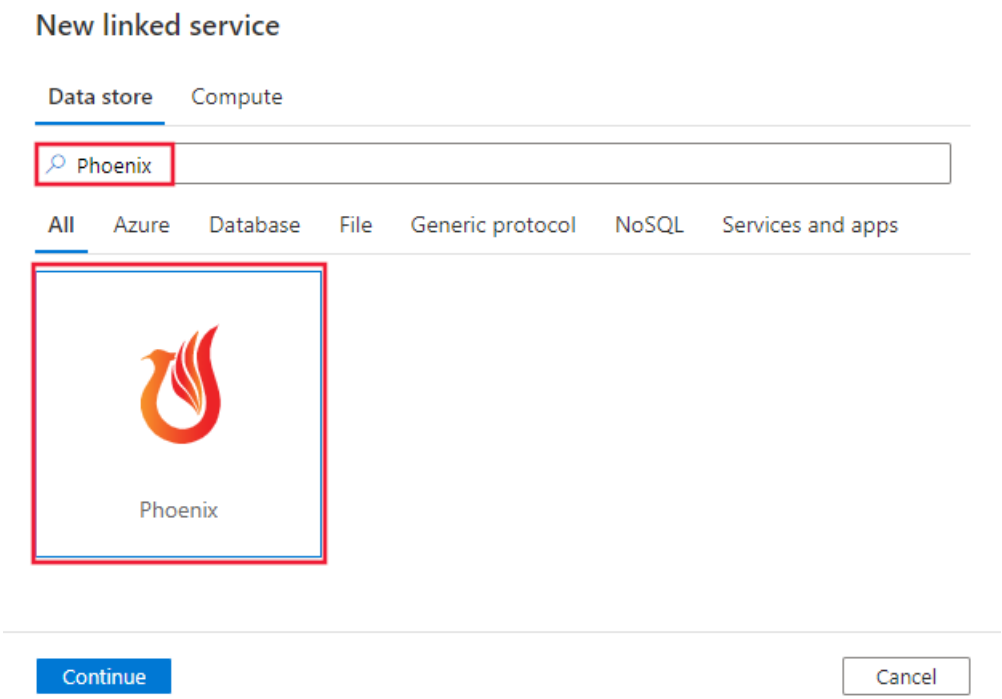
Use the following steps to create a linked service to Phoenix in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

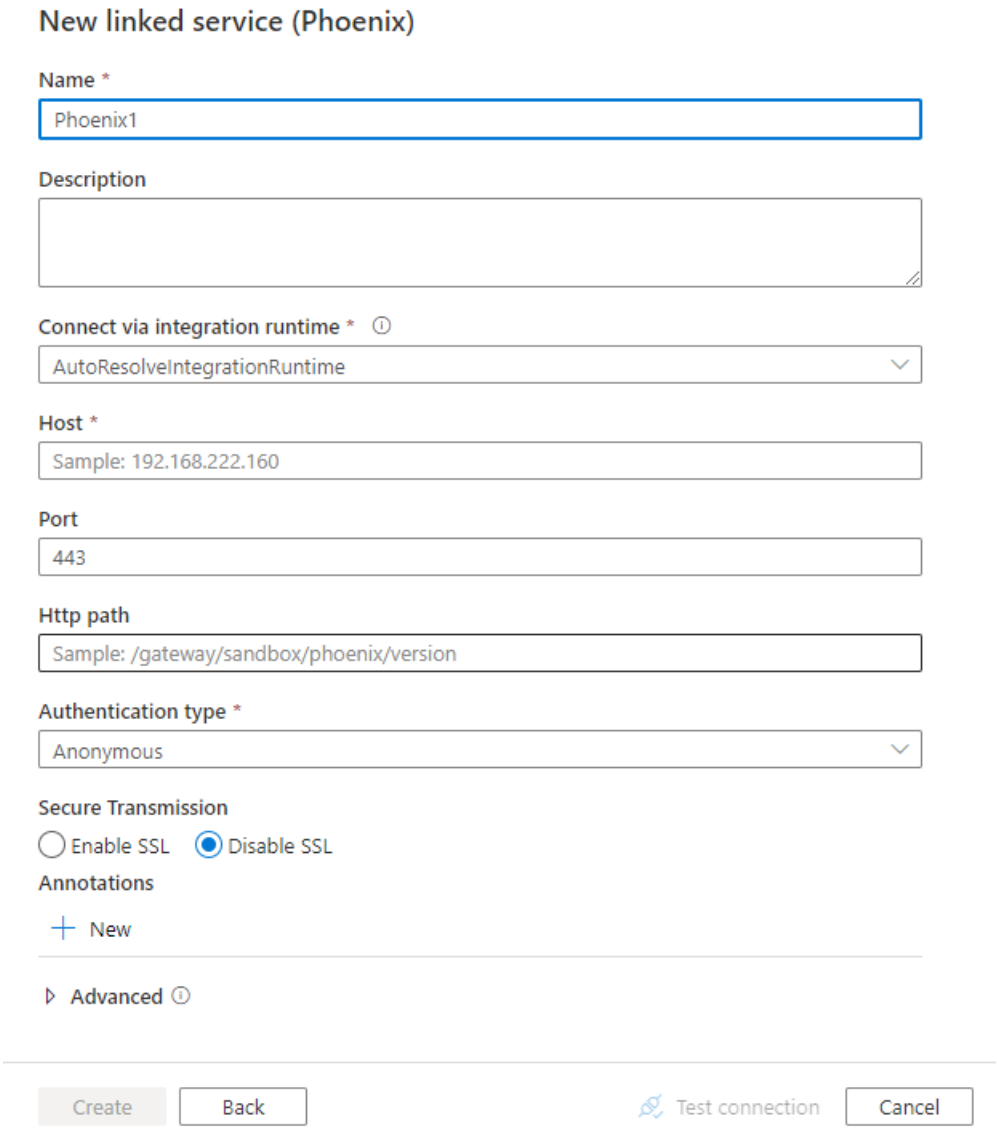
   - Azure Data Factory
   - Azure Synapse



2. Search for Phoenix and select the Phoenix connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Phoenix connector.

Linked service properties

The following properties are supported for Phoenix linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Phoenix**	Yes
host	The IP address or host name of the Phoenix server. (that is, 192.168.222.160)	Yes

PROPERTY	DESCRIPTION	REQUIRED
port	The TCP port that the Phoenix server uses to listen for client connections. The default value is 8765. If you connect to Azure HDInsights, specify port as 443.	No
httpPath	The partial URL corresponding to the Phoenix server. (that is, /gateway/sandbox/phoenix/version). Specify `/hbasephoenix0` if using HDInsights cluster.	No
authenticationType	The authentication mechanism used to connect to the Phoenix server. Allowed values are: **Anonymous**, **UsernameAndPassword**, **WindowsAzureHDInsightService**	Yes
username	The user name used to connect to the Phoenix server.	No
password	The password corresponding to the user name. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
enableSsl	Specifies whether the connections to the server are encrypted using TLS. The default value is false.	No
trustedCertPath	The full path of the .pem file containing trusted CA certificates for verifying the server when connecting over TLS. This property can only be set when using TLS on self-hosted IR. The default value is the cacerts.pem file installed with the IR.	No
useSystemTrustStore	Specifies whether to use a CA certificate from the system trust store or from a specified PEM file. The default value is false.	No
allowHostNameCNMismatch	Specifies whether to require a CA-issued TLS/SSL certificate name to match the host name of the server when connecting over TLS. The default value is false.	No
allowSelfSignedServerCert	Specifies whether to allow self-signed certificates from the server. The default value is false.	No

PROPERTY	DESCRIPTION	REQUIRED
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

> **NOTE**
> If your cluster doesn't support sticky session e.g. HDInsight, explicitly add node index at the end of the http path setting, e.g. specify `/hbasephoenix0` instead of `/hbasephoenix`.

Example:

```
{
    "name": "PhoenixLinkedService",
    "properties": {
        "type": "Phoenix",
        "typeProperties": {
            "host" : "<cluster>.azurehdinsight.net",
            "port" : "443",
            "httpPath" : "/hbasephoenix0",
            "authenticationType" : "WindowsAzureHDInsightService",
            "username" : "<username>",
            "password": {
                 "type": "SecureString",
                 "value": "<password>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Phoenix dataset.

To copy data from Phoenix, set the type property of the dataset to **PhoenixObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **PhoenixObject**	Yes
schema	Name of the schema.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table with schema. This property is supported for backward compatibility. Use `schema` and `table` for new workload.	No (if "query" in activity source is specified)

Example

```
{
    "name": "PhoenixDataset",
    "properties": {
        "type": "PhoenixObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Phoenix linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Phoenix source.

Phoenix as source

To copy data from Phoenix, set the source type in the copy activity to **PhoenixSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **PhoenixSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromPhoenix",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Phoenix input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "PhoenixSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from PostgreSQL using Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from a PostgreSQL database. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This PostgreSQL connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

Specifically, this PostgreSQL connector supports PostgreSQL **version 7.4 and above**.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

The Integration Runtime provides a built-in PostgreSQL driver starting from version 3.7, therefore you don't need to manually install any driver.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK

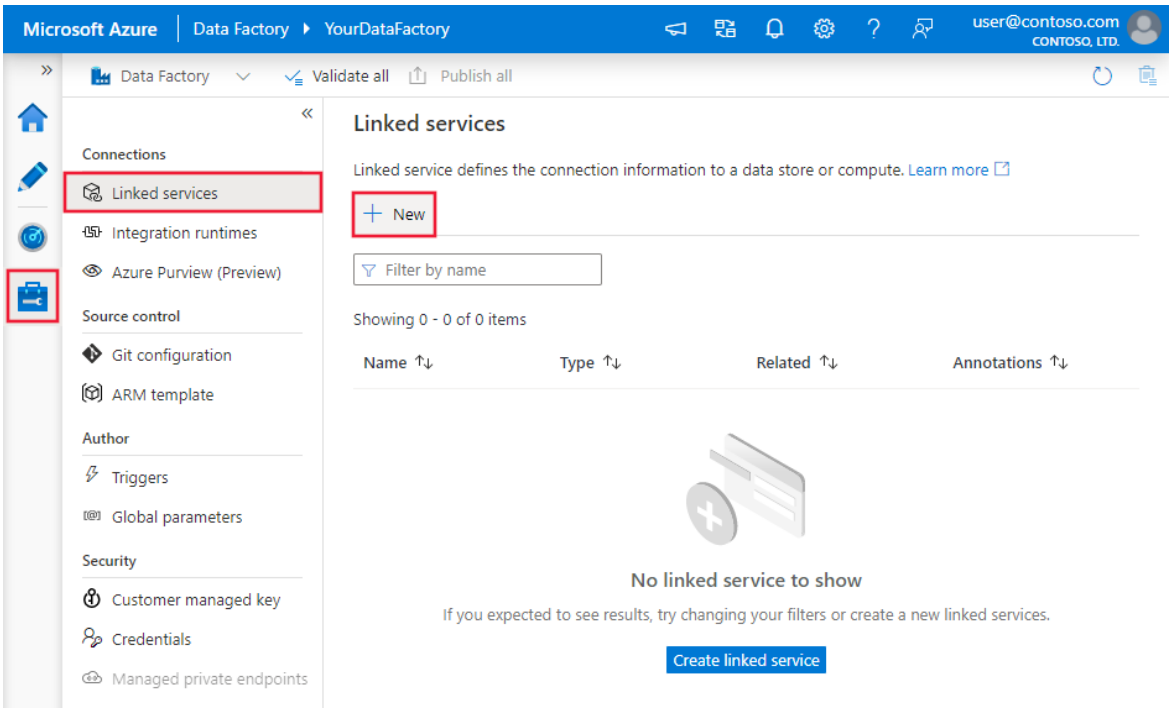
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to PostgreSQL using UI

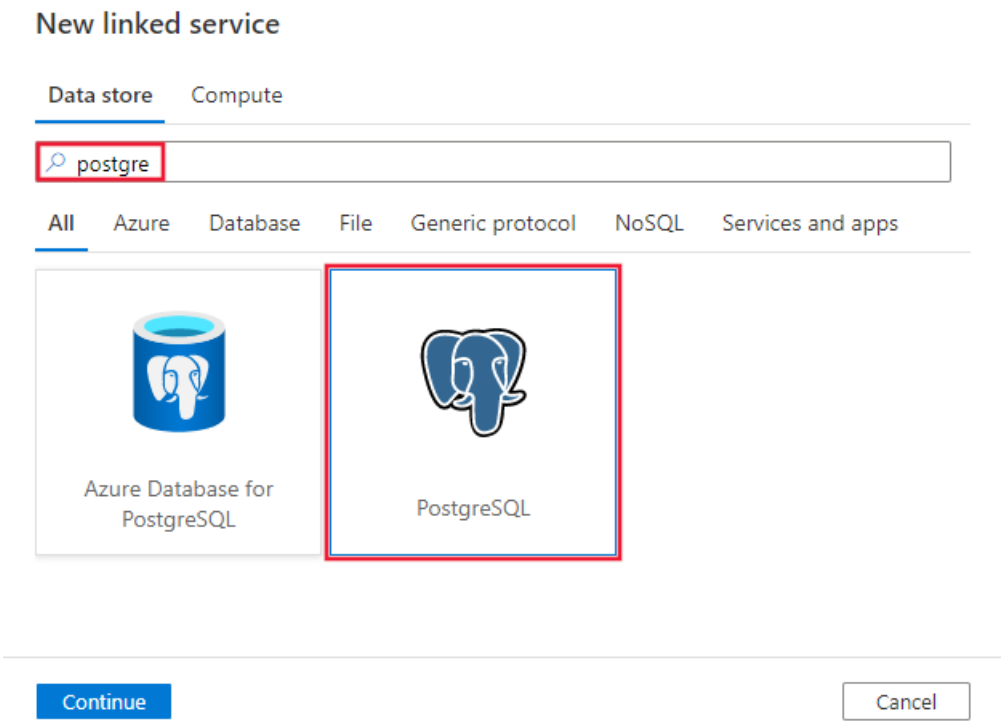
Use the following steps to create a linked service to PostgreSQL in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

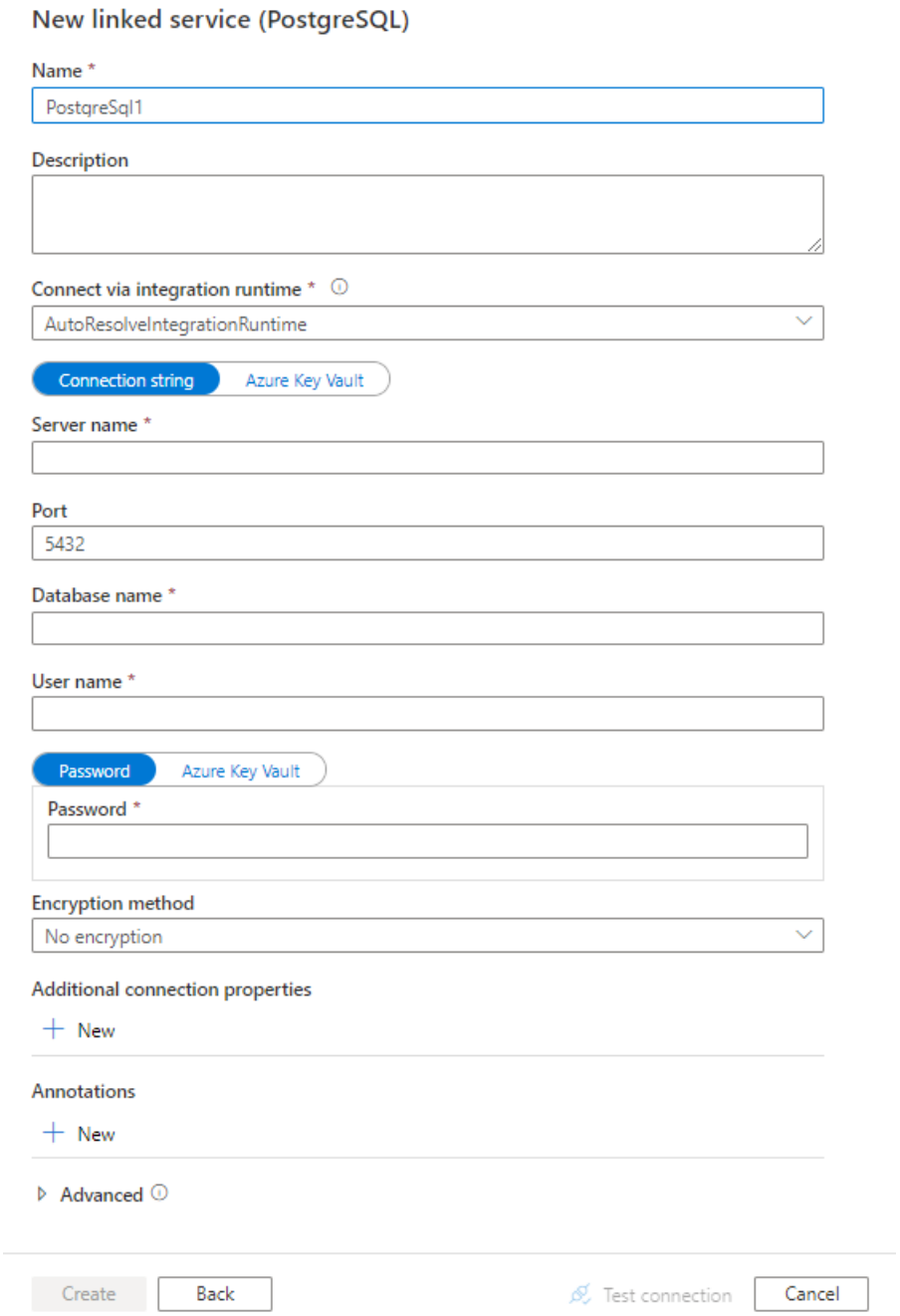
   - Azure Data Factory
   - Azure Synapse



2. Search for Postgre and select the PostgreSQL connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to PostgreSQL connector.

Linked service properties

The following properties are supported for PostgreSQL linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **PostgreSql**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	An ODBC connection string to connect to Azure Database for PostgreSQL. You can also put password in Azure Key Vault and pull the `password` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

A typical connection string is `Server=<server>;Database=<database>;Port=<port>;UID=<username>;Password=<Password>`. More properties you can set per your case:

PROPERTY	DESCRIPTION	OPTIONS	REQUIRED
EncryptionMethod (EM)	The method the driver uses to encrypt data sent between the driver and the database server. E.g., `EncryptionMethod=<0/1/6>;`	0 (No Encryption) **(Default)** / 1 (SSL) / 6 (RequestSSL)	No
ValidateServerCertificate (VSC)	Determines whether the driver validates the certificate that is sent by the database server when SSL encryption is enabled (Encryption Method=1). E.g., `ValidateServerCertificate=<0/1>;`	0 (Disabled) **(Default)** / 1 (Enabled)	No

Example:

```
{
    "name": "PostgreSqlLinkedService",
    "properties": {
        "type": "PostgreSql",
        "typeProperties": {
            "connectionString": "Server=<server>;Database=<database>;Port=<port>;UID=<username>;Password=
<Password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "PostgreSqlLinkedService",
    "properties": {
        "type": "PostgreSql",
        "typeProperties": {
            "connectionString": "Server=<server>;Database=<database>;Port=<port>;UID=<username>;",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

If you were using PostgreSQL linked service with the following payload, it is still supported as-is, while you are suggested to use the new one going forward.

Previous payload:

```
{
    "name": "PostgreSqlLinkedService",
    "properties": {
        "type": "PostgreSql",
        "typeProperties": {
            "server": "<server>",
            "database": "<database>",
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by PostgreSQL dataset.

To copy data from PostgreSQL, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **PostgreSqlTable**	Yes

PROPERTY	DESCRIPTION	REQUIRED
schema	Name of the schema.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table with schema. This property is supported for backward compatibility. Use `schema` and `table` for new workload.	No (if "query" in activity source is specified)

Example

```
{
    "name": "PostgreSQLDataset",
    "properties":
    {
        "type": "PostgreSqlTable",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<PostgreSQL linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

If you were using `RelationalTable` typed dataset, it's still supported as-is, while you are suggested to use the new one going forward.

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by PostgreSQL source.

PostgreSQL as source

To copy data from PostgreSQL, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **PostgreSqlSource**	Yes
query	Use the custom SQL query to read data. For example: `"query": "SELECT * FROM \"MySchema\".\"MyTable\""` .	No (if "tableName" in dataset is specified)

> **NOTE**
> Schema and table names are case-sensitive. Enclose them in `""` (double quotes) in the query.

Example:

```
"activities":[
    {
        "name": "CopyFromPostgreSQL",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<PostgreSQL input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "PostgreSqlSource",
                "query": "SELECT * FROM \"MySchema\".\"MyTable\""
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

If you were using `RelationalSource` typed source, it is still supported as-is, while you are suggested to use the new one going forward.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Presto using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Presto. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Presto connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

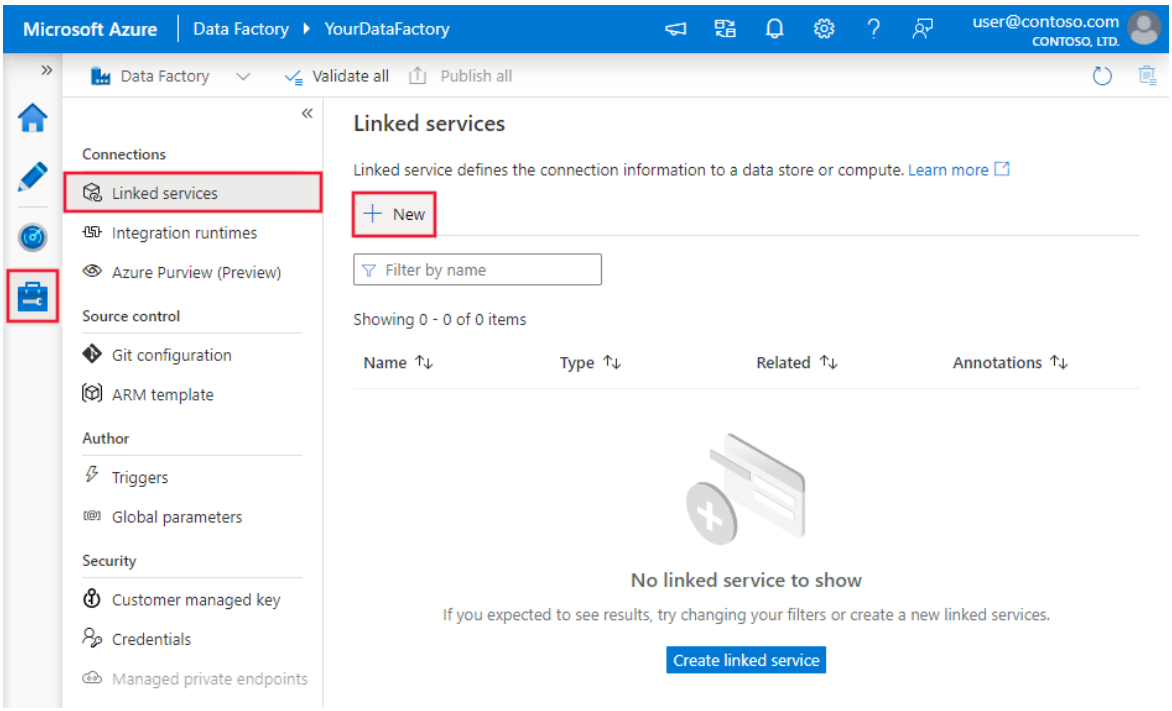
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Presto using UI

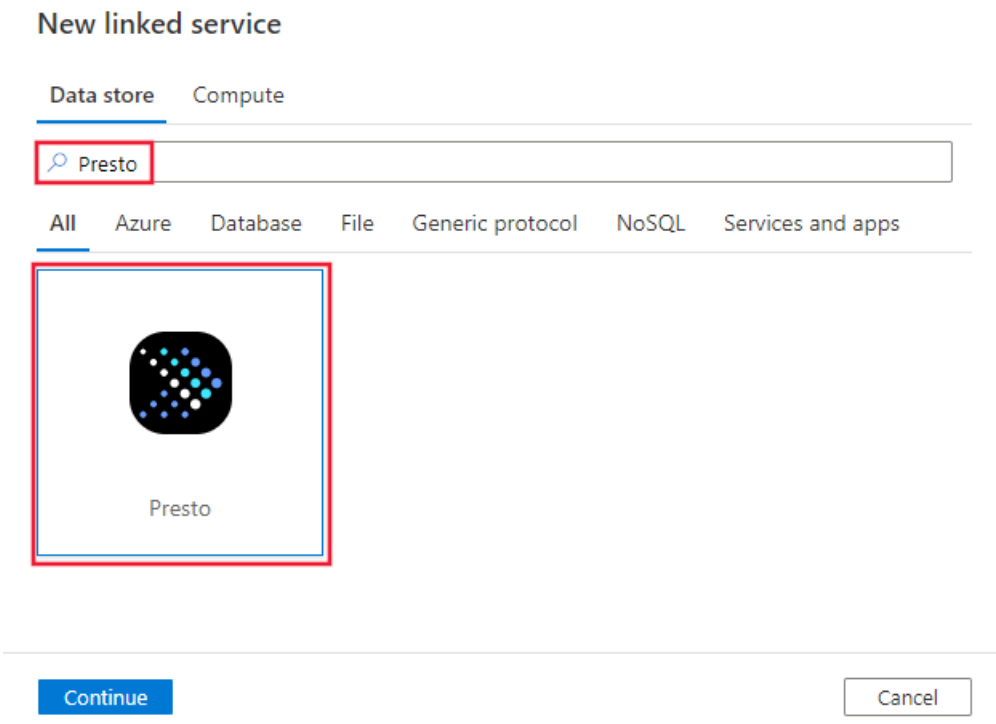
Use the following steps to create a linked service to Presto in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

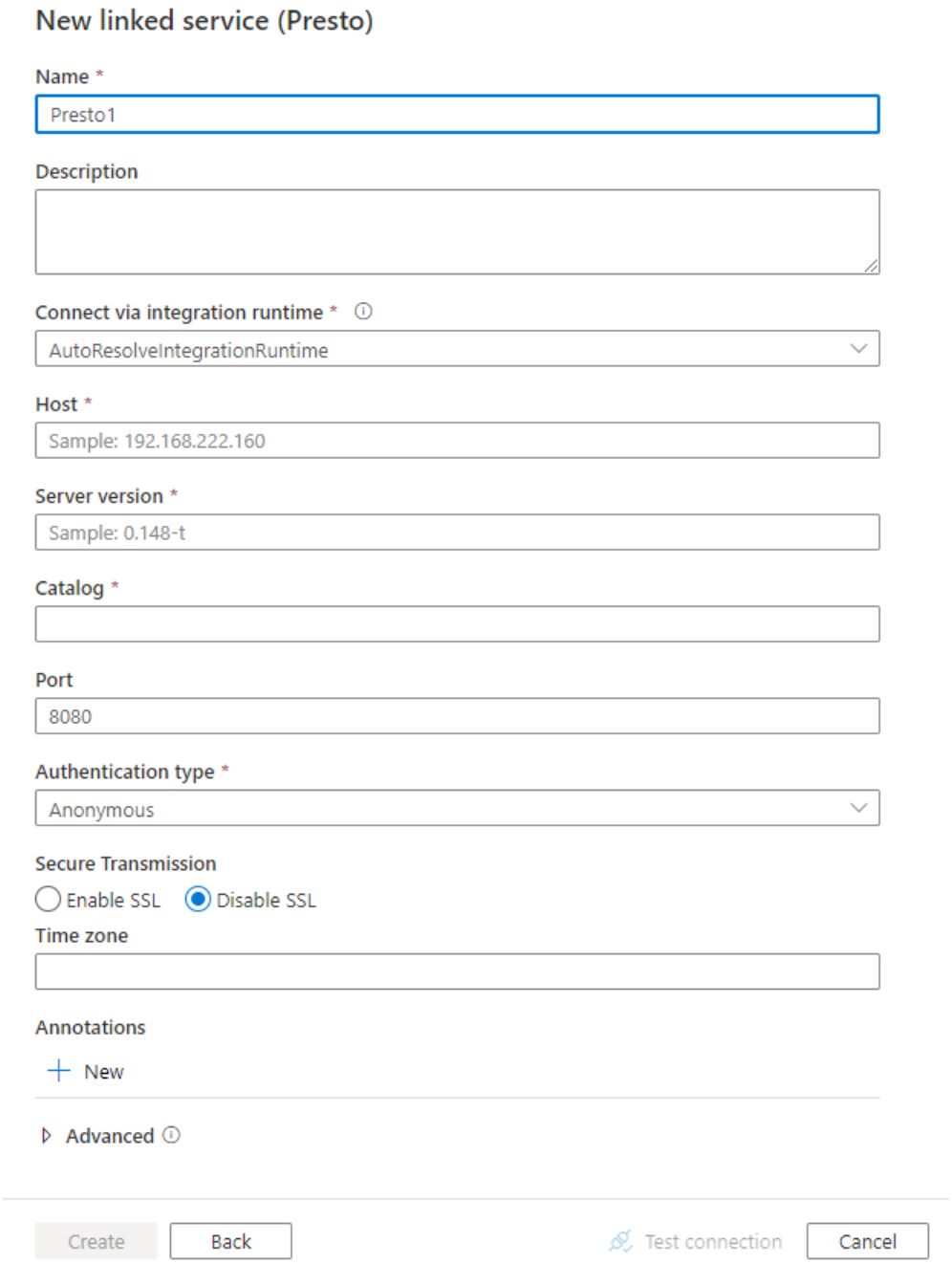
   - Azure Data Factory
   - Azure Synapse



2. Search for Presto and select the Presto connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Presto connector.

Linked service properties

The following properties are supported for Presto linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Presto**	Yes
host	The IP address or host name of the Presto server. (e.g. 192.168.222.160)	Yes

PROPERTY	DESCRIPTION	REQUIRED
serverVersion	The version of the Presto server. (e.g. 0.148-t)	Yes
catalog	The catalog context for all request against the server.	Yes
port	The TCP port that the Presto server uses to listen for client connections. The default value is 8080.	No
authenticationType	The authentication mechanism used to connect to the Presto server. Allowed values are: **Anonymous**, **LDAP**	Yes
username	The user name used to connect to the Presto server.	No
password	The password corresponding to the user name. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
enableSsl	Specifies whether the connections to the server are encrypted using TLS. The default value is false.	No
trustedCertPath	The full path of the .pem file containing trusted CA certificates for verifying the server when connecting over TLS. This property can only be set when using TLS on self-hosted IR. The default value is the cacerts.pem file installed with the IR.	No
useSystemTrustStore	Specifies whether to use a CA certificate from the system trust store or from a specified PEM file. The default value is false.	No
allowHostNameCNMismatch	Specifies whether to require a CA-issued TLS/SSL certificate name to match the host name of the server when connecting over TLS. The default value is false.	No
allowSelfSignedServerCert	Specifies whether to allow self-signed certificates from the server. The default value is false.	No
timeZoneID	The local time zone used by the connection. Valid values for this option are specified in the IANA Time Zone Database. The default value is the system time zone.	No

Example:

```
{
    "name": "PrestoLinkedService",
    "properties": {
        "type": "Presto",
        "typeProperties": {
            "host" : "<host>",
            "serverVersion" : "0.148-t",
            "catalog" : "<catalog>",
            "port" : "<port>",
            "authenticationType" : "LDAP",
            "username" : "<username>",
            "password": {
                 "type": "SecureString",
                 "value": "<password>"
            },
            "timeZoneID" : "Europe/Berlin"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Presto dataset.

To copy data from Presto, set the type property of the dataset to **PrestoObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **PrestoObject**	Yes
schema	Name of the schema.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table with schema. This property is supported for backward compatibility. Use `schema` and `table` for new workload.	No (if "query" in activity source is specified)

Example

```
{
    "name": "PrestoDataset",
    "properties": {
        "type": "PrestoObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Presto linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Presto source.

Presto as source

To copy data from Presto, set the source type in the copy activity to **PrestoSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **PrestoSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromPresto",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Presto input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "PrestoSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Transform data in Quickbase (Preview) using Azure Data Factory or Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Data Flow to transform data in Quickbase (Preview). To learn more, read the introductory article for Azure Data Factory or Azure Synapse Analytics.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Quickbase connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Mapping data flow (source/-)	①

① Azure integration runtime ② Self-hosted integration runtime

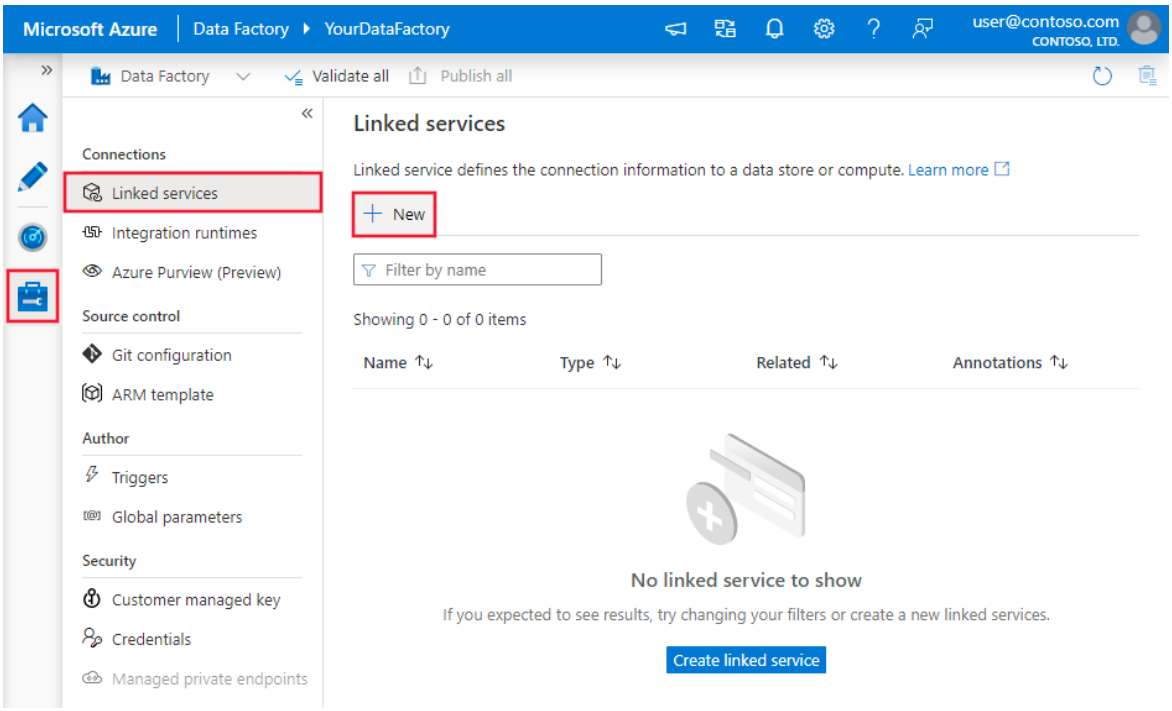
For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Create a Quickbase linked service using UI

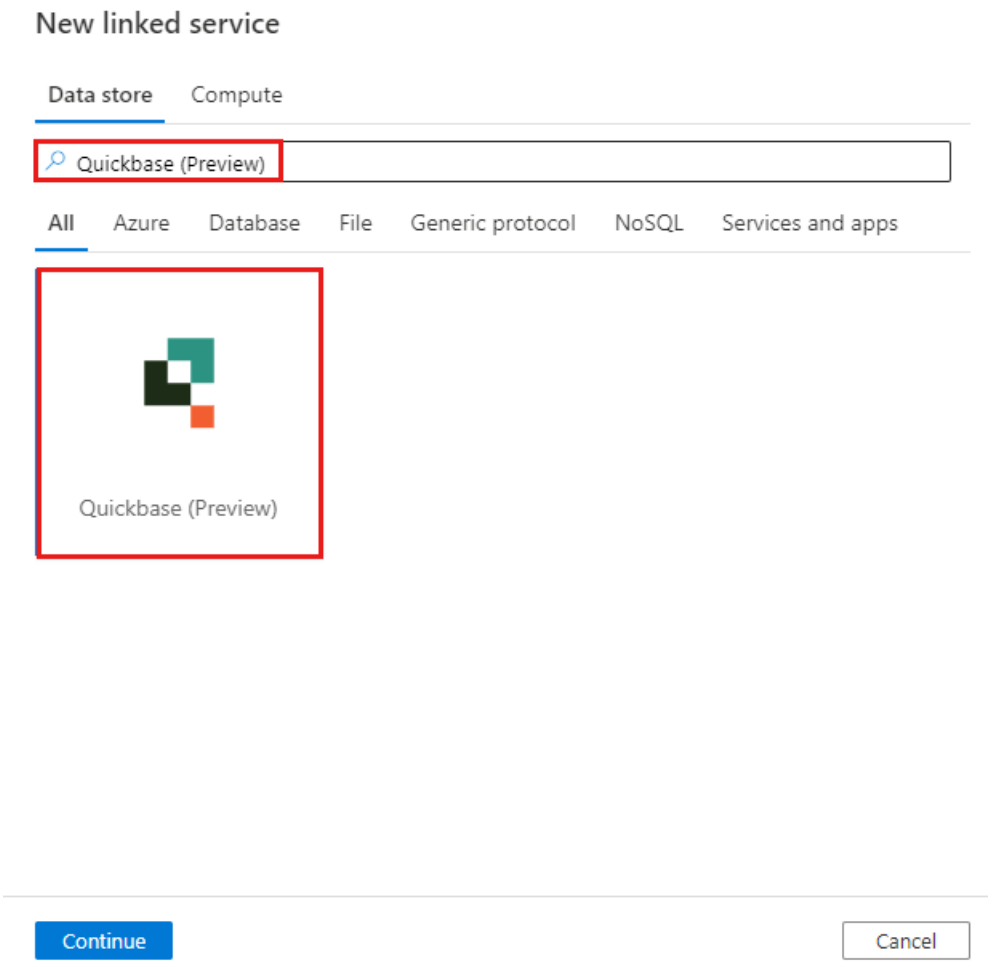
Use the following steps to create a Quickbase linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then select New:

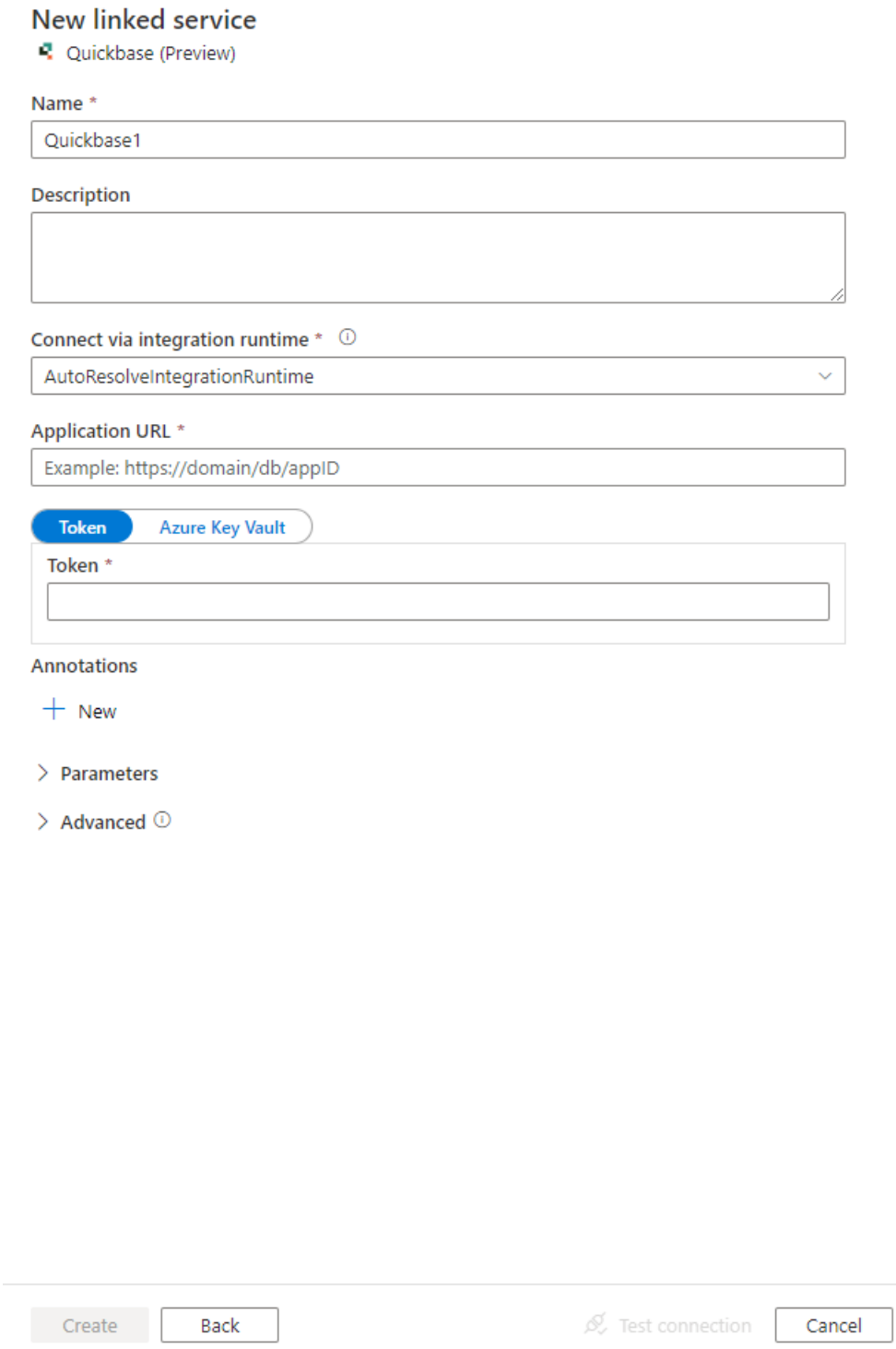
   - Azure Data Factory
   - Azure Synapse



2. Search for Quickbase (Preview) and select the Quickbase (Preview) connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide information about properties that are used to define Data Factory and Synapse pipeline entities specific to Quickbase.

Linked service properties

The following properties are supported for the Quickbase linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **Quickbase**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
url	The application URL of the Quickbase service.	Yes
userToken	Specify a user token for the Quickbase. Mark this field as **SecureString** to store it securely. Or, you can reference a secret stored in Azure Key Vault.	Yes

Example:

```
{
    "name": "QuickbaseLinkedService",
    "properties": {
        "type": "Quickbase",
        "typeProperties": {
            "url": "<application url>",
            "userToken": {
                "type": "SecureString",
                "value": "<user token>"
            }
        }
    }
}
```

Mapping data flow properties

When transforming data in mapping data flow, you can read tables from Quickbase. For more information, see the source transformation in mapping data flows. You can only use an inline dataset as source type.

Source transformation

The below table lists the properties supported by Quickbase source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Table	Data flow will fetch all the data from the table specified in the source options.	Yes when use inline mode	-	table
Report	Data flow will fetch the specified report for the table specified in the source options.	No	-	report

Quickbase source script examples

```
source(allowSchemaDrift: true,
 validateSchema: false,
 store: 'quickbase',
 format: 'rest',
 table: 'Table',
 report: 'Report') ~> Quickbasesource
```

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from QuickBooks Online using Azure Data Factory or Synapse Analytics (Preview)

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from QuickBooks Online. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This QuickBooks connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

This connector supports QuickBooks OAuth 2.0 authentication.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

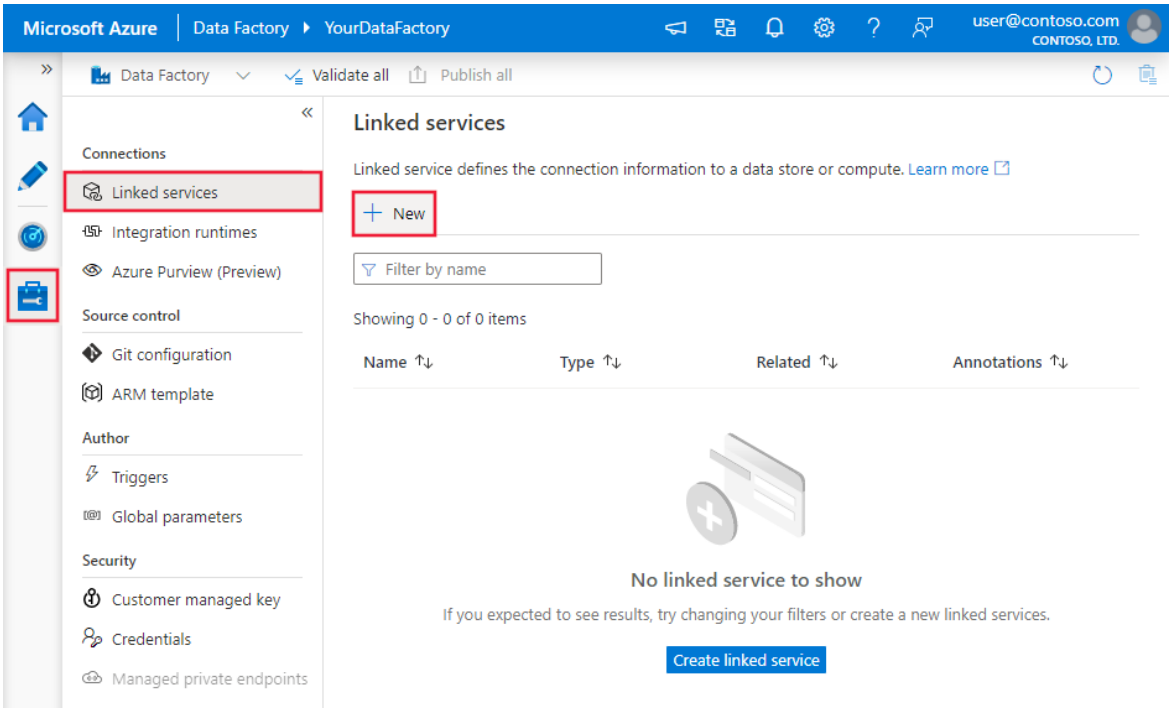
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to QuickBooks using UI

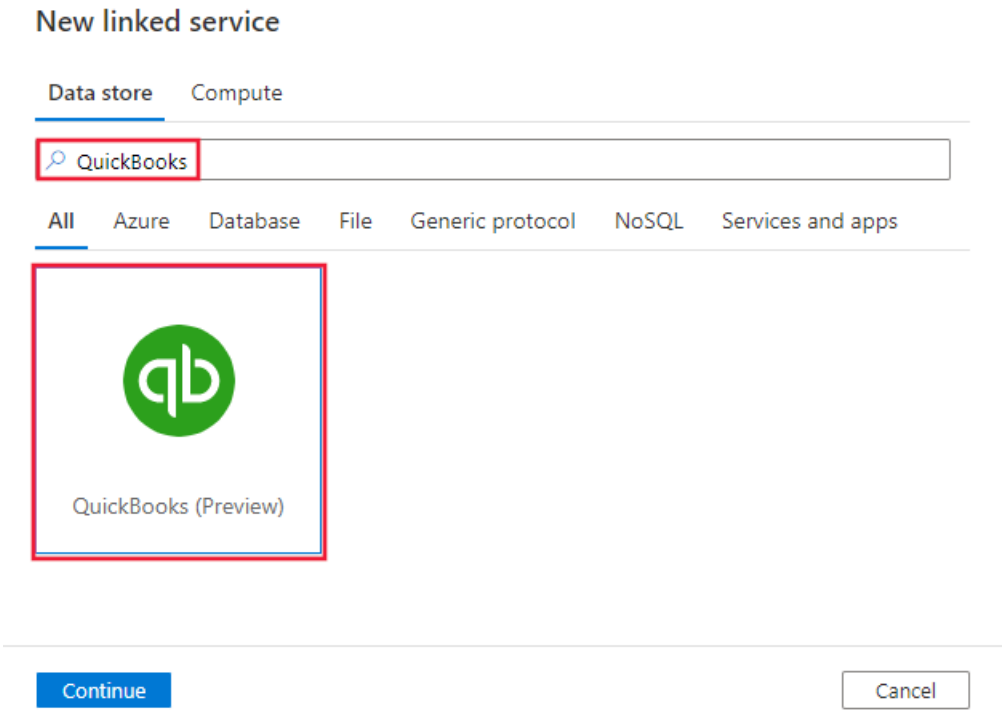
Use the following steps to create a linked service to QuickBooks in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

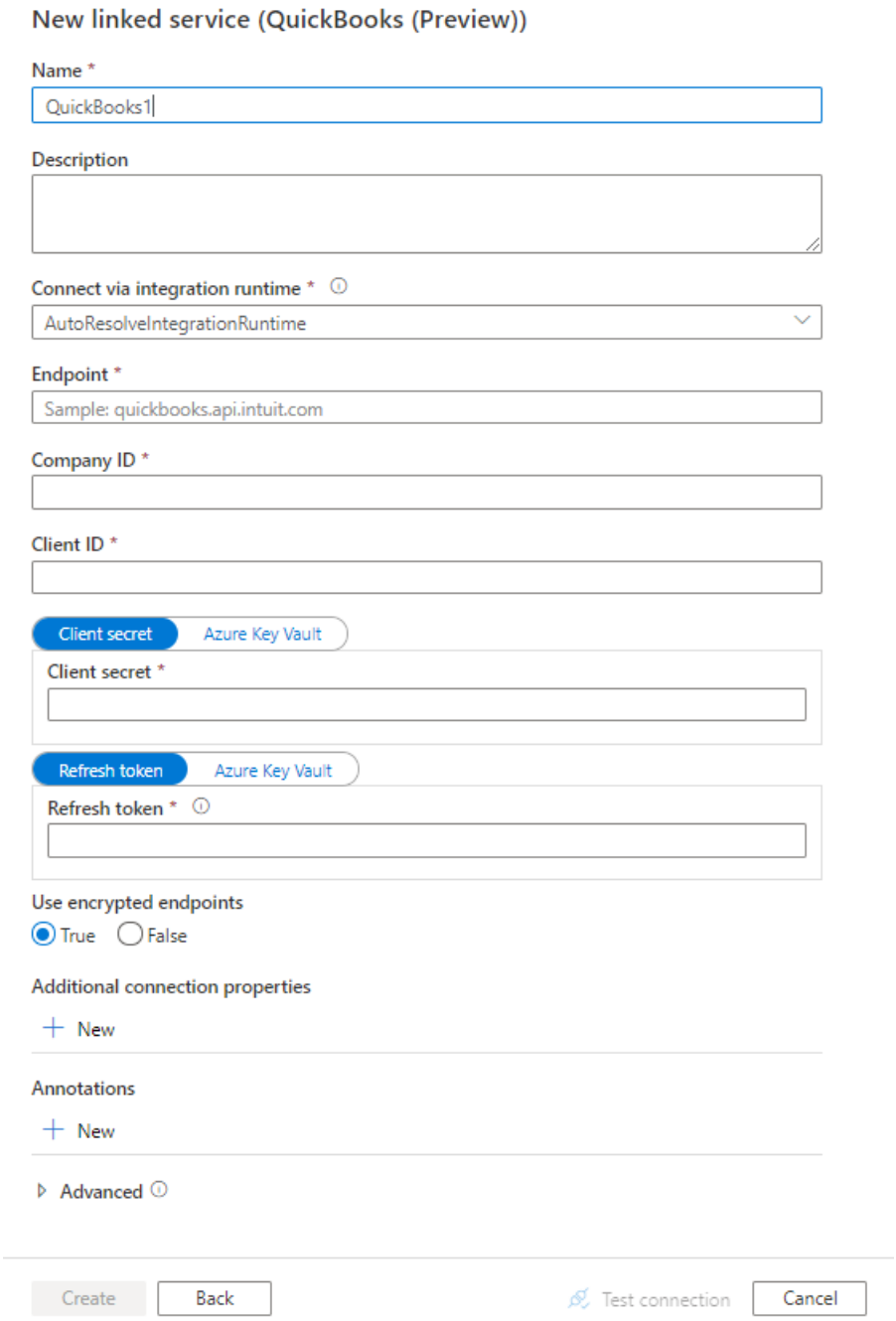
- Azure Data Factory
- Azure Synapse



2. Search for QuickBooks and select the QuickBooks connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to QuickBooks connector.

Linked service properties

The following properties are supported for QuickBooks linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **QuickBooks**	Yes
connectionProperties	A group of properties that defines how to connect to QuickBooks.	Yes
Under `connectionProperties`:		

PROPERTY	DESCRIPTION	REQUIRED
endpoint	The endpoint of the QuickBooks Online server. (that is, quickbooks.api.intuit.com)	Yes
companyId	The company ID of the QuickBooks company to authorize. For info about how to find the company ID, see How do I find my Company ID.	Yes
consumerKey	The client ID of your QuickBooks Online application for OAuth 2.0 authentication. Learn more from here.	Yes
consumerSecret	The client secret of your QuickBooks Online application for OAuth 2.0 authentication. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
refreshToken	The OAuth 2.0 refresh token associated with the QuickBooks application. Learn more from here. Note refresh token will be expired after 180 days. Customer need to regularly update the refresh token. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No

Example:

```
{
    "name": "QuickBooksLinkedService",
    "properties": {
        "type": "QuickBooks",
        "typeProperties": {
            "connectionProperties":{
                "endpoint":"quickbooks.api.intuit.com",
                "companyId":"<company id>",
                "consumerKey":"<consumer key>",
                "consumerSecret":{
                     "type": "SecureString",
                     "value": "<clientSecret>"
             },
                "refreshToken":{
                     "type": "SecureString",
                     "value": "<refresh token>"
             },
                "useEncryptedEndpoints":true
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by QuickBooks dataset.

To copy data from QuickBooks Online, set the type property of the dataset to **QuickBooksObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **QuickBooksObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "QuickBooksDataset",
    "properties": {
        "type": "QuickBooksObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<QuickBooks linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by QuickBooks source.

QuickBooks as source

To copy data from QuickBooks Online, set the source type in the copy activity to **QuickBooksSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **QuickBooksSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM "Bill" WHERE Id = '123'"` .	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromQuickBooks",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<QuickBooks input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "QuickBooksSource",
                "query": "SELECT * FROM \"Bill\" WHERE Id = '123' "
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Copy data from Quickbooks Desktop

The Copy Activity in the service cannot copy data directly from Quickbooks Desktop. To copy data from Quickbooks Desktop, export your Quickbooks data to a comma-separated-values (CSV) file and then upload the file to Azure Blob Storage. From there, you can use the service to copy the data to the sink of your choice.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy and transform data from and to a REST endpoint by using Azure Data Factory

8/19/2022 • 21 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory to copy data from and to a REST endpoint. The article builds on Copy Activity in Azure Data Factory, which presents a general overview of Copy Activity.

The difference among this REST connector, HTTP connector, and the Web table connector are:

- **REST connector** specifically supports copying data from RESTful APIs.
- **HTTP connector** is generic to retrieve data from any HTTP endpoint, for example, to download file. Before this REST connector you may happen to use HTTP connector to copy data from RESTful APIs, which is supported but less functional comparing to REST connector.
- **Web table connector** extracts table content from an HTML webpage.

Supported capabilities

This REST connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Mapping data flow (source/sink)	①

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see Supported data stores.

Specifically, this generic REST connector supports:

- Copying data from a REST endpoint by using the **GET** or **POST** methods and copying data to a REST endpoint by using the **POST**, **PUT** or **PATCH** methods.
- Copying data by using one of the following authentications: **Anonymous**, **Basic**, **AAD service principal**, and **user-assigned managed identity**.
- **Pagination** in the REST APIs.
- For REST as source, copying the REST JSON response as-is or parse it by using schema mapping. Only response payload in **JSON** is supported.

> **TIP**
> To test a request for data retrieval before you configure the REST connector in Data Factory, learn about the API specification for header and body requirements. You can use tools like Postman or a web browser to validate.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

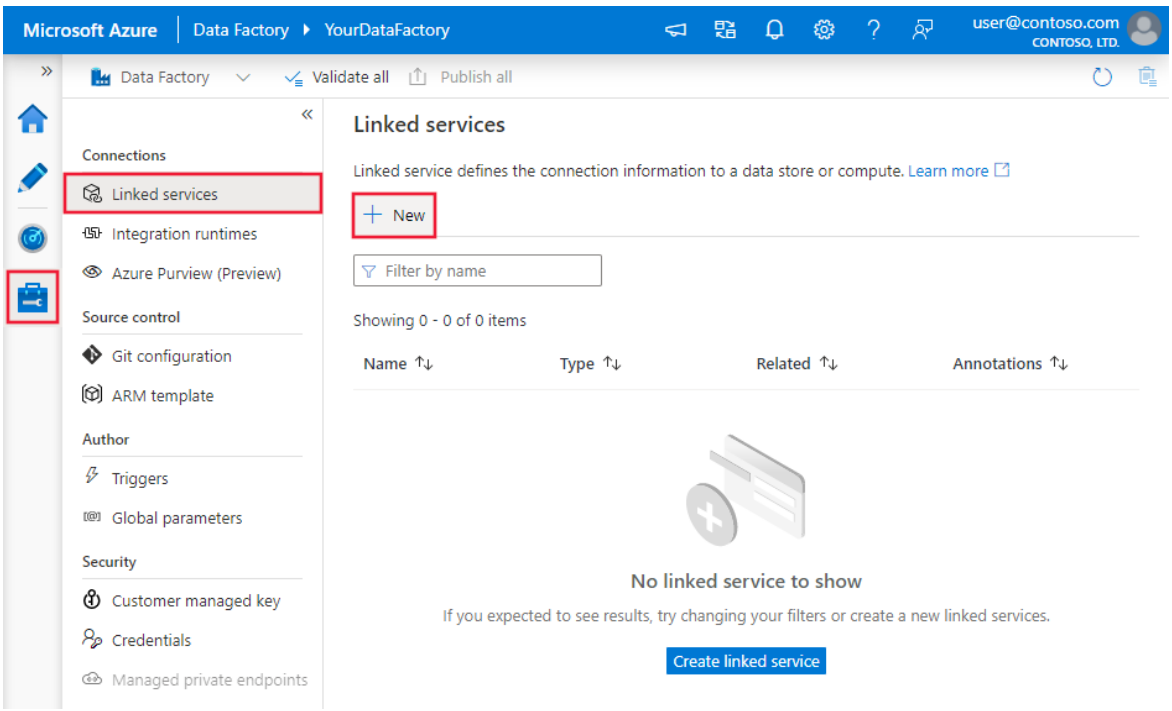
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a REST linked service using UI

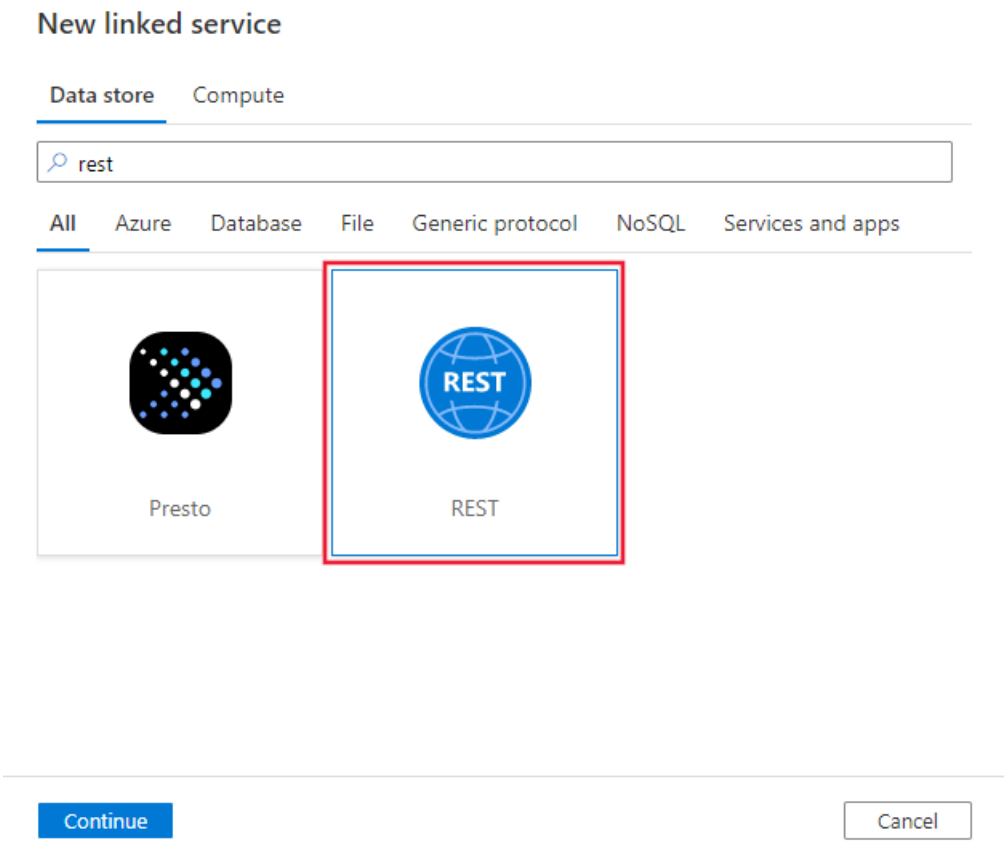
Use the following steps to create a REST linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

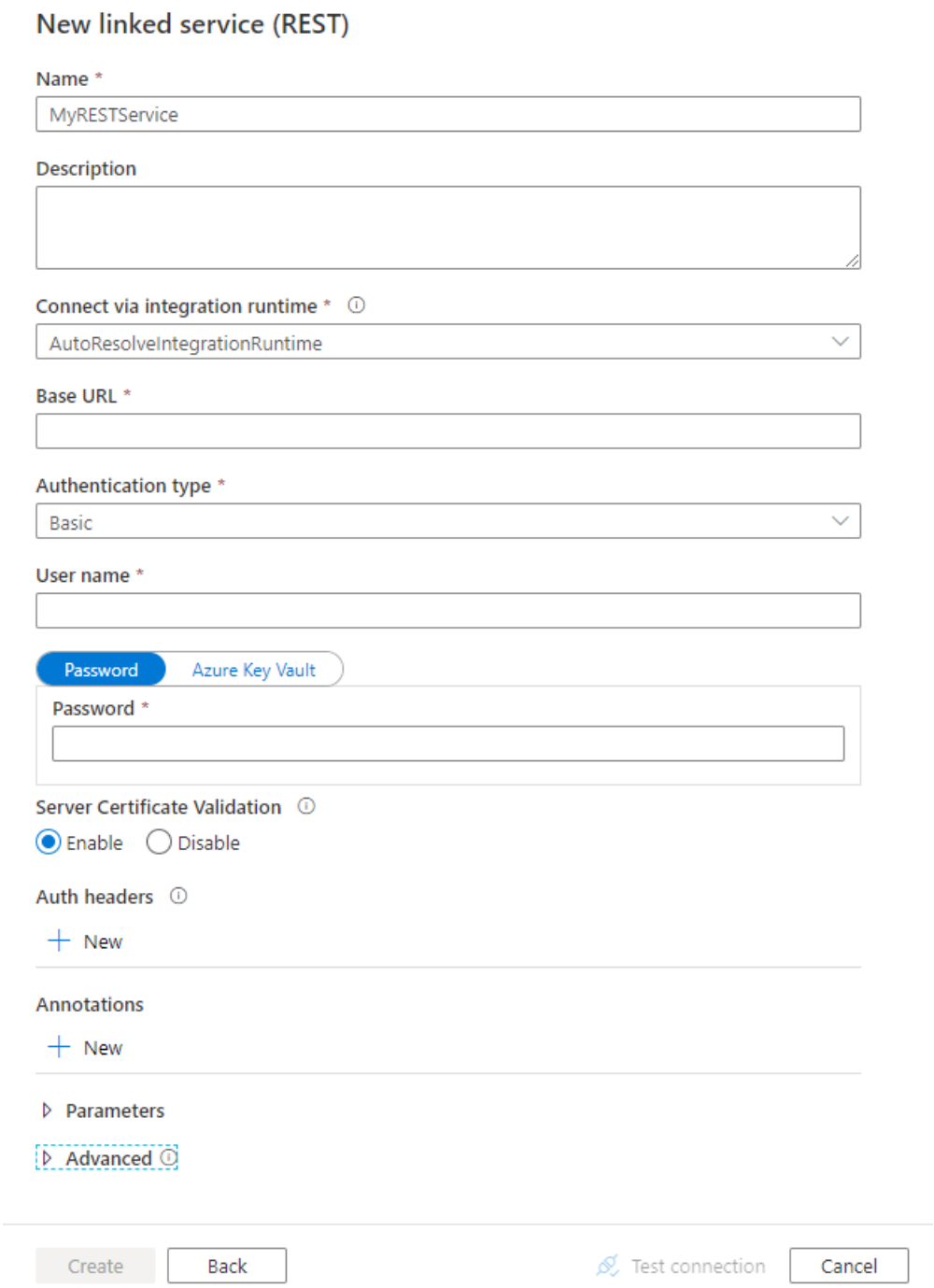
   - Azure Data Factory
   - Azure Synapse



2. Search for REST and select the REST connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties you can use to define Data Factory entities that are specific to the REST connector.

Linked service properties

The following properties are supported for the REST linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property must be set to **RestService**.	Yes
url	The base URL of the REST service.	Yes

PROPERTY	DESCRIPTION	REQUIRED
enableServerCertificateValidation	Whether to validate server-side TLS/SSL certificate when connecting to the endpoint.	No (the default is **true**)
authenticationType	Type of authentication used to connect to the REST service. Allowed values are **Anonymous**, **Basic**, **AadServicePrincipal**, **OAuth2ClientCredential**, and **ManagedServiceIdentity**. You can additionally configure authentication headers in `authHeader` property. Refer to corresponding sections below on more properties and examples respectively.	Yes
authHeaders	Additional HTTP request headers for authentication. For example, to use API key authentication, you can select authentication type as "Anonymous" and specify API key in the header.	No
connectVia	The Integration Runtime to use to connect to the data store. Learn more from Prerequisites section. If not specified, this property uses the default Azure Integration Runtime.	No

For different authentication types, see the corresponding sections for details.

- Basic authentication
- AAD service principal authentication
- OAuth2 Client Credential authentication
- User-assigned managed identity authentication
- Anonymous authentication

Use basic authentication

Set the **authenticationType** property to **Basic**. In addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
userName	The user name to use to access the REST endpoint.	Yes
password	The password for the user (the **userName** value). Mark this field as a **SecureString** type to store it securely in Data Factory. You can also reference a secret stored in Azure Key Vault.	Yes

Example

```
{
    "name": "RESTLinkedService",
    "properties": {
        "type": "RestService",
        "typeProperties": {
            "authenticationType": "Basic",
            "url" : "<REST endpoint>",
            "userName": "<user name>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Use AAD service principal authentication

Set the **authenticationType** property to **AadServicePrincipal**. In addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalId	Specify the Azure Active Directory application's client ID.	Yes
servicePrincipalKey	Specify the Azure Active Directory application's key. Mark this field as a **SecureString** to store it securely in Data Factory, or reference a secret stored in Azure Key Vault.	Yes
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. Retrieve it by hovering the mouse in the top-right corner of the Azure portal.	Yes
aadResourceId	Specify the AAD resource you are requesting for authorization, for example, `https://management.core.windows.net`.	Yes
azureCloudType	For service principal authentication, specify the type of Azure cloud environment to which your AAD application is registered. Allowed values are **AzurePublic**, **AzureChina**, **AzureUsGovernment**, and **AzureGermany**. By default, the data factory's cloud environment is used.	No

Example

```
{
    "name": "RESTLinkedService",
    "properties": {
        "type": "RestService",
        "typeProperties": {
            "url": "<REST endpoint e.g. https://www.example.com/>",
            "authenticationType": "AadServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "value": "<service principal key>",
                "type": "SecureString"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>",
            "aadResourceId": "<AAD resource URL e.g. https://management.core.windows.net>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Use OAuth2 Client Credential authentication

Set the **authenticationType** property to **OAuth2ClientCredential**. In addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
tokenEndpoint	The token endpoint of the authorization server to acquire the access token.	Yes
clientId	The client ID associated with your application.	Yes
clientSecret	The client secret associated with your application. Mark this field as a **SecureString** type to store it securely in Data Factory. You can also reference a secret stored in Azure Key Vault.	Yes
scope	The scope of the access required. It describes what kind of access will be requested.	No
resource	The target service or resource to which the access will be requested.	No

Example

```
{
    "name": "RESTLinkedService",
    "properties": {
        "type": "RestService",
        "typeProperties": {
            "url": "<REST endpoint e.g. https://www.example.com/>",
            "enableServerCertificateValidation": true,
            "authenticationType": "OAuth2ClientCredential",
            "clientId": "<client ID>",
            "clientSecret": {
                "type": "SecureString",
                "value": "<client secret>"
            },
            "tokenEndpoint": "<token endpoint>",
            "scope": "<scope>",
            "resource": "<resource>"
        }
    }
}
```

Use user-assigned managed identity authentication

Set the **authenticationType** property to **ManagedServiceIdentity**. In addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
aadResourceId	Specify the AAD resource you are requesting for authorization, for example, `https://management.core.windows.net`.	Yes
credentials	Specify the user-assigned managed identity as the credential object.	Yes

Example

```
{
    "name": "RESTLinkedService",
    "properties": {
        "type": "RestService",
        "typeProperties": {
            "url": "<REST endpoint e.g. https://www.example.com/>",
            "authenticationType": "ManagedServiceIdentity",
            "aadResourceId": "<AAD resource URL e.g. https://management.core.windows.net>",
            "credential": {
                "referenceName": "credential1",
                "type": "CredentialReference"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Using authentication headers

In addition, you can configure request headers for authentication along with the built-in authentication types.

Example: Using API key authentication

```
{
    "name": "RESTLinkedService",
    "properties": {
        "type": "RestService",
        "typeProperties": {
            "url": "<REST endpoint>",
            "authenticationType": "Anonymous",
            "authHeader": {
                "x-api-key": {
                    "type": "SecureString",
                    "value": "<API key>"
                }
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

This section provides a list of properties that the REST dataset supports.

For a full list of sections and properties that are available for defining datasets, see Datasets and linked services.

To copy data from REST, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the dataset must be set to **RestResource**.	Yes
relativeUrl	A relative URL to the resource that contains the data. When this property isn't specified, only the URL that's specified in the linked service definition is used. The HTTP connector copies data from the combined URL: `[URL specified in linked service]/[relative URL specified in dataset]` .	No

If you were setting `requestMethod`, `additionalHeaders`, `requestBody` and `paginationRules` in dataset, it is still supported as-is, while you are suggested to use the new model in activity going forward.

Example:

```
{
    "name": "RESTDataset",
    "properties": {
        "type": "RestResource",
        "typeProperties": {
            "relativeUrl": "<relative url>"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<REST linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy Activity properties

This section provides a list of properties supported by the REST source and sink.

For a full list of sections and properties that are available for defining activities, see Pipelines.

REST as source

The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the copy activity source must be set to **RestSource**.	Yes
requestMethod	The HTTP method. Allowed values are **GET** (default) and **POST**.	No
additionalHeaders	Additional HTTP request headers.	No
requestBody	The body for the HTTP request.	No
paginationRules	The pagination rules to compose next page requests. Refer to pagination support section on details.	No
httpRequestTimeout	The timeout (the **TimeSpan** value) for the HTTP request to get a response. This value is the timeout to get a response, not the timeout to read response data. The default value is **00:01:40**.	No
requestInterval	The time to wait before sending the request for next page. The default value is **00:00:01**	No

NOTE

REST connector ignores any "Accept" header specified in `additionalHeaders`. As REST connector only support response in JSON, it will auto generate a header of `Accept: application/json`.

Example 1: Using the Get method with pagination

```
"activities":[
    {
        "name": "CopyFromREST",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<REST input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "RestSource",
                "additionalHeaders": {
                    "x-user-defined": "helloworld"
                },
                "paginationRules": {
                    "AbsoluteUrl": "$.paging.next"
                },
                "httpRequestTimeout": "00:01:00"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Example 2: Using the Post method

```
"activities":[
    {
        "name": "CopyFromREST",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<REST input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "RestSource",
                "requestMethod": "Post",
                "requestBody": "<body for POST REST request>",
                "httpRequestTimeout": "00:01:00"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

REST as sink

The following properties are supported in the copy activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the copy activity sink must be set to **RestSink**.	Yes
requestMethod	The HTTP method. Allowed values are **POST** (default), **PUT**, and **PATCH**.	No
additionalHeaders	Additional HTTP request headers.	No
httpRequestTimeout	The timeout (the **TimeSpan** value) for the HTTP request to get a response. This value is the timeout to get a response, not the timeout to write the data. The default value is **00:01:40**.	No
requestInterval	The interval time between different requests in millisecond. Request interval value should be a number between [10, 60000].	No
httpCompressionType	HTTP compression type to use while sending data with Optimal Compression Level. Allowed values are **none** and **gzip**.	No

PROPERTY	DESCRIPTION	REQUIRED
writeBatchSize	Number of records to write to the REST sink per batch. The default value is 10000.	No

REST connector as sink works with the REST APIs that accept JSON. The data will be sent in JSON with the following pattern. As needed, you can use the copy activity schema mapping to reshape the source data to conform to the expected payload by the REST API.

```
[
    { <data object> },
    { <data object> },
    ...
]
```

Example:

```
"activities":[
    {
        "name": "CopyToREST",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<REST output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "RestSink",
                "requestMethod": "POST",
                "httpRequestTimeout": "00:01:40",
                "requestInterval": 10,
                "writeBatchSize": 10000,
                "httpCompressionType": "none",
            },
        }
    }
]
```

Mapping data flow properties

REST is supported in data flows for both integration datasets and inline datasets.

Source transformation

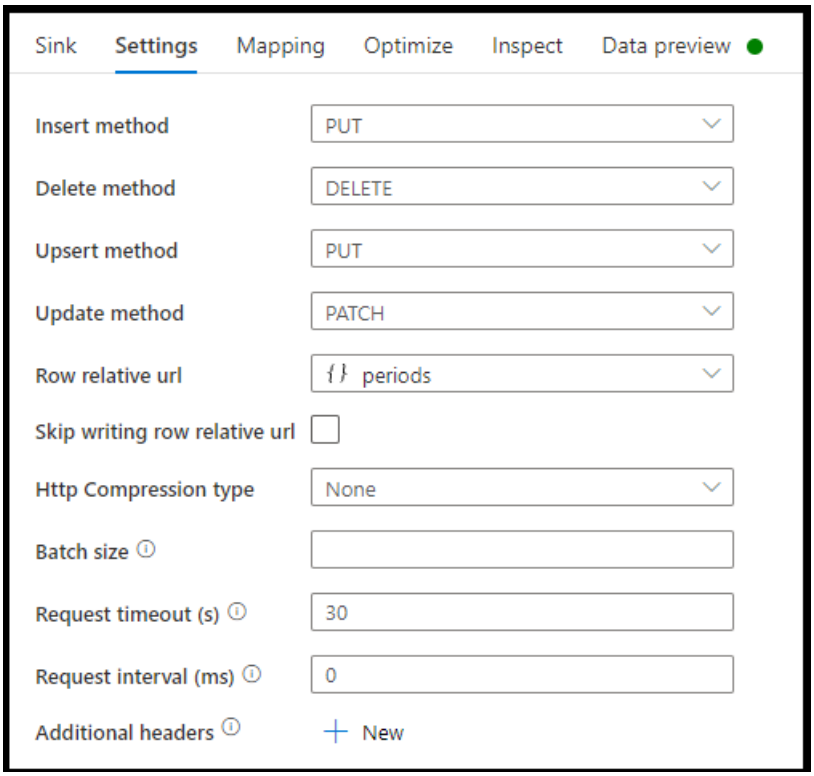
PROPERTY	DESCRIPTION	REQUIRED
requestMethod	The HTTP method. Allowed values are **GET** and **POST**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
relativeUrl	A relative URL to the resource that contains the data. When this property isn't specified, only the URL that's specified in the linked service definition is used. The HTTP connector copies data from the combined URL: `[URL specified in linked service]/[relative URL specified in dataset]` .	No
additionalHeaders	Additional HTTP request headers.	No
httpRequestTimeout	The timeout (the **TimeSpan** value) for the HTTP request to get a response. This value is the timeout to get a response, not the timeout to write the data. The default value is **00:01:40**.	No
requestInterval	The interval time between different requests in millisecond. Request interval value should be a number between [10, 60000].	No
QueryParameters.*request_query_parameter* OR QueryParameters['request_query_parameter']	"request_query_parameter" is user-defined, which references one query parameter name in the next HTTP request URL.	No

Sink transformation

PROPERTY	DESCRIPTION	REQUIRED
additionalHeaders	Additional HTTP request headers.	No
httpRequestTimeout	The timeout (the **TimeSpan** value) for the HTTP request to get a response. This value is the timeout to get a response, not the timeout to write the data. The default value is **00:01:40**.	No
requestInterval	The interval time between different requests in millisecond. Request interval value should be a number between [10, 60000].	No
httpCompressionType	HTTP compression type to use while sending data with Optimal Compression Level. Allowed values are **none** and **gzip**.	No
writeBatchSize	Number of records to write to the REST sink per batch. The default value is 10000.	No

You can set the delete, insert, update, and upsert methods as well as the relative row data to send to the REST sink for CRUD operations.



Sample data flow script

Notice the use of an alter row transformation prior to the sink to instruct ADF what type of action to take with your REST sink. I.e. insert, update, upsert, delete.

```
AlterRow1 sink(allowSchemaDrift: true,
 validateSchema: false,
 deletable:true,
 insertable:true,
 updateable:true,
 upsertable:true,
 rowRelativeUrl: 'periods',
 insertHttpMethod: 'PUT',
 deleteHttpMethod: 'DELETE',
 upsertHttpMethod: 'PUT',
 updateHttpMethod: 'PATCH',
 timeout: 30,
 requestFormat: ['type' -> 'json'],
 skipDuplicateMapInputs: true,
 skipDuplicateMapOutputs: true) ~> sink1
```

Pagination support

When copying data from REST APIs, normally, the REST API limits its response payload size of a single request under a reasonable number; while to return large amount of data, it splits the result into multiple pages and requires callers to send consecutive requests to get next page of the result. Usually, the request for one page is dynamic and composed by the information returned from the response of previous page.

This generic REST connector supports the following pagination patterns:

- Next request's absolute or relative URL = property value in current response body
- Next request's absolute or relative URL = header value in current response headers
- Next request's query parameter = property value in current response body
- Next request's query parameter = header value in current response headers
- Next request's header = property value in current response body

- Next request's header = header value in current response headers

Pagination rules are defined as a dictionary in dataset, which contains one or more case-sensitive key-value pairs. The configuration will be used to generate the request starting from the second page. The connector will stop iterating when it gets HTTP status code 204 (No Content), or any of the JSONPath expressions in "paginationRules" returns null.

Supported keys in pagination rules:

KEY	DESCRIPTION
AbsoluteUrl	Indicates the URL to issue the next request. It can be **either absolute URL or relative URL**.
QueryParameters.*request_query_parameter* OR QueryParameters['request_query_parameter']	"request_query_parameter" is user-defined, which references one query parameter name in the next HTTP request URL.
Headers.*request_header* OR Headers['request_header']	"request_header" is user-defined, which references one header name in the next HTTP request.
EndCondition:*end_condition*	"end_condition" is user-defined, which indicates the condition that will end the pagination loop in the next HTTP request.
MaxRequestNumber	Indicates the maximum pagination request number. Leave it as empty means no limit.
SupportRFC5988	By default, this is set to true if no pagination rule is defined. You can disable this rule by setting `supportRFC5988` to false or remove this property from script.

Supported values in pagination rules:

VALUE	DESCRIPTION
Headers.*response_header* OR Headers['response_header']	"response_header" is user-defined, which references one header name in the current HTTP response, the value of which will be used to issue next request.
A JSONPath expression starting with "$" (representing the root of the response body)	The response body should contain only one JSON object. The JSONPath expression should return a single primitive value, which will be used to issue next request.

NOTE

The pagination rules in mapping data flows is different from it in copy activity in the following aspects:

1. Range is not supported in mapping data flows.
2. `['']` is not supported in mapping data flows. Instead, use `{}` to escape special character. For example, `body.{@odata.nextLink}`, whose JSON node `@odata.nextLink` contains special character `.`.
3. The end condition is supported in mapping data flows, but the condition syntax is different from it in copy activity. `body` is used to indicate the response body instead of `$`. `header` is used to indicate the response header instead of `headers`. Here are two examples showing this difference:
   - Example 1:
     Copy activity: **"EndCondition:$.data": "Empty"**
     Mapping data flows: **"EndCondition:body.data": "Empty"**
   - Example 2:
     Copy activity: **"EndCondition:headers.complete": "Exist"**
     Mapping data flows: **"EndCondition:header.complete": "Exist"**

Pagination rules examples

This section provides a list of examples for pagination rules settings.

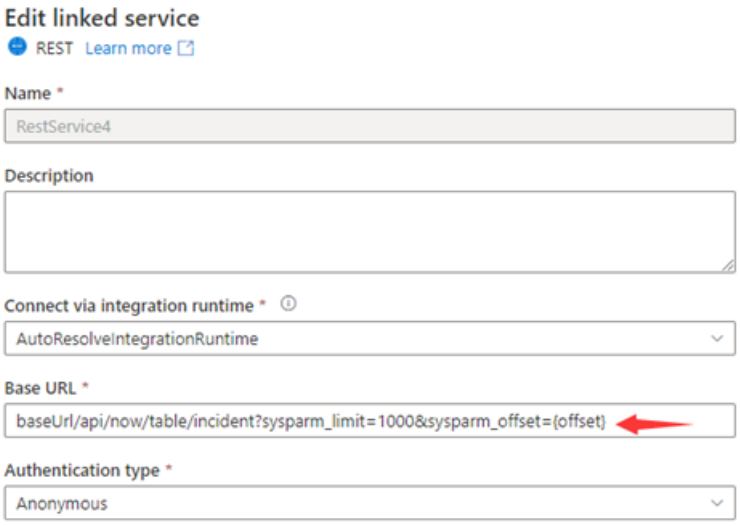
Example 1: Variables in QueryParameters

This example provides the configuration steps to send multiple requests whose variables are in QueryParameters.

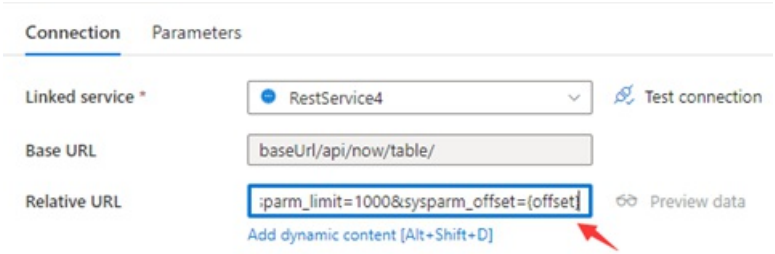
Multiple requests:

```
baseUrl/api/now/table/incident?sysparm_limit=1000&sysparm_offset=0,
baseUrl/api/now/table/incident?sysparm_limit=1000&sysparm_offset=1000,
......
baseUrl/api/now/table/incident?sysparm_limit=1000&sysparm_offset=10000
```

Step 1: Input `sysparm_offset={offset}` either in **Base URL** or **Relative URL** as shown in the following screenshots:



or



Step 2: Set **Pagination rules** as either option 1 or option 2:

- Option1: **"QueryParameters.{offset}" : "RANGE:0:10000:1000"**
- Option2: **"AbsoluteUrl.{offset}" : "RANGE:0:10000:1000"**

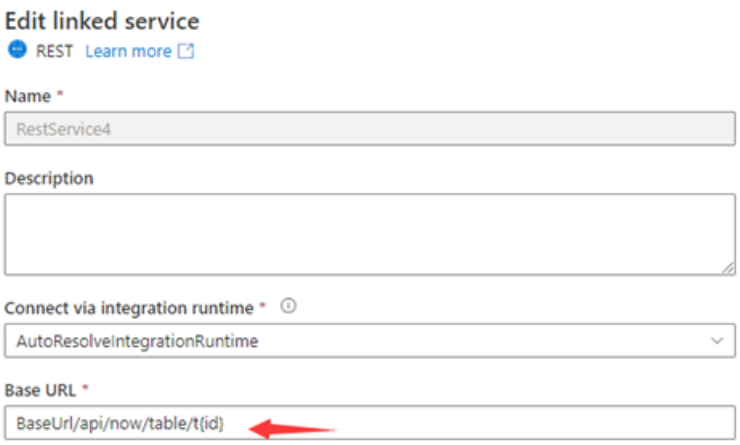
Example 2: Variables in AbsoluteUrl

This example provides the configuration steps to send multiple requests whose variables are in AbsoluteUrl.

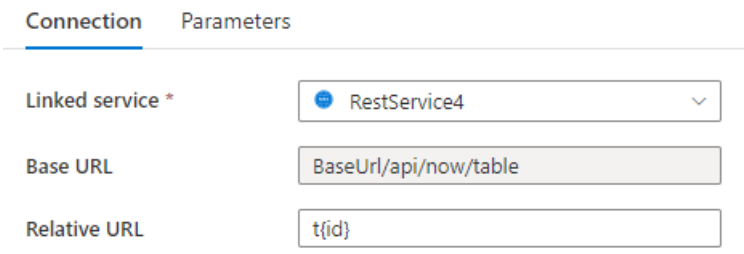
Multiple requests:

```
BaseUrl/api/now/table/t1
BaseUrl/api/now/table/t2
......
BaseUrl/api/now/table/t100
```

Step 1: Input `{id}` either in **Base URL** in the linked service configuration page or **Relative URL** in the dataset connection pane.



or



Step 2: Set **Pagination rules** as **"AbsoluteUrl.{id}" :"RANGE:1:100:1"**.

Example 3: Variables in Headers

This example provides the configuration steps to send multiple requests whose variables are in Headers.

Multiple requests:

RequestUrl: *https://example/table*

Request 1: `Header(id->0)`

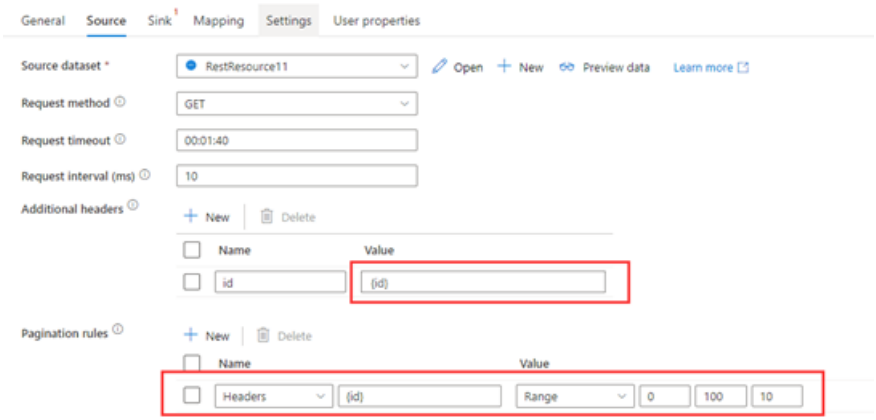
Request 2: `Header(id->10)`

.....

Request 100: `Header(id->100)`

Step 1: Input `{id}` in **Additional headers**.

Step 2: Set **Pagination rules** as **"Headers.{id}" : "RARNGE:0:100:10"**.



Example 4: **Variables are in AbsoluteUrl/QueryParameters/Headers, the end variable is not pre-defined and the end condition is based on the response**

This example provides configuration steps to send multiple requests whose variables are in AbsoluteUrl/QueryParameters/Headers but the end variable is not defined. For different responses, different end condition rule settings are shown in Example 4.1-4.6.

Multiple requests:

```
Request 1: baseUrl/api/now/table/incident?sysparm_limit=1000&sysparm_offset=0,
Request 2: baseUrl/api/now/table/incident?sysparm_limit=1000&sysparm_offset=1000,
Request 3: baseUrl/api/now/table/incident?sysparm_limit=1000&sysparm_offset=2000,
......
```

Two responses encountered in this example:

Response 1 :

```
{
    Data: [
        {key1: val1, key2: val2
        },
        {key1: val3, key2: val4
        }
    ]
}
```

Response 2 :

```
{
    Data: [
        {key1: val5, key2: val6
        },
        {key1: val7, key2: val8
        }
    ]
}
```

Step 1: Set the range of **Pagination rules** as Example 1 and leave the end of range empty as **"AbsoluteUrl.**

{offset}": "RANGE:0::1000".

Step 2: Set different end condition rules according to different last responses. See below examples:

- **Example 4.1: The pagination ends when the value of the specific node in response is empty**

  The REST API returns the last response in the following structure:

  ```
  {
      Data: []
  }
  ```

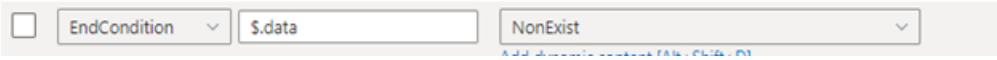
  Set the end condition rule as **"EndCondition:$.data": "Empty"** to end the pagination when the value of the specific node in response is empty.

- **Example 4.2: The pagination ends when the value of the specific node in response dose not exist**

  The REST API returns the last response in the following structure:

  ```
  {}
  ```

  Set the end condition rule as **"EndCondition:$.data": "NonExist"** to end the pagination when the value of the specific node in response dose not exist.

- **Example 4.3: The pagination ends when the value of the specific node in response exists**

  The REST API returns the last response in the following structure:

  ```
  {
      Data: [
          {key1: val991, key2: val992
          },
          {key1: val993, key2: val994
          }
      ],
              Complete: true
  }
  ```

  Set the end condition rule as **"EndCondition:$.Complete": "Exist"** to end the pagination when the value of the specific node in response exists.

- **Example 4.4: The pagination ends when the value of the specific node in response is a user-defined const value**

  The REST API returns the response in the following structure:

```
{
    Data: [
        {key1: val1, key2: val2
        },
        {key1: val3, key2: val4
        }
    ],
            Complete: false
}
```

.....

And the last response is in the following structure:

```
{
    Data: [
        {key1: val991, key2: val992
        },
        {key1: val993, key2: val994
        }
    ],
            Complete: true
}
```

Set the end condition rule as **"EndCondition:$.Complete": "Const:true"** to end the pagination when the value of the specific node in response is a user-defined const value.

- **Example 4.5: The pagination ends when the value of the header key in response equals to user-defined const value**

  The header keys in REST API responses are shown in the structure below:

  Response header 1: `header(Complete->0)`
  .....
  Last Response header: `header(Complete->1)`

  Set the end condition rule as **"EndCondition:headers.Complete": "Const:1"** to end the pagination when the value of the header key in response is equal to user-defined const value.

- **Example 4.6: The pagination ends when the key exists in the response header**

  The header keys in REST API responses are shown in the structure below:

  Response header 1: `header()`
  .....
  Last Response header: `header(CompleteTime->20220920)`

  Set the end condition rule as **"EndCondition:headers.CompleteTime": "Exist"** to end the pagination when the key exists in the response header.

Example 5: Set end condition to avoid endless requests when range rule is not defined

This example provides the configuration steps to send multiple requests when the range rule is not used. The

end condition can be set refer to Example 4.1-4.6 to avoid endless requests. The REST API returns response in the following structure, in which case next page's URL is represented in ***paging.next***.

```
{
    "data": [
        {
            "created_time": "2017-12-12T14:12:20+0000",
            "name": "album1",
            "id": "1809938745705498_1809939942372045"
        },
        {
            "created_time": "2017-12-12T14:14:03+0000",
            "name": "album2",
            "id": "1809938745705498_1809941802371859"
        },
        {
            "created_time": "2017-12-12T14:14:11+0000",
            "name": "album3",
            "id": "1809938745705498_1809941879038518"
        }
    ],
    "paging": {
        "cursors": {
            "after": "MTAxNTExOTQ1MjAwNzI5NDE=",
            "before": "NDMyNzQyODI3OTQw"
        },
        "previous": "https://graph.facebook.com/me/albums?limit=25&before=NDMyNzQyODI3OTQw",
        "next": "https://graph.facebook.com/me/albums?limit=25&after=MTAxNTExOTQ1MjAwNzI5NDE="
    }
}
...
```

The last response is:

```
{
    "data": [],
    "paging": {
        "cursors": {
            "after": "MTAxNTExOTQ1MjAwNzI5NDE=",
            "before": "NDMyNzQyODI3OTQw"
        },
        "previous": "https://graph.facebook.com/me/albums?limit=25&before=NDMyNzQyODI3OTQw",
        "next": "Same with Last Request URL"
    }
}
```

Step 1: Set **Pagination rules** as **"AbsoluteUrl": "$.paging.next"**.

Step 2: If `next` in the last response is always same with the last request URL and not empty, endless requests will be sent. The end condition can be used to avoid endless requests. Therefore, set the end condition rule refer to Example 4.1-4.6.

Example 6: Set the max request number to avoid endless request

Set **MaxRequestNumber** to avoid endless request as shown in the following screenshot:

Example 7: The RFC 5988 pagination rule is supported by default

The backend will automatically get the next URL based on the RFC 5988 style links in the header.

Sample of the HTTP header returned by a service that complies with RFC 5988¶

```
Link: <https://rest-api.acme.com/api/v1/customer.json?opaqueA>; rel="current",
      <https://rest-api.acme.com/api/v1/customer.json?opaqueB>; rel="next",
      <https://rest-api.acme.com/api/v1/customer.json?opaqueC>; rel="first",
      <https://rest-api.acme.com/api/v1/customer.json?opaqueD>; rel="last"
```

In this example, the data source will send the next request to https://rest-api.acme.com/api/v1/customer.json?opaqueB

If there is no URL marked with `rel="next"`, the data source will consider that there are no more pages.

TIP

If you don't want to enable this default pagination rule, you can set `supportRFC5988` to `false` or just delete it in the script.

```
"paginationRules": {
    "supportRFC5988": "false"
}
```

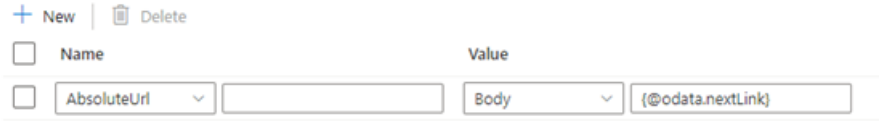
Example 8: The next request URL is from the response body when use pagination in mapping data flows

This example states how to set the pagination rule and the end condition rule in mapping data flows when the next request URL is from the response body.

The response schema is shown below:

Column name	Type
body	{} complex
@odata.context	abc string
@odata.nextLink	abc string
value	[] complex
headers	map

The pagination rules should be set as the following screenshot:



By default, the pagination will stop when body .**{@odata.nextLink}** is null or empty.

But if the value of **@odata.nextLink** in the last response body is equal to the last request URL, then it will lead to the endless loop. To avoid this condition, define end condition rules.

- If **Value** in the last response is **Empty**, then the end condition rule can be set as below:

- If the value of the complete key in the response header equals to true indicates the end of pagination,

then the end condition rule can be set as below:

Example 9: The response format is XML and the next request URL is from the response body when use pagination in mapping data flows

This example states how to set the pagination rule in mapping data flows when the response format is XML and the next request URL is from the response body. As shown in the following screenshot, the first URL is *https://<user>.dfs.core.windows.net/bugfix/test/movie_1.xml*

```
▼<movie>
   <movieId>1</movieId>
   <title>Rambo</title>
   <releaseDate>1945-12-12</releaseDate>
   <rated>true</rated>
   <screenedOn>1946-12-12 12:12:12</screenedOn>
   <ticketPrice>10.345</ticketPrice>
   <next>https:// <user>     .dfs.core.windows.net/bugfix/test/movie_2.xml</next>
 </movie>
```

The response schema is shown below:

Column name	Type
body	complex
movie	complex
movieId	short
next	string
rated	boolean
releaseDate	date
screenedOn	timestamp
ticketPrice	double
title	string
headers	map

The pagination rule syntax is the same as in Example 8 and should be set as below in this example:

Export JSON response as-is

You can use this REST connector to export REST API JSON response as-is to various file-based stores. To achieve

such schema-agnostic copy, skip the "structure" (also called *schema*) section in dataset and schema mapping in copy activity.

Schema mapping

To copy data from REST endpoint to tabular sink, refer to schema mapping.

Next steps

For a list of data stores that Copy Activity supports as sources and sinks in Azure Data Factory, see Supported data stores and formats.

Copy data from and to Salesforce using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 10 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory and Azure Synapse pipelines to copy data from and to Salesforce. It builds on the Copy Activity overview article that presents a general overview of the copy activity.

Supported capabilities

This Salesforce connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks, see the Supported data stores table.

Specifically, this Salesforce connector supports:

- Salesforce Developer, Professional, Enterprise, or Unlimited editions.
- Copying data from and to Salesforce production, sandbox, and custom domain.

> **NOTE**
> This function supports copy of any schema from the above mentioned Salesforce environments, including the Nonprofit Success Pack (NPSP).

The Salesforce connector is built on top of the Salesforce REST/Bulk API. When copying data from Salesforce, the connector automatically chooses between REST and Bulk APIs based on the data size – when the result set is large, Bulk API is used for better performance; You can explicitly set the API version used to read/write data via `apiVersion` property in linked service. When copying data to Salesforce, the connector uses BULK API v1.

> **NOTE**
> The connector no longer sets default version for Salesforce API. For backward compatibility, if a default API version was set before, it keeps working. The default value is 45.0 for source, and 40.0 for sink.

Prerequisites

API permission must be enabled in Salesforce.

Salesforce request limits

Salesforce has limits for both total API requests and concurrent API requests. Note the following points:

- If the number of concurrent requests exceeds the limit, throttling occurs and you see random failures.
- If the total number of requests exceeds the limit, the Salesforce account is blocked for 24 hours.

You might also receive the "REQUEST_LIMIT_EXCEEDED" error message in both scenarios. For more information, see the "API request limits" section in Salesforce developer limits.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

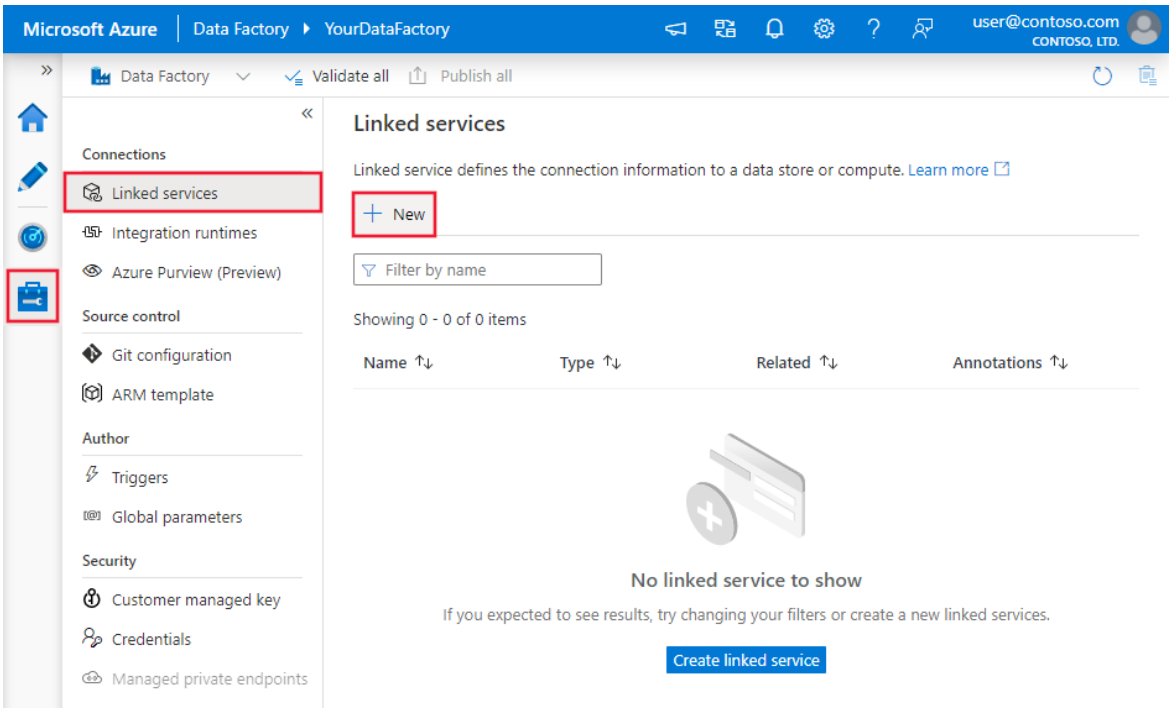
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Salesforce using UI

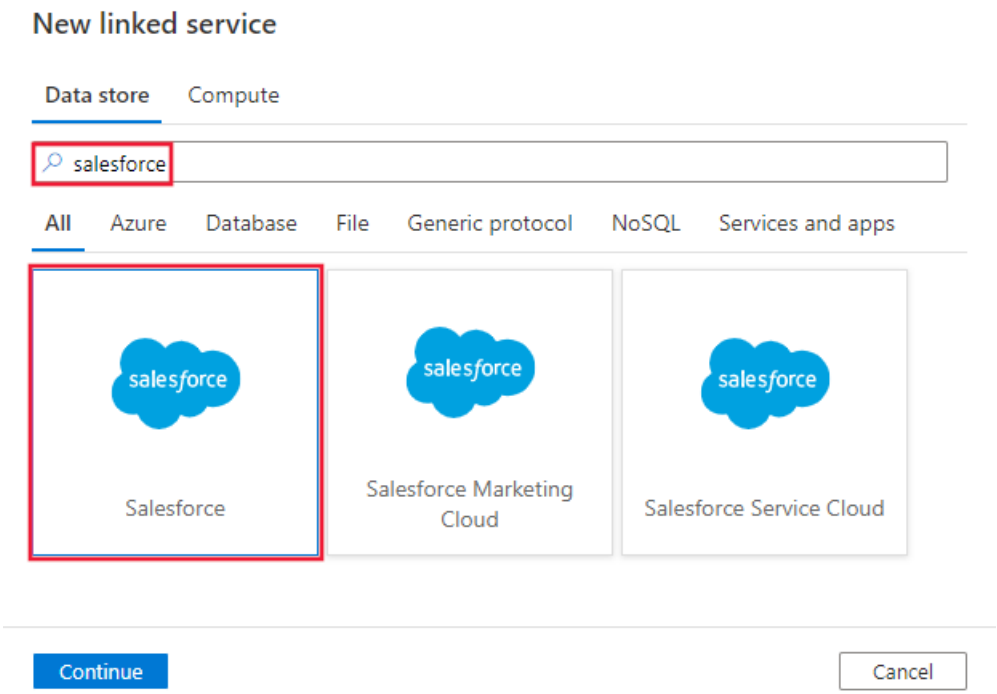
Use the following steps to create a linked service to Salesforce in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

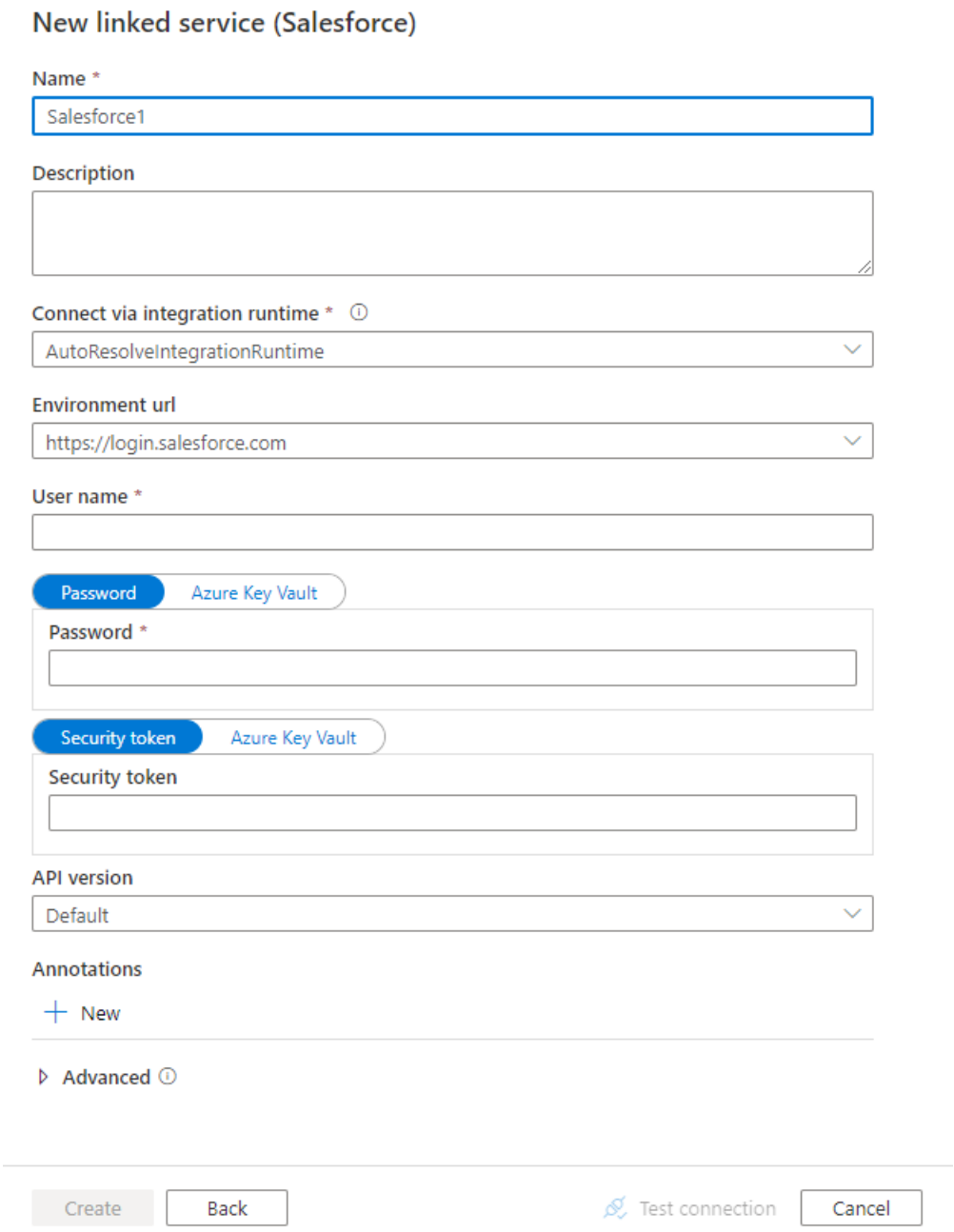
   - Azure Data Factory
   - Azure Synapse



2. Search for Salesforce and select the Salesforce connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to the Salesforce connector.

Linked service properties

The following properties are supported for the Salesforce linked service.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **Salesforce**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
environmentUrl	Specify the URL of the Salesforce instance. - Default is `"https://login.salesforce.com"`. - To copy data from sandbox, specify `"https://test.salesforce.com"`. - To copy data from custom domain, specify, for example, `"https://[domain].my.salesforce.com"`.	No
username	Specify a user name for the user account.	Yes
password	Specify a password for the user account. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
securityToken	Specify a security token for the user account. To learn about security tokens in general, see Security and the API. The security token can be skipped only if you add the Integration Runtime's IP to the trusted IP address list on Salesforce. When using Azure IR, refer to Azure Integration Runtime IP addresses. For instructions on how to get and reset a security token, see Get a security token. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
apiVersion	Specify the Salesforce REST/Bulk API version to use, e.g. `52.0`.	No
connectVia	The integration runtime to be used to connect to the data store. If not specified, it uses the default Azure Integration Runtime.	No

Example: Store credentials

```
{
    "name": "SalesforceLinkedService",
    "properties": {
        "type": "Salesforce",
        "typeProperties": {
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            },
            "securityToken": {
                "type": "SecureString",
                "value": "<security token>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Store credentials in Key Vault

```
{
    "name": "SalesforceLinkedService",
    "properties": {
        "type": "Salesforce",
        "typeProperties": {
            "username": "<username>",
            "password": {
                "type": "AzureKeyVaultSecret",
                "secretName": "<secret name of password in AKV>",
                "store":{
                    "referenceName": "<Azure Key Vault linked service>",
                    "type": "LinkedServiceReference"
                }
            },
            "securityToken": {
                "type": "AzureKeyVaultSecret",
                "secretName": "<secret name of security token in AKV>",
                "store":{
                    "referenceName": "<Azure Key Vault linked service>",
                    "type": "LinkedServiceReference"
                }
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the Salesforce dataset.

To copy data from and to Salesforce, set the type property of the dataset to **SalesforceObject**. The following properties are supported.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **SalesforceObject**.	Yes
objectApiName	The Salesforce object name to retrieve data from.	No for source, Yes for sink

> **IMPORTANT**
>
> The "__c" part of **API Name** is needed for any custom object.

Example:

```
{
    "name": "SalesforceDataset",
    "properties": {
        "type": "SalesforceObject",
        "typeProperties": {
            "objectApiName": "MyTable__c"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Salesforce linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

> **NOTE**
>
> For backward compatibility: When you copy data from Salesforce, if you use the previous "RelationalTable" type dataset, it keeps working while you see a suggestion to switch to the new "SalesforceObject" type.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **RelationalTable**.	Yes
tableName	Name of the table in Salesforce.	No (if "query" in the activity source is specified)

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Salesforce source and sink.

Salesforce as a source type

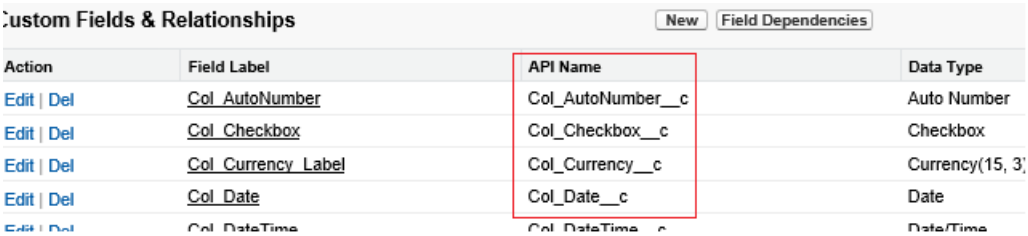
To copy data from Salesforce, set the source type in the copy activity to **SalesforceSource**. The following

properties are supported in the copy activity **source** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **SalesforceSource**.	Yes
query	Use the custom query to read data. You can use Salesforce Object Query Language (SOQL) query or SQL-92 query. See more tips in query tips section. If query is not specified, all the data of the Salesforce object specified in "objectApiName" in dataset will be retrieved.	No (if "objectApiName" in the dataset is specified)
readBehavior	Indicates whether to query the existing records, or query all records including the deleted ones. If not specified, the default behavior is the former. Allowed values: **query** (default), **queryAll**.	No

IMPORTANT

The "__c" part of **API Name** is needed for any custom object.



Example:

```
"activities":[
    {
        "name": "CopyFromSalesforce",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Salesforce input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SalesforceSource",
                "query": "SELECT Col_Currency__c, Col_Date__c, Col_Email__c FROM AllDataType__c"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

NOTE

For backward compatibility: When you copy data from Salesforce, if you use the previous "RelationalSource" type copy, the source keeps working while you see a suggestion to switch to the new "SalesforceSource" type.

NOTE

Salesforce source doesn't support proxy settings in the self-hosted integration runtime, but sink does.

Salesforce as a sink type

To copy data to Salesforce, set the sink type in the copy activity to **SalesforceSink**. The following properties are supported in the copy activity **sink** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to **SalesforceSink**.	Yes
writeBehavior	The write behavior for the operation. Allowed values are **Insert** and **Upsert**.	No (default is Insert)
externalIdFieldName	The name of the external ID field for the upsert operation. The specified field must be defined as "External ID Field" in the Salesforce object. It can't have NULL values in the corresponding input data.	Yes for "Upsert"

PROPERTY	DESCRIPTION	REQUIRED
writeBatchSize	The row count of data written to Salesforce in each batch.	No (default is 5,000)
ignoreNullValues	Indicates whether to ignore NULL values from input data during a write operation. Allowed values are **true** and **false**. - **True**: Leave the data in the destination object unchanged when you do an upsert or update operation. Insert a defined default value when you do an insert operation. - **False**: Update the data in the destination object to NULL when you do an upsert or update operation. Insert a NULL value when you do an insert operation.	No (default is false)
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example: Salesforce sink in a copy activity

```
"activities":[
    {
        "name": "CopyToSalesforce",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Salesforce output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "SalesforceSink",
                "writeBehavior": "Upsert",
                "externalIdFieldName": "CustomerId__c",
                "writeBatchSize": 10000,
                "ignoreNullValues": true
            }
        }
    }
]
```

Query tips

Retrieve data from a Salesforce report

You can retrieve data from Salesforce reports by specifying a query as `{call "<report name>"}`. An example is `"query": "{call \"TestReport\"}"`.

Retrieve deleted records from the Salesforce Recycle Bin

To query the soft deleted records from the Salesforce Recycle Bin, you can specify `readBehavior` as `queryAll`.

Difference between SOQL and SQL query syntax

When copying data from Salesforce, you can use either SOQL query or SQL query. Note that these two has different syntax and functionality support, do not mix it. You are suggested to use the SOQL query, which is natively supported by Salesforce. The following table lists the main differences:

SYNTAX	SOQL MODE	SQL MODE
Column selection	Need to enumerate the fields to be copied in the query, e.g. `SELECT field1, filed2 FROM objectname`	`SELECT *` is supported in addition to column selection.
Quotation marks	Filed/object names cannot be quoted.	Field/object names can be quoted, e.g. `SELECT "id" FROM "Account"`
Datetime format	Refer to details here and samples in next section.	Refer to details here and samples in next section.
Boolean values	Represented as `False` and `True`, e.g. `SELECT … WHERE IsDeleted=True`.	Represented as 0 or 1, e.g. `SELECT … WHERE IsDeleted=1`.
Column renaming	Not supported.	Supported, e.g.: `SELECT a AS b FROM …`.
Relationship	Supported, e.g. `Account_vod__r.nvs_Country__c`.	Not supported.

Retrieve data by using a where clause on the DateTime column

When you specify the SOQL or SQL query, pay attention to the DateTime format difference. For example:

- SOQL sample:

```
SELECT Id, Name, BillingCity FROM Account WHERE LastModifiedDate >=
@{formatDateTime(pipeline().parameters.StartTime,'yyyy-MM-ddTHH:mm:ssZ')} AND LastModifiedDate <
@{formatDateTime(pipeline().parameters.EndTime,'yyyy-MM-ddTHH:mm:ssZ')}
```

- SQL sample:

```
SELECT * FROM Account WHERE LastModifiedDate >=
{ts'@{formatDateTime(pipeline().parameters.StartTime,'yyyy-MM-dd HH:mm:ss')}'} AND LastModifiedDate <
{ts'@{formatDateTime(pipeline().parameters.EndTime,'yyyy-MM-dd HH:mm:ss')}'}
```

Error of MALFORMED_QUERY: Truncated

If you hit error of "MALFORMED_QUERY: Truncated", normally it's due to you have JunctionIdList type column in data and Salesforce has limitation on supporting such data with large number of rows. To mitigate, try to exclude JunctionIdList column or limit the number of rows to copy (you can partition to multiple copy activity runs).

Data type mapping for Salesforce

When you copy data from Salesforce, the following mappings are used from Salesforce data types to interim data types within the service internally. To learn about how the copy activity maps the source schema and data

type to the sink, see Schema and data type mappings.

SALESFORCE DATA TYPE	SERVICE INTERIM DATA TYPE
Auto Number	String
Checkbox	Boolean
Currency	Decimal
Date	DateTime
Date/Time	DateTime
Email	String
ID	String
Lookup Relationship	String
Multi-Select Picklist	String
Number	Decimal
Percent	Decimal
Phone	String
Picklist	String
Text	String
Text Area	String
Text Area (Long)	String
Text Area (Rich)	String
Text (Encrypted)	String
URL	String

NOTE

Salesforce Number type is mapping to Decimal type in Azure Data Factory and Azure Synapse pipelines as a service interim data type. Decimal type honors the defined precision and scale.For data whose decimal places exceeds the defined scale, its value will be rounded off in preview data and copy.To avoid getting such precision loss in Azure Data Factory and Azure Synapse pipelines, consider increasing the decimal places to a reasonably large value in **Custom Field Definition Edit** page of Salesforce.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from and to Salesforce Service Cloud using Azure Data Factory or Synapse Analytics

8/19/2022 • 9 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from and to Salesforce Service Cloud. It builds on the Copy Activity overview article that presents a general overview of the copy activity.

Supported capabilities

This Salesforce Service Cloud connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks, see the Supported data stores table.

Specifically, this Salesforce Service Cloud connector supports:

- Salesforce Developer, Professional, Enterprise, or Unlimited editions.
- Copying data from and to Salesforce production, sandbox, and custom domain.

The Salesforce connector is built on top of the Salesforce REST/Bulk API. By default, when copying data from Salesforce, the connector uses v45 and automatically chooses between REST and Bulk APIs based on the data size – when the result set is large, Bulk API is used for better performance; when writing data to Salesforce, the connector uses v40 of Bulk API. You can also explicitly set the API version used to read/write data via `apiVersion` property in linked service.

Prerequisites

API permission must be enabled in Salesforce.

Salesforce request limits

Salesforce has limits for both total API requests and concurrent API requests. Note the following points:

- If the number of concurrent requests exceeds the limit, throttling occurs and you see random failures.
- If the total number of requests exceeds the limit, the Salesforce account is blocked for 24 hours.

You might also receive the "REQUEST_LIMIT_EXCEEDED" error message in both scenarios. For more information, see the "API request limits" section in Salesforce developer limits.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

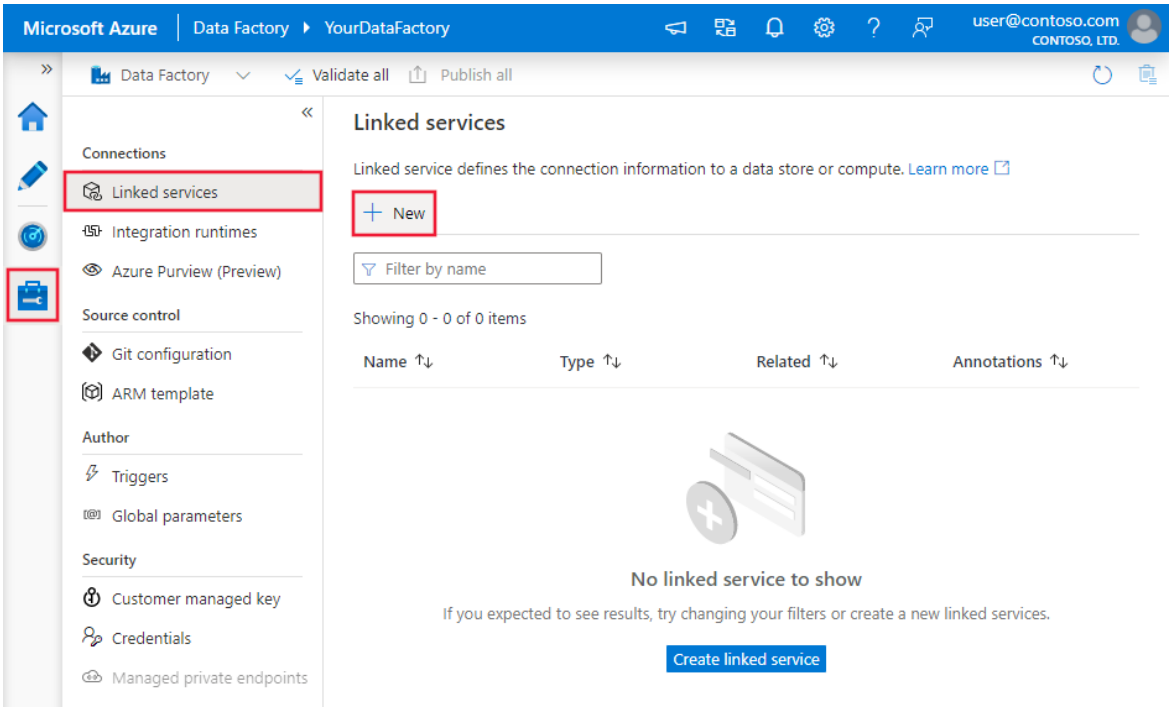
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Salesforce Service Cloud using UI

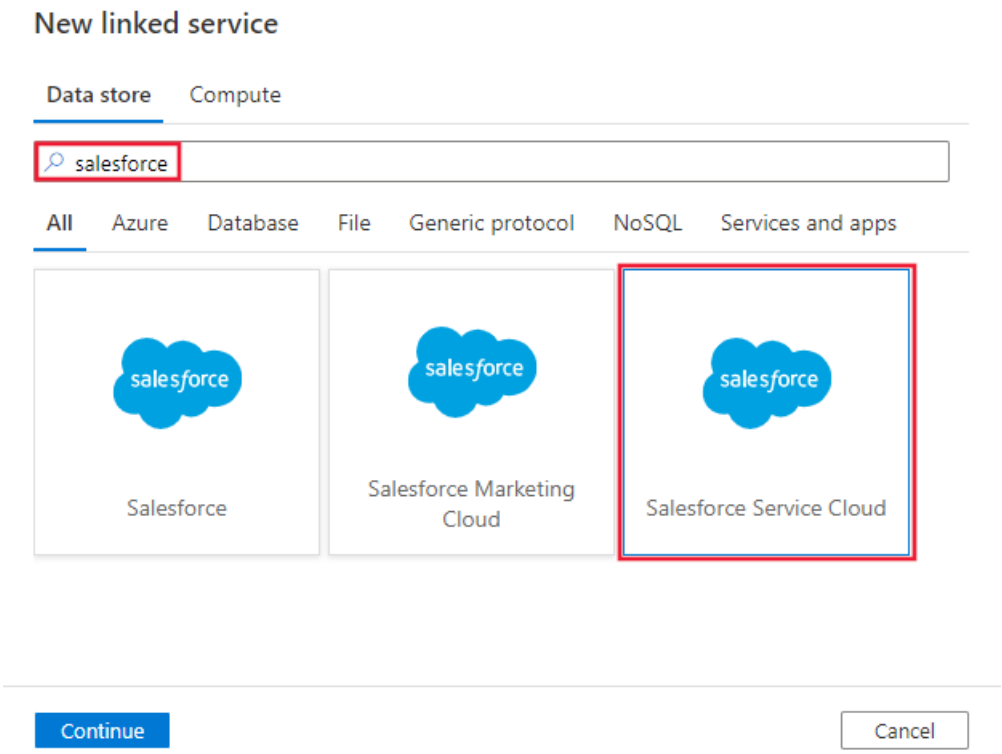
Use the following steps to create a linked service to Salesforce Service Cloud in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

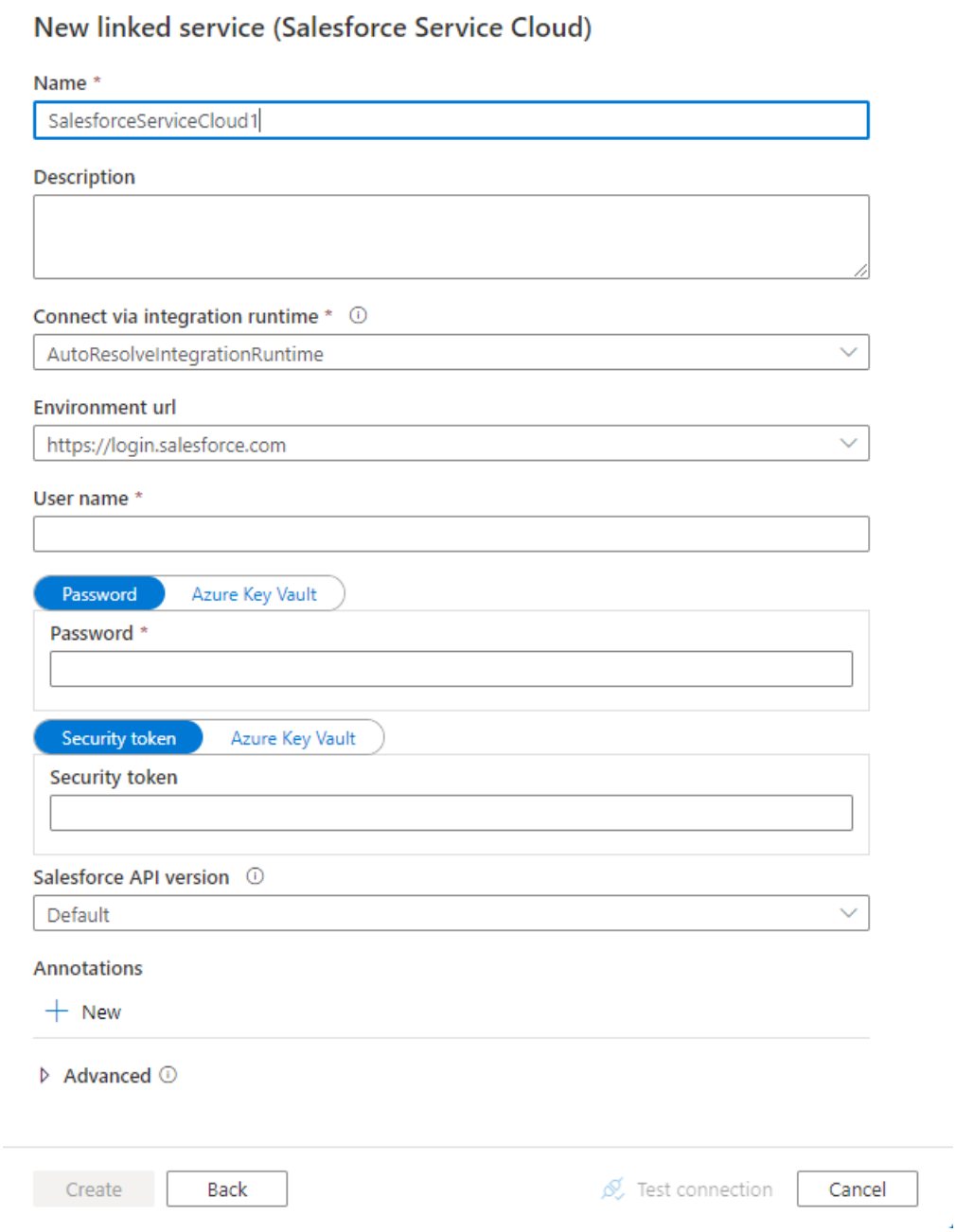
   - Azure Data Factory
   - Azure Synapse



2. Search for Salesforce and select the Salesforce Service Cloud connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to the Salesforce Service Cloud connector.

Linked service properties

The following properties are supported for the Salesforce linked service.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **SalesforceServiceCloud**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
environmentUrl	Specify the URL of the Salesforce Service Cloud instance. - Default is `"https://login.salesforce.com"`. - To copy data from sandbox, specify `"https://test.salesforce.com"`. - To copy data from custom domain, specify, for example, `"https://[domain].my.salesforce.com"`.	No
username	Specify a user name for the user account.	Yes
password	Specify a password for the user account. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
securityToken	Specify a security token for the user account. To learn about security tokens in general, see Security and the API. The security token can be skipped only if you add the Integration Runtime's IP to the trusted IP address list on Salesforce. When using Azure IR, refer to Azure Integration Runtime IP addresses. For instructions on how to get and reset a security token, see Get a security token. Mark this field as a SecureString to store it securely in Data Factory, or reference a secret stored in Azure Key Vault.	No
apiVersion	Specify the Salesforce REST/Bulk API version to use, e.g. `48.0`. By default, the connector uses v45 to copy data from Salesforce, and uses v40 to copy data to Salesforce.	No
connectVia	The integration runtime to be used to connect to the data store. If not specified, it uses the default Azure Integration Runtime.	No

Example: Store credentials

```
{
    "name": "SalesforceServiceCloudLinkedService",
    "properties": {
        "type": "SalesforceServiceCloud",
        "typeProperties": {
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            },
            "securityToken": {
                "type": "SecureString",
                "value": "<security token>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Store credentials in Key Vault

```
{
    "name": "SalesforceServiceCloudLinkedService",
    "properties": {
        "type": "SalesforceServiceCloud",
        "typeProperties": {
            "username": "<username>",
            "password": {
                "type": "AzureKeyVaultSecret",
                "secretName": "<secret name of password in AKV>",
                "store":{
                    "referenceName": "<Azure Key Vault linked service>",
                    "type": "LinkedServiceReference"
                }
            },
            "securityToken": {
                "type": "AzureKeyVaultSecret",
                "secretName": "<secret name of security token in AKV>",
                "store":{
                    "referenceName": "<Azure Key Vault linked service>",
                    "type": "LinkedServiceReference"
                }
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the Salesforce Service Cloud dataset.

To copy data from and to Salesforce Service Cloud, the following properties are supported.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **SalesforceServiceCloudObject**.	Yes
objectApiName	The Salesforce object name to retrieve data from.	No for source, Yes for sink

> **IMPORTANT**
>
> The "__c" part of **API Name** is needed for any custom object.

Custom Object Definition Detail

Singular Label	AllDataType
Plural Label	AllDataTypes
Object Name	AllDataType
API Name	AllDataType__c

Example:

```
{
    "name": "SalesforceServiceCloudDataset",
    "properties": {
        "type": "SalesforceServiceCloudObject",
        "typeProperties": {
            "objectApiName": "MyTable__c"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Salesforce Service Cloud linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **RelationalTable**.	Yes
tableName	Name of the table in Salesforce Service Cloud.	No (if "query" in the activity source is specified)

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Salesforce Service Cloud source and sink.

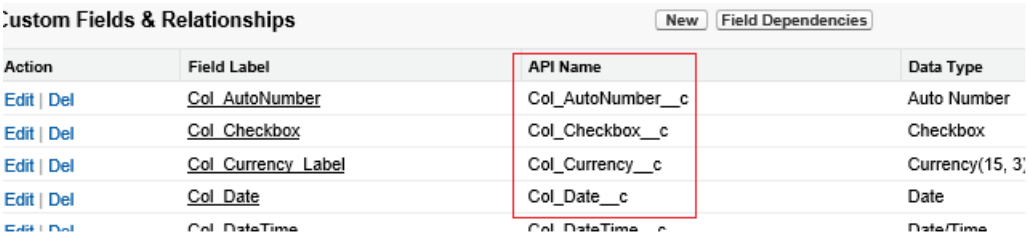
Salesforce Service Cloud as a source type

To copy data from Salesforce Service Cloud, the following properties are supported in the copy activity **source** section.

PROPERTY	DESCRIPTION	REQUIRED

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **SalesforceServiceCloudSource**.	Yes
query	Use the custom query to read data. You can use Salesforce Object Query Language (SOQL) query or SQL-92 query. See more tips in query tips section. If query is not specified, all the data of the Salesforce Service Cloud object specified in "objectApiName" in dataset will be retrieved.	No (if "objectApiName" in the dataset is specified)
readBehavior	Indicates whether to query the existing records, or query all records including the deleted ones. If not specified, the default behavior is the former. Allowed values: **query** (default), **queryAll**.	No

> **IMPORTANT**
>
> The "__c" part of **API Name** is needed for any custom object.



Example:

```
"activities":[
    {
        "name": "CopyFromSalesforceServiceCloud",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Salesforce Service Cloud input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SalesforceServiceCloudSource",
                "query": "SELECT Col_Currency__c, Col_Date__c, Col_Email__c FROM AllDataType__c"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

NOTE

Salesforce Service Cloud source doesn't support proxy settings in the self-hosted integration runtime, but sink does.

Salesforce Service Cloud as a sink type

To copy data to Salesforce Service Cloud, the following properties are supported in the copy activity **sink** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to **SalesforceServiceCloudSink**.	Yes
writeBehavior	The write behavior for the operation. Allowed values are **Insert** and **Upsert**.	No (default is Insert)
externalIdFieldName	The name of the external ID field for the upsert operation. The specified field must be defined as "External ID Field" in the Salesforce Service Cloud object. It can't have NULL values in the corresponding input data.	Yes for "Upsert"
writeBatchSize	The row count of data written to Salesforce Service Cloud in each batch.	No (default is 5,000)

PROPERTY	DESCRIPTION	REQUIRED
ignoreNullValues	Indicates whether to ignore NULL values from input data during a write operation. Allowed values are **true** and **false**. - **True**: Leave the data in the destination object unchanged when you do an upsert or update operation. Insert a defined default value when you do an insert operation. - **False**: Update the data in the destination object to NULL when you do an upsert or update operation. Insert a NULL value when you do an insert operation.	No (default is false)
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToSalesforceServiceCloud",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Salesforce Service Cloud output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "SalesforceServiceCloudSink",
                "writeBehavior": "Upsert",
                "externalIdFieldName": "CustomerId__c",
                "writeBatchSize": 10000,
                "ignoreNullValues": true
            }
        }
    }
]
```

Query tips

Retrieve data from a Salesforce Service Cloud report

You can retrieve data from Salesforce Service Cloud reports by specifying a query as `{call "<report name>"}`.
An example is `"query": "{call \"TestReport\"}"`.

Retrieve deleted records from the Salesforce Service Cloud Recycle Bin

To query the soft deleted records from the Salesforce Service Cloud Recycle Bin, you can specify `readBehavior` as `queryAll`.

Difference between SOQL and SQL query syntax

When copying data from Salesforce Service Cloud, you can use either SOQL query or SQL query. Note that these two has different syntax and functionality support, do not mix it. You are suggested to use the SOQL query, which is natively supported by Salesforce Service Cloud. The following table lists the main differences:

SYNTAX	SOQL MODE	SQL MODE
Column selection	Need to enumerate the fields to be copied in the query, e.g. `SELECT field1, filed2 FROM objectname`	`SELECT *` is supported in addition to column selection.
Quotation marks	Filed/object names cannot be quoted.	Field/object names can be quoted, e.g. `SELECT "id" FROM "Account"`
Datetime format	Refer to details here and samples in next section.	Refer to details here and samples in next section.
Boolean values	Represented as `False` and `True`, e.g. `SELECT … WHERE IsDeleted=True`.	Represented as 0 or 1, e.g. `SELECT … WHERE IsDeleted=1`.
Column renaming	Not supported.	Supported, e.g.: `SELECT a AS b FROM …`.
Relationship	Supported, e.g. `Account_vod__r.nvs_Country__c`.	Not supported.

Retrieve data by using a where clause on the DateTime column

When you specify the SOQL or SQL query, pay attention to the DateTime format difference. For example:

- SOQL sample:

```
SELECT Id, Name, BillingCity FROM Account WHERE LastModifiedDate >=
@{formatDateTime(pipeline().parameters.StartTime,'yyyy-MM-ddTHH:mm:ssZ')} AND LastModifiedDate <
@{formatDateTime(pipeline().parameters.EndTime,'yyyy-MM-ddTHH:mm:ssZ')}
```

- SQL sample:

```
SELECT * FROM Account WHERE LastModifiedDate >=
{ts'@{formatDateTime(pipeline().parameters.StartTime,'yyyy-MM-dd HH:mm:ss')}'} AND LastModifiedDate <
{ts'@{formatDateTime(pipeline().parameters.EndTime,'yyyy-MM-dd HH:mm:ss')}'}
```

Error of MALFORMED_QUERY: Truncated

If you hit error of "MALFORMED_QUERY: Truncated", normally it's due to you have JunctionIdList type column in data and Salesforce has limitation on supporting such data with large number of rows. To mitigate, try to exclude JunctionIdList column or limit the number of rows to copy (you can partition to multiple copy activity runs).

Data type mapping for Salesforce Service Cloud

When you copy data from Salesforce Service Cloud, the following mappings are used from Salesforce Service Cloud data types to interim data types used internally within the service. To learn about how the copy activity maps the source schema and data type to the sink, see Schema and data type mappings.

SALESFORCE SERVICE CLOUD DATA TYPE	INTERIM SERVICE DATA TYPE
Auto Number	String
Checkbox	Boolean
Currency	Decimal
Date	DateTime
Date/Time	DateTime
Email	String
ID	String
Lookup Relationship	String
Multi-Select Picklist	String
Number	Decimal
Percent	Decimal
Phone	String
Picklist	String
Text	String
Text Area	String
Text Area (Long)	String
Text Area (Rich)	String
Text (Encrypted)	String
URL	String

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from Salesforce Marketing Cloud using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory or Synapse Analytics pipelines to copy data from Salesforce Marketing Cloud. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Salesforce Marketing Cloud connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The Salesforce Marketing Cloud connector supports OAuth 2 authentication, and it supports both legacy and enhanced package types. The connector is built on top of the Salesforce Marketing Cloud REST API.

> **NOTE**
> This connector doesn't support retrieving custom objects or custom data extensions.

Getting started

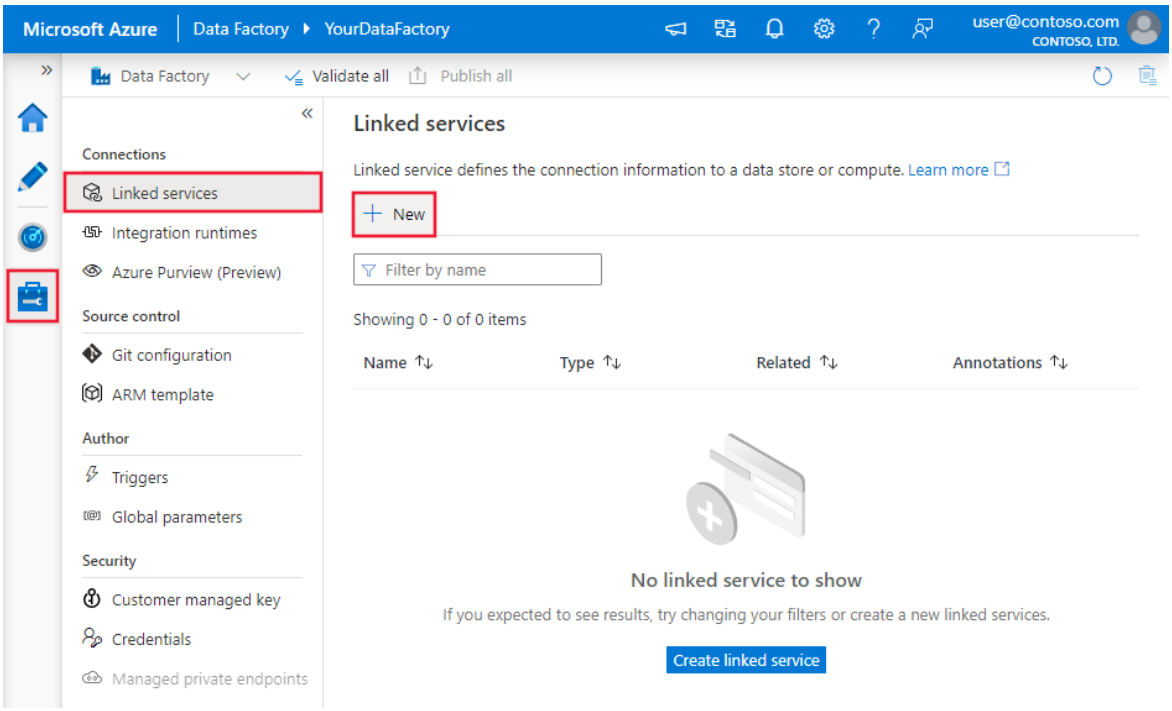
You can create a pipeline with copy activity using .NET SDK, Python SDK, Azure PowerShell, REST API, or Azure Resource Manager template. See Copy activity tutorial for step-by-step instructions to create a pipeline with a copy activity.

Create a linked service to Salesforce Marketing Cloud using UI

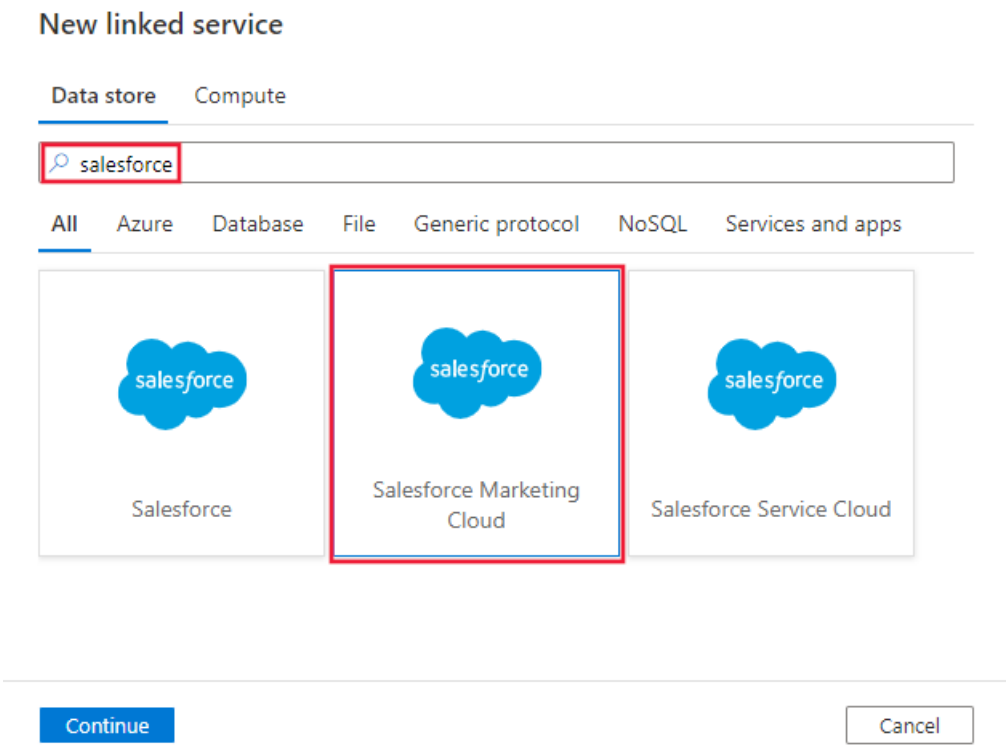
Use the following steps to create a linked service to Salesforce Marketing Cloud in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

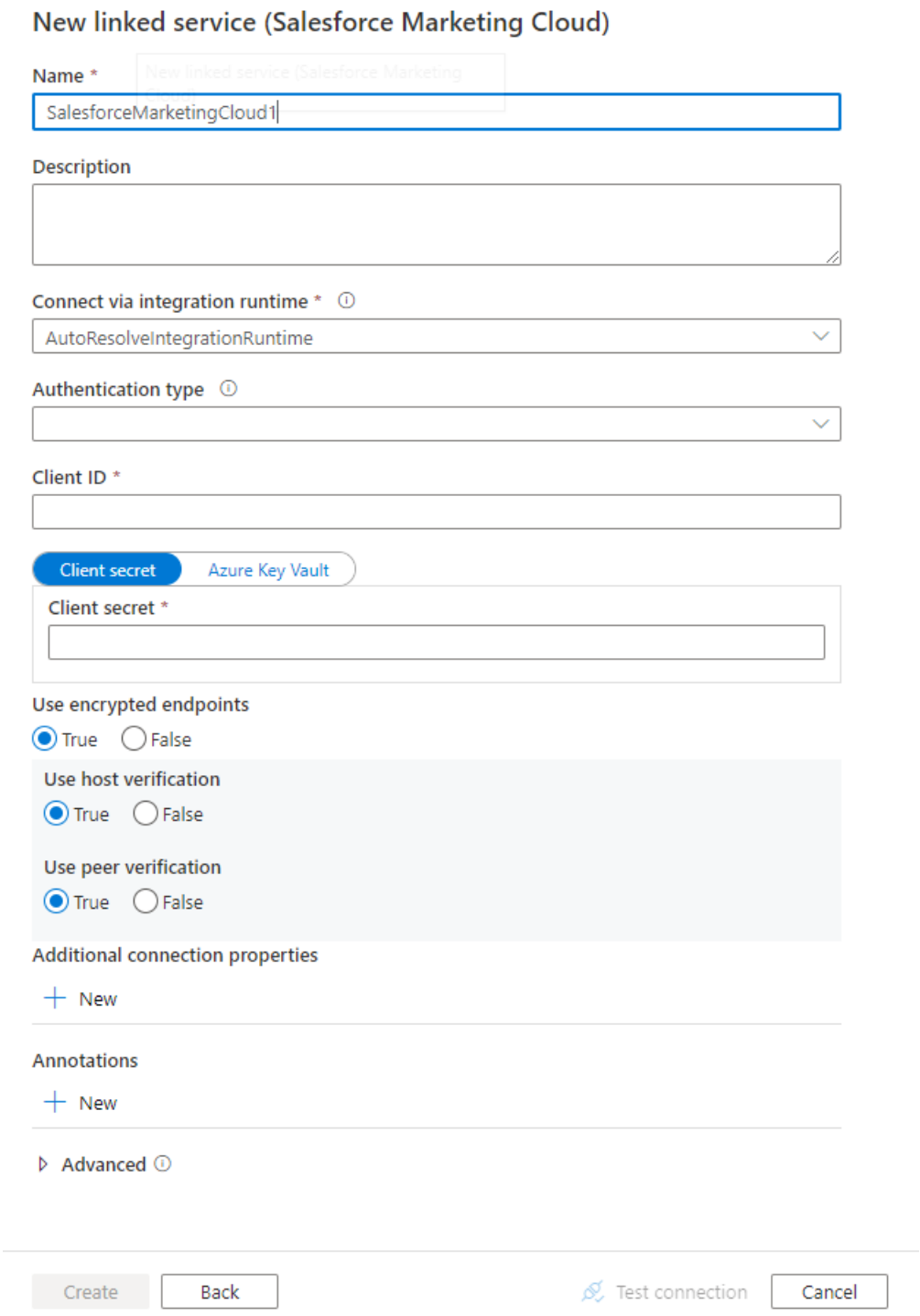
   - Azure Data Factory
   - Azure Synapse



2. Search for Salesforce and select the Salesforce Marketing Cloud connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Salesforce Marketing Cloud connector.

Linked service properties

The following properties are supported for Salesforce Marketing Cloud linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **SalesforceMarketingCloud**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionProperties	A group of properties that defines how to connect to Salesforce Marketing Cloud.	Yes
Under `connectionProperties`*:*		
authenticationType	Specifies the authentication method to use. Allowed values are `Enhanced sts OAuth 2.0` or `OAuth_2.0`. Salesforce Marketing Cloud legacy package only supports `OAuth_2.0`, while enhanced package needs `Enhanced sts OAuth 2.0`. Since August 1, 2019, Salesforce Marketing Cloud has removed the ability to create legacy packages. All new packages are enhanced packages.	Yes
host	For enhanced package, the host should be your subdomain which is represented by a 28-character string starting with the letters "mc", e.g. `mc563885gzs27c5t9-63k636ttgm`. For legacy package, specify `www.exacttargetapis.com`.	Yes
clientId	The client ID associated with the Salesforce Marketing Cloud application.	Yes
clientSecret	The client secret associated with the Salesforce Marketing Cloud application. You can choose to mark this field as a SecureString to store it securely in the service, or store the secret in Azure Key Vault and let the service copy activity pull from there when performing data copy - learn more from Store credentials in Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example: using enhanced STS OAuth 2 authentication for enhanced package

```
{
    "name": "SalesforceMarketingCloudLinkedService",
    "properties": {
        "type": "SalesforceMarketingCloud",
        "typeProperties": {
            "connectionProperties": {
                "host": "<subdomain e.g. mc563885gzs27c5t9-63k636ttgm>",
                "authenticationType": "Enhanced sts OAuth 2.0",
                "clientId": "<clientId>",
                "clientSecret": {
                     "type": "SecureString",
                     "value": "<clientSecret>"
             },
                "useEncryptedEndpoints": true,
                "useHostVerification": true,
                "usePeerVerification": true
            }
        }
    }
}
```

Example: using OAuth 2 authentication for legacy package

```
{
    "name": "SalesforceMarketingCloudLinkedService",
    "properties": {
        "type": "SalesforceMarketingCloud",
        "typeProperties": {
            "connectionProperties": {
                "host": "www.exacttargetapis.com",
                "authenticationType": "OAuth_2.0",
                "clientId": "<clientId>",
                "clientSecret": {
                     "type": "SecureString",
                     "value": "<clientSecret>"
             },
                "useEncryptedEndpoints": true,
                "useHostVerification": true,
                "usePeerVerification": true
            }
        }
    }
}
```

If you were using Salesforce Marketing Cloud linked service with the following payload, it is still supported as-is, while you are suggested to use the new one going forward which adds enhanced package support.

```
{
    "name": "SalesforceMarketingCloudLinkedService",
    "properties": {
        "type": "SalesforceMarketingCloud",
        "typeProperties": {
            "clientId": "<clientId>",
            "clientSecret": {
                 "type": "SecureString",
                 "value": "<clientSecret>"
            },
            "useEncryptedEndpoints": true,
            "useHostVerification": true,
            "usePeerVerification": true
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Salesforce Marketing Cloud dataset.

To copy data from Salesforce Marketing Cloud, set the type property of the dataset to **SalesforceMarketingCloudObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **SalesforceMarketingCloudObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "SalesforceMarketingCloudDataset",
    "properties": {
        "type": "SalesforceMarketingCloudObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<SalesforceMarketingCloud linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Salesforce Marketing Cloud source.

Salesforce Marketing Cloud as source

To copy data from Salesforce Marketing Cloud, set the source type in the copy activity to **SalesforceMarketingCloudSource**. The following properties are supported in the copy activity **source**

section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **SalesforceMarketingCloudSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromSalesforceMarketingCloud",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SalesforceMarketingCloud input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SalesforceMarketingCloudSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

NOTE

Contacts table is not supported.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from SAP Business Warehouse via Open Hub using Azure Data Factory or Synapse Analytics

8/19/2022 • 10 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from an SAP Business Warehouse (BW) via Open Hub. It builds on the copy activity overview article that presents a general overview of copy activity.

> **TIP**
> To learn about overall support for the SAP data integration scenario, see SAP data integration whitepaper with detailed introduction on each SAP connector, comparison and guidance.

Supported capabilities

This SAP Business Warehouse Open Hub connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	②
Lookup activity	②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

Specifically, this SAP Business Warehouse Open Hub connector supports:

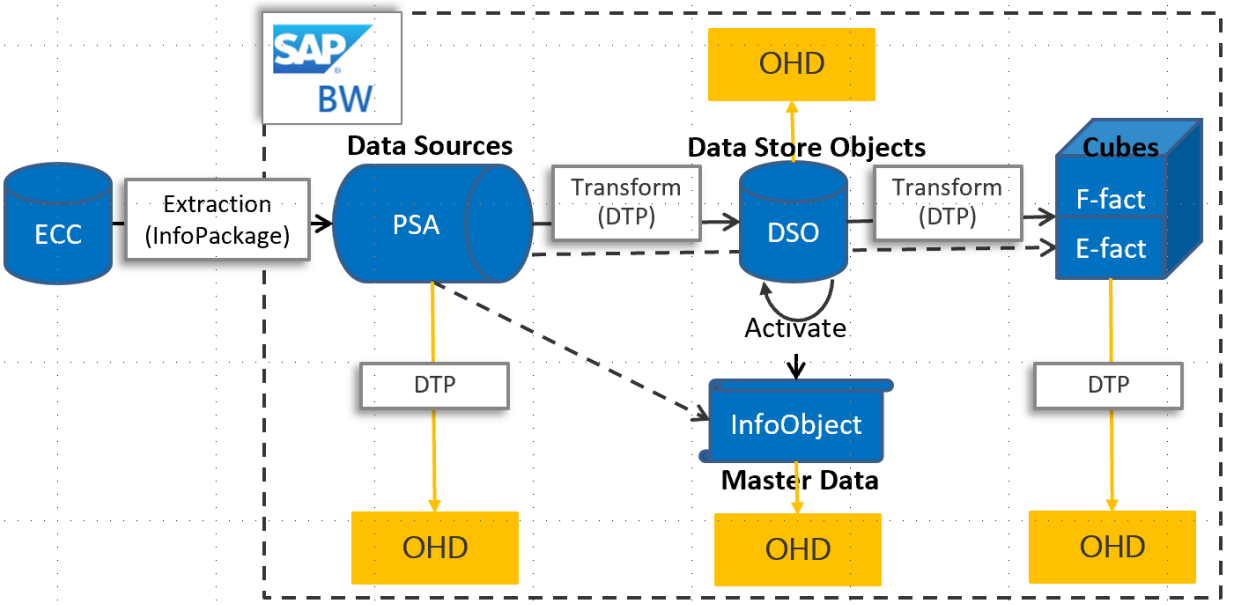
- SAP Business Warehouse **version 7.01 or higher (in a recent SAP Support Package Stack released after the year 2015)**. SAP BW/4HANA is not supported by this connector.
- Copying data via Open Hub Destination local table, which underneath can be DSO, InfoCube, MultiProvider, DataSource, etc.
- Copying data using basic authentication.
- Connecting to an SAP application server or SAP message server.
- Retrieving data via RFC.

SAP BW Open Hub Integration

SAP BW Open Hub Service is an efficient way to extract data from SAP BW. The following diagram shows one of the typical flows customers have in their SAP system, in which case data flows from SAP ECC -> PSA -> DSO -> Cube.

SAP BW Open Hub Destination (OHD) defines the target to which the SAP data is relayed. Any objects supported by SAP Data Transfer Process (DTP) can be used as open hub data sources, for example, DSO, InfoCube, DataSource, etc. Open Hub Destination type - where the relayed data is stored - can be database tables (local or remote) and flat files. This SAP BW Open Hub connector support copying data from OHD local table in BW. In

case you are using other types, you can directly connect to the database or file system using other connectors.



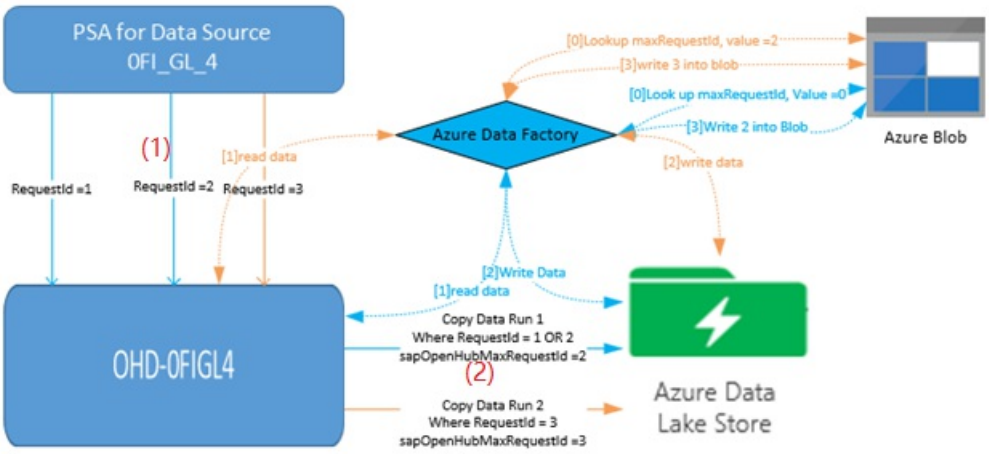
Delta extraction flow

The SAP BW Open Hub Connector offers two optional properties: `excludeLastRequest` and `baseRequestId` which can be used to handle delta load from Open Hub.

- **excludeLastRequestId**: Whether to exclude the records of the last request. Default value is true.
- **baseRequestId**: The ID of request for delta loading. Once it is set, only data with requestId larger than the value of this property will be retrieved.

Overall, the extraction from SAP InfoProviders consists of two steps:

1. **SAP BW Data Transfer Process (DTP)** This step copies the data from an SAP BW InfoProvider to an SAP BW Open Hub table
2. **Data copy** In this step, the Open Hub table is read by the connector



In the first step, a DTP is executed. Each execution creates a new SAP request ID. The request ID is stored in the Open Hub table and is then used by the connector to identify the delta. The two steps run asynchronously: the DTP is triggered by SAP, and the data copy is triggered through the service.

By default, the service is not reading the latest delta from the Open Hub table (option "exclude last request" is true). Hereby, the data in the service is not 100% up to date with the data in the Open Hub table (the last delta is missing). In return, this procedure ensures that no rows get lost caused by the asynchronous extraction. It works fine even when the service is reading the Open Hub table while the DTP is still writing into the same table.

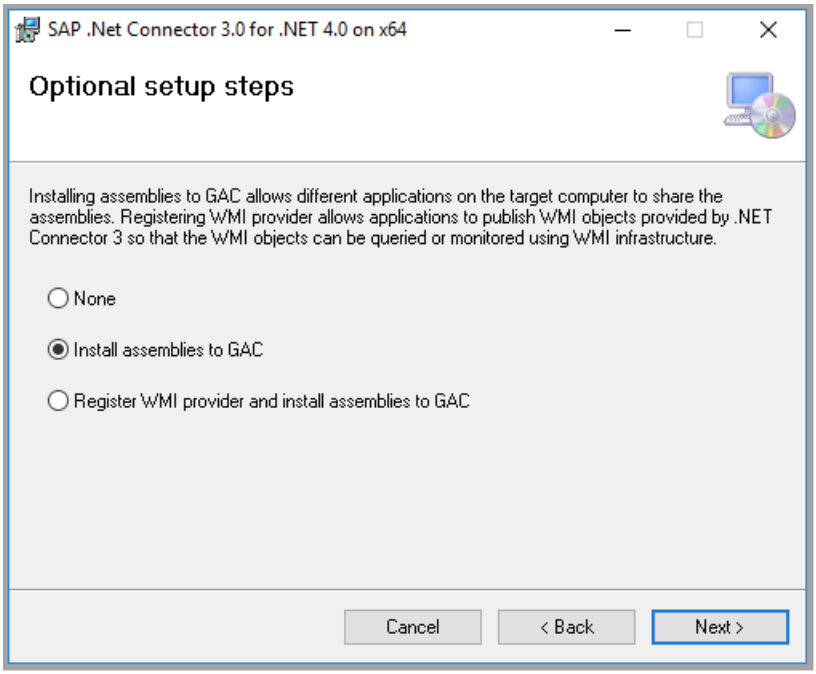
You typically store the max copied request ID in the last run by the service in a staging data store (such as Azure Blob in above diagram). Therefore, the same request is not read a second time by the service in the subsequent run. Meanwhile, note the data is not automatically deleted from the Open Hub table.

For proper delta handling, it is not allowed to have request IDs from different DTPs in the same Open Hub table. Therefore, you must not create more than one DTP for each Open Hub Destination (OHD). When needing Full and Delta extraction from the same InfoProvider, you should create two OHDs for the same InfoProvider.

Prerequisites

To use this SAP Business Warehouse Open Hub connector, you need to:

- Set up a Self-hosted Integration Runtime with version 3.13 or above. See Self-hosted Integration Runtime article for details.
- Download the **64-bit SAP .NET Connector 3.0** from SAP's website, and install it on the Self-hosted IR machine. When installing, in the optional setup steps window, make sure you select the **Install Assemblies to GAC** option as shown in the following image.



- SAP user being used in the BW connector needs to have following permissions:
  - Authorization for RFC and SAP BW.
  - Permissions to the "Execute" Activity of Authorization Object "S_SDSAUTH".
- Create SAP Open Hub Destination type as **Database Table** with "Technical Key" option checked. It is also recommended to leave the Deleting Data from Table as unchecked although it is not required. Use the DTP (directly execute or integrate into existing process chain) to land data from source object (such as cube) you have chosen to the open hub destination table.

Getting started

> **TIP**
> For a walkthrough of using SAP BW Open Hub connector, see Load data from SAP Business Warehouse (BW).

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool

- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

The following sections provide details about properties that are used to define entities specific to SAP Business Warehouse Open Hub connector.

Linked service properties

The following properties are supported for SAP Business Warehouse Open Hub linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **SapOpenHub**	Yes
server	Name of the server on which the SAP BW instance resides.	Yes
systemNumber	System number of the SAP BW system. Allowed value: two-digit decimal number represented as a string.	Yes
messageServer	The host name of the SAP message server. Use to connect to an SAP message server.	No
messageServerService	The service name or port number of the message server. Use to connect to an SAP message server.	No
systemId	The ID of the SAP system where the table is located. Use to connect to an SAP message server.	No
logonGroup	The logon group for the SAP system. Use to connect to an SAP message server.	No
clientId	Client ID of the client in the SAP W system. Allowed value: three-digit decimal number represented as a string.	Yes
language	Language that the SAP system uses.	No (default value is **EN**)
userName	Name of the user who has access to the SAP server.	Yes

PROPERTY	DESCRIPTION	REQUIRED
password	Password for the user. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. A Self-hosted Integration Runtime is required as mentioned in Prerequisites.	Yes

Example:

```
{
    "name": "SapBwOpenHubLinkedService",
    "properties": {
        "type": "SapOpenHub",
        "typeProperties": {
            "server": "<server name>",
            "systemNumber": "<system number>",
            "clientId": "<client id>",
            "userName": "<SAP user>",
            "password": {
                "type": "SecureString",
                "value": "<Password for SAP user>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the SAP BW Open Hub dataset.

To copy data from and to SAP BW Open Hub, set the type property of the dataset to **SapOpenHubTable**. The following properties are supported.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **SapOpenHubTable**.	Yes
openHubDestinationName	The name of the Open Hub Destination to copy data from.	Yes

If you were setting `excludeLastRequest` and `baseRequestId` in dataset, it is still supported as-is, while you are suggested to use the new model in activity source going forward.

Example:

```
{
    "name": "SAPBWOpenHubDataset",
    "properties": {
        "type": "SapOpenHubTable",
        "typeProperties": {
            "openHubDestinationName": "<open hub destination name>"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<SAP BW Open Hub linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by SAP BW Open Hub source.

SAP BW Open Hub as source

To copy data from SAP BW Open Hub, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the copy activity source must be set to **SapOpenHubSource**.	Yes
excludeLastRequest	Whether to exclude the records of the last request.	No (default is **true**)
baseRequestId	The ID of request for delta loading. Once it is set, only data with requestId **larger than** the value of this property will be retrieved.	No
customRfcReadTableFunctionModule	A custom RFC function module that can be used to read data from an SAP table. You can use a custom RFC function module to define how the data is retrieved from your SAP system and returned to the service. The custom function module must have an interface implemented (import, export, tables) that's similar to `/SAPDS/RFC_READ_TABLE2`, which is the default interface used by the service.	No
sapDataColumnDelimiter	The single character that is used as delimiter passed to SAP RFC to split the output data.	No

TIP

If your Open Hub table only contains the data generated by single request ID, for example, you always do full load and overwrite the existing data in the table, or you only run the DTP once for test, remember to uncheck the "excludeLastRequest" option in order to copy the data out.

To speed up the data loading, you can set `parallelCopies` on the copy activity to load data from SAP BW Open Hub in parallel. For example, if you set `parallelCopies` to four, the service concurrently executes four RFC calls, and each RFC call retrieves a portion of data from your SAP BW Open Hub table partitioned by the DTP request ID and package ID. This applies when the number of unique DTP request ID + package ID is bigger than the value of `parallelCopies`. When copying data into file-based data store, it's also recommanded to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

Example:

```
"activities":[
    {
        "name": "CopyFromSAPBWOpenHub",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SAP BW Open Hub input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SapOpenHubSource",
                "excludeLastRequest": true
            },
            "sink": {
                "type": "<sink type>"
            },
            "parallelCopies": 4
        }
    }
]
```

Data type mapping for SAP BW Open Hub

When copying data from SAP BW Open Hub, the following mappings are used from SAP BW data types to interim data types used internally within the service. See Schema and data type mappings to learn about how copy activity maps the source schema and data type to the sink.

SAP ABAP TYPE	INTERIM SERVICE DATA TYPE
C (String)	String
I (integer)	Int32

SAP ABAP TYPE	INTERIM SERVICE DATA TYPE
F (Float)	Double
D (Date)	String
T (Time)	String
P (BCD Packed, Currency, Decimal, Qty)	Decimal
N (Numc)	String
X (Binary and Raw)	String

Lookup activity properties

To learn details about the properties, check Lookup activity.

Troubleshooting tips

Symptoms: If you are running SAP BW on HANA and observe only subset of data is copied over using copy activity (1 million rows), the possible cause is that you enable "SAP HANA Execution" option in your DTP, in which case the service can only retrieve the first batch of data.

Resolution: Disable "SAP HANA Execution" option in DTP, reprocess the data, then try executing the copy activity again.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from SAP Business Warehouse with Azure Data Factory or Synapse Analytics

8/19/2022 • 11 minutes to read • Edit Online

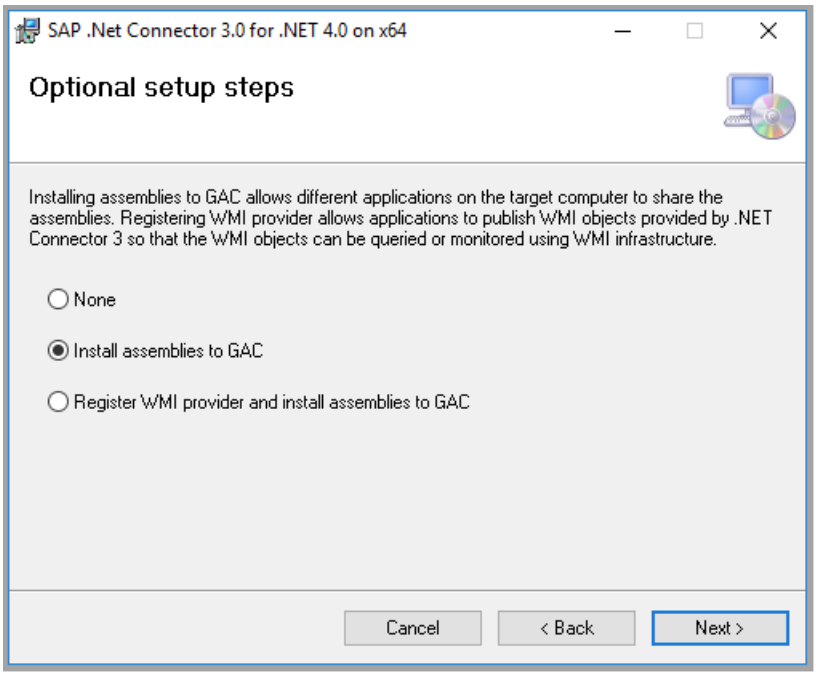
APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article shows how to use Azure Data Factory or Synapse Analytics pipelines to copy data from SAP Business Warehouse (BW) via Open Hub to Azure Data Lake Storage Gen2. You can use a similar process to copy data to other supported sink data stores.

> **TIP**
> For general information about copying data from SAP BW, including SAP BW Open Hub integration and delta extraction flow, see Copy data from SAP Business Warehouse via Open Hub by using Azure Data Factory.

Prerequisites

- **Azure Data Factory or Synapse workspace**: If you don't have one, follow the steps to create a data factory or create a Synapse workspace.
- **SAP BW Open Hub Destination (OHD) with destination type "Database Table"**: To create an OHD or to check that your OHD is configured correctly for integration with the service, see the SAP BW Open Hub Destination configurations section of this article.
- **The SAP BW user needs the following permissions**:
  - Authorization for Remote Function Calls (RFC) and SAP BW.
  - Permissions to the "Execute" activity of the **S_SDSAUTH** authorization object.
- **A self-hosted integration runtime (IR) with SAP .NET connector 3.0**. Follow these setup steps:
  1. Install and register the self-hosted integration runtime, version 3.13 or later. (This process is described later in this article.)
  2. Download the 64-bit SAP Connector for Microsoft .NET 3.0 from SAP's website, and install it on the same computer as the self-hosted IR. During installation, make sure that you select **Install Assemblies to GAC** in the **Optional setup steps** dialog box, as the following image shows:



Do a full copy from SAP BW Open Hub

In the Azure portal, go to your service. Select **Open** on the **Open Azure Data Factory Studio** or **Open Synapse Studio** tile to open the service UI in a separate tab.

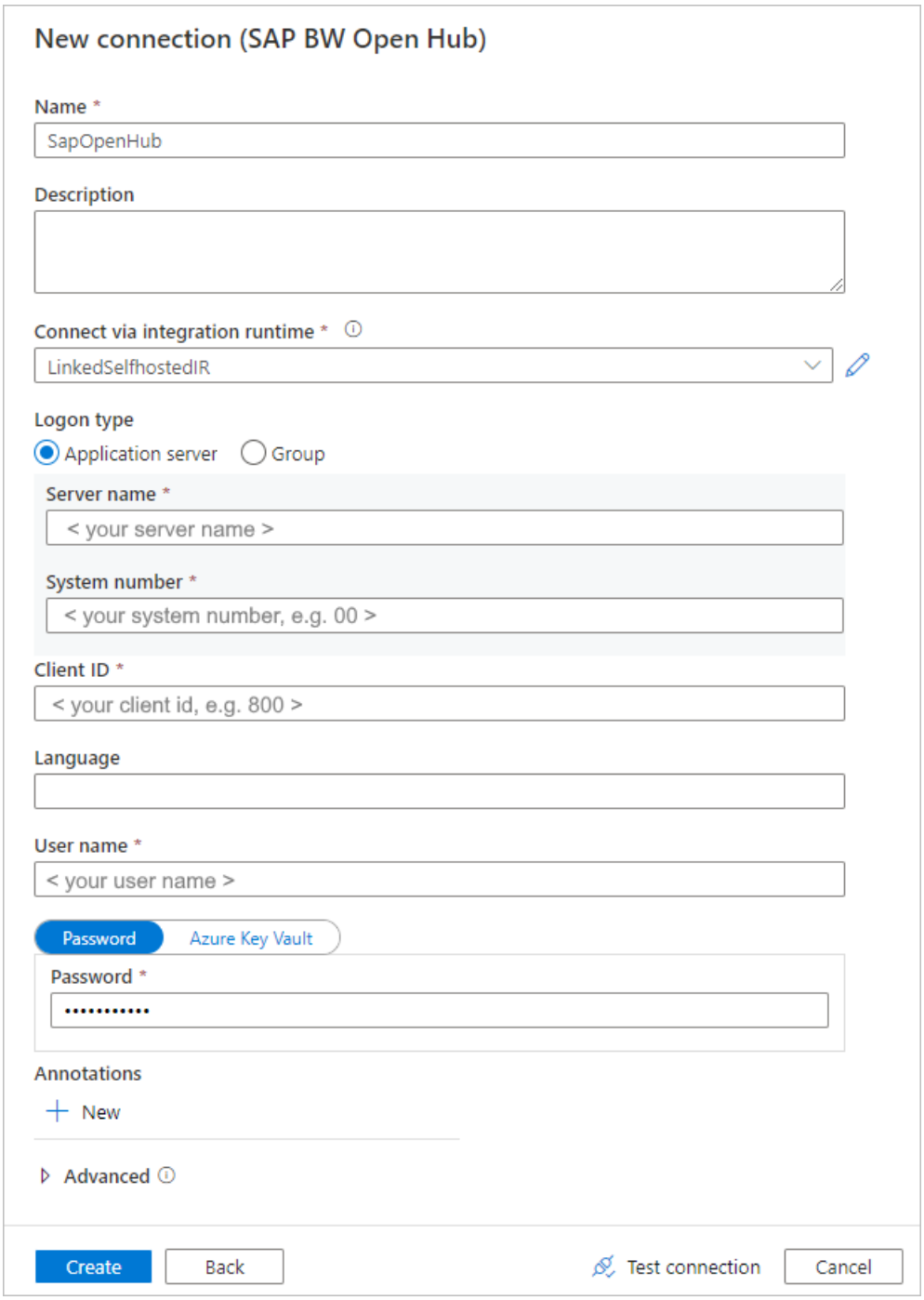
1. On the home page, select **Ingest** to open the Copy Data tool.
2. On the **Properties** page, choose **Built-in copy task** under **Task type**, and choose **Run once now** under **Task cadence or task schedule**, and then select **Next**.
3. On the **Source data store** page, select **+ New connection**. Select **SAP BW Open Hub** from the connector gallery, and then select **Continue**. To filter the connectors, you can type **SAP** in the search box.
4. On the **New connection (SAP BW Open Hub)** page, follow these steps to create a new connection.
   a. From the **Connect via integration runtime** list, select an existing self-hosted IR. Or, choose to create one if you don't have one yet.

      To create a new self-hosted IR, select **+New**, and then select **Self-hosted**. Enter a **Name**, and then select **Next**. Select **Express setup** to install on the current computer, or follow the **Manual setup** steps that are provided.

      As mentioned in Prerequisites, make sure that you have SAP Connector for Microsoft .NET 3.0 installed on the same computer where the self-hosted IR is running.

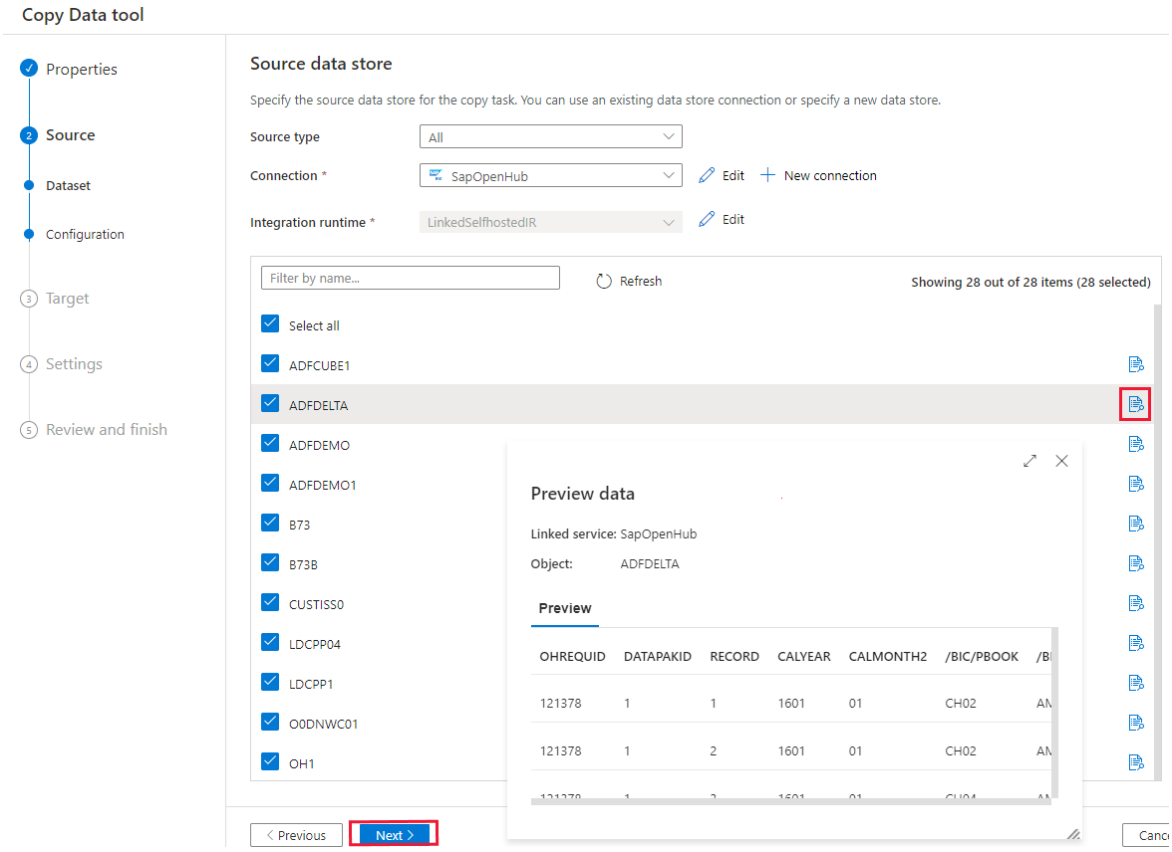
   b. Fill in the SAP BW **Server name**, **System number**, **Client ID**, **Language** (if other than **EN**), **User name**, and **Password**.

   c. Select **Test connection** to validate the settings, and then select **Create**.



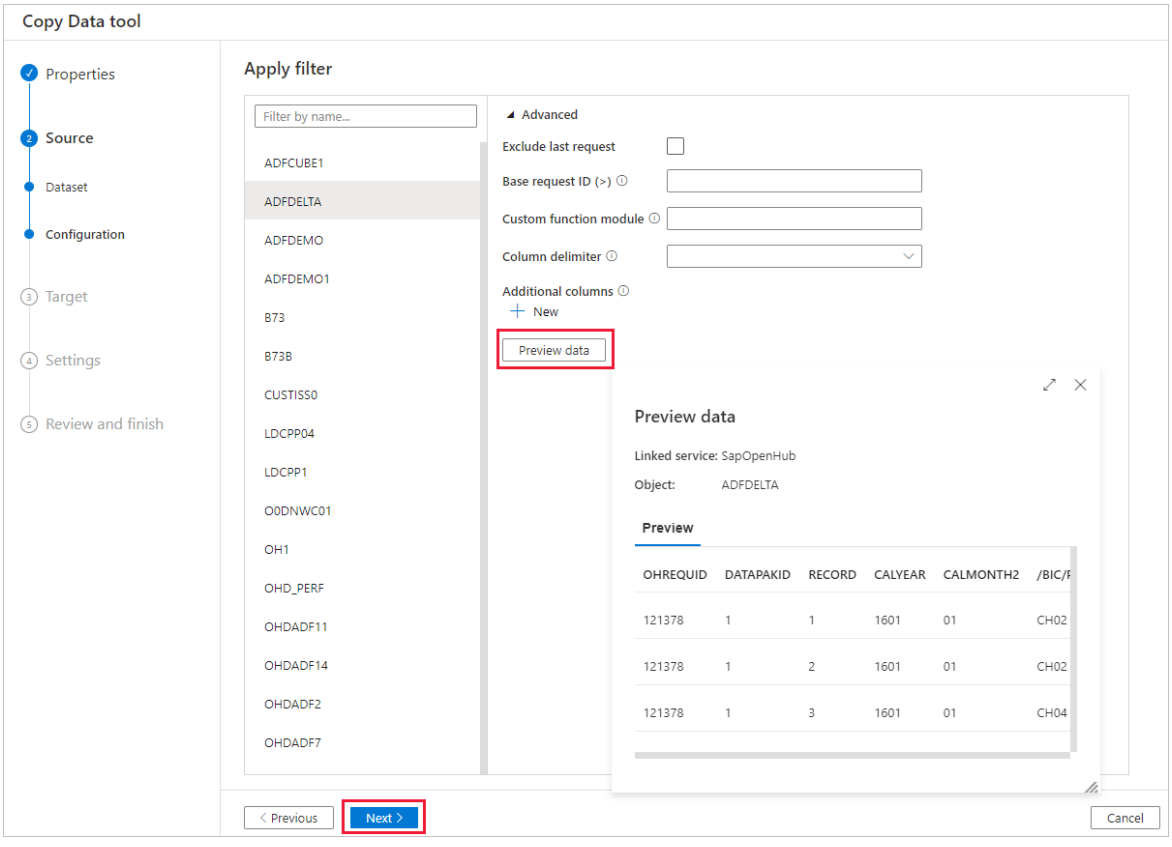
d. On the **Source data store** page, select the newly created connection in the **Connection** block.

e. In the section of selecting Open Hub destinations, browse the Open Hub Destinations that are available in your SAP BW. You can preview the data in each destination by selecting the preview button at the end of each row. Select the OHD to copy data from, and then select **Next**.



5. Specify a filter, if you need one. If your OHD only contains data from a single data-transfer process (DTP) execution with a single request ID, or you're sure that your DTP is finished and you want to copy the data, clear the **Exclude Last Request** check box in **Advanced** section. You can preview the data by selecting **Preview data** button.

   Learn more about these settings in the SAP BW Open Hub Destination configurations section of this article. Then select **Next**.



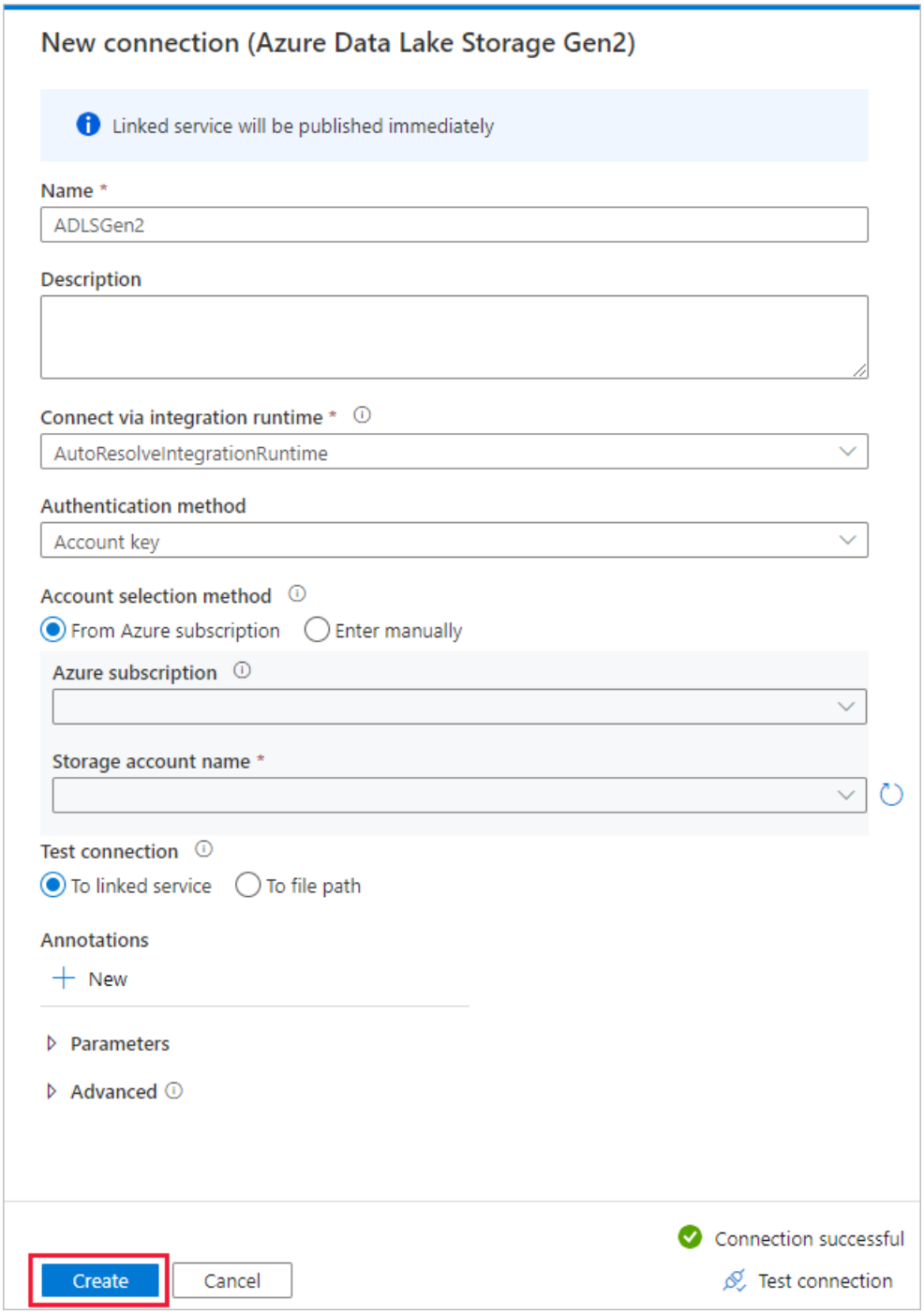
6. On the **Destination data store** page, select **+ New connection** > **Azure Data Lake Storage Gen2** >

Continue.

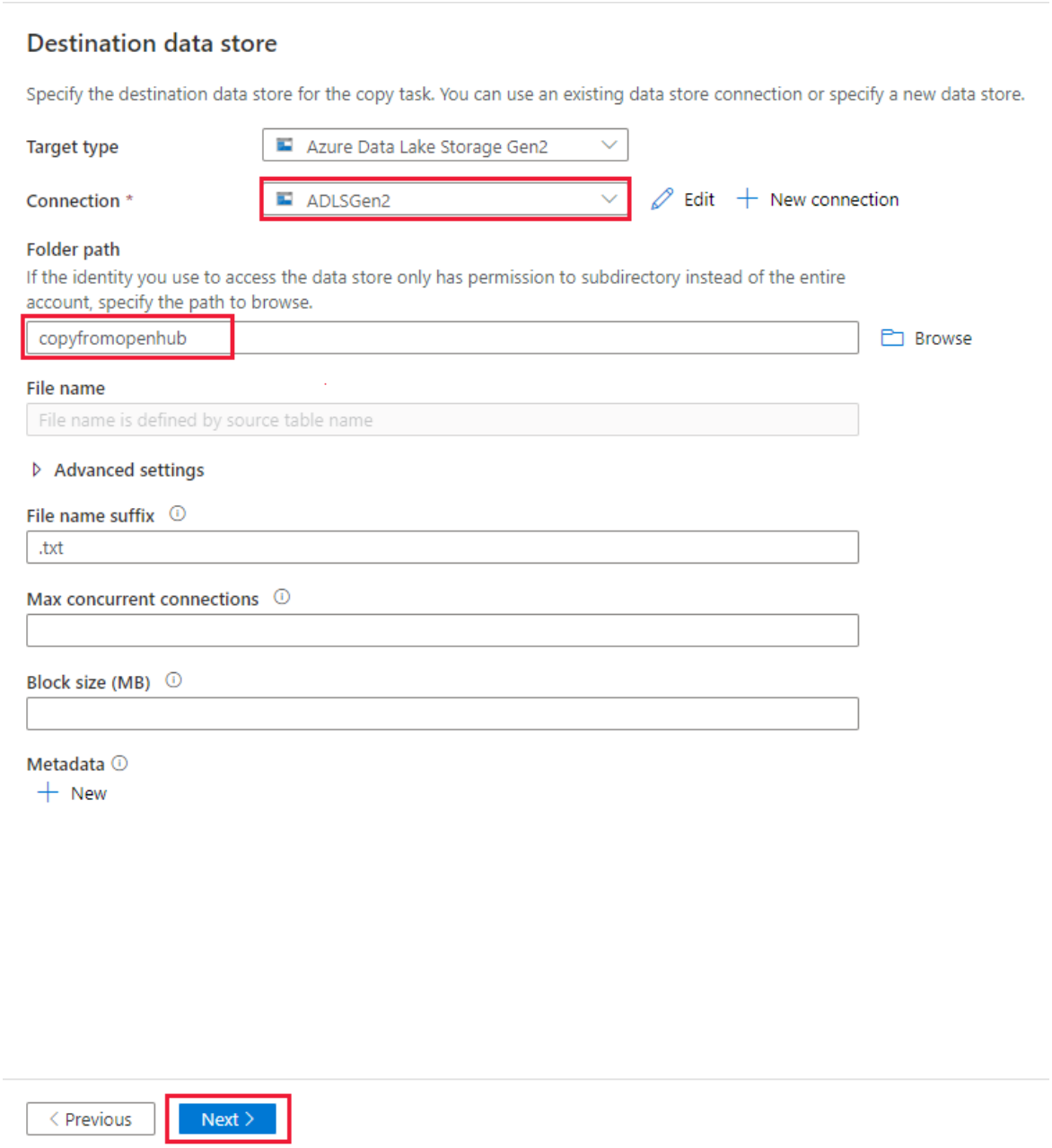
7. On the **New connection (Azure Data Lake Storage Gen2)** page, follow these steps to create a connection.

   a. Select your Data Lake Storage Gen2-capable account from the **Name** drop-down list.

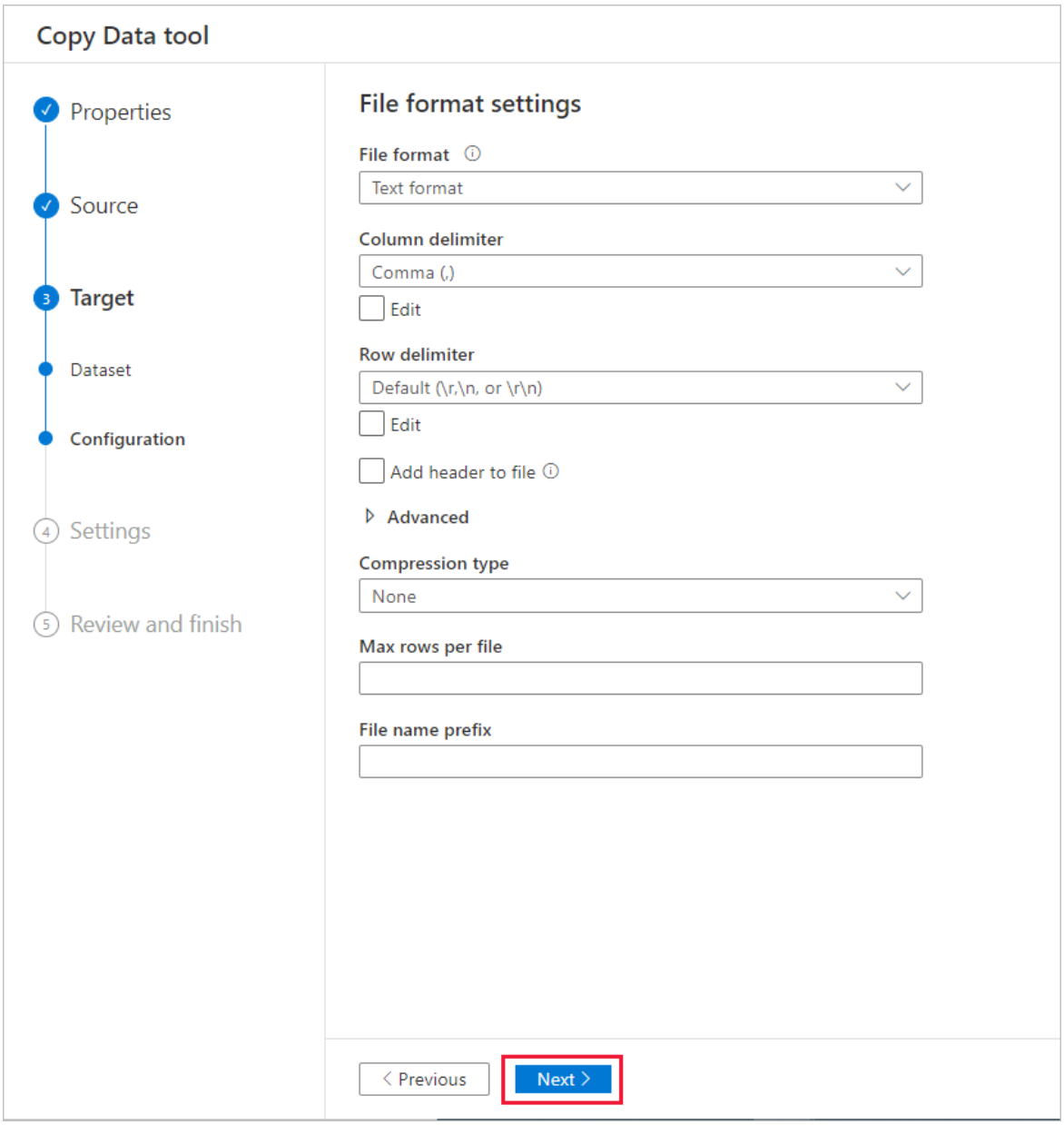
   b. Select **Create** to create the connection.



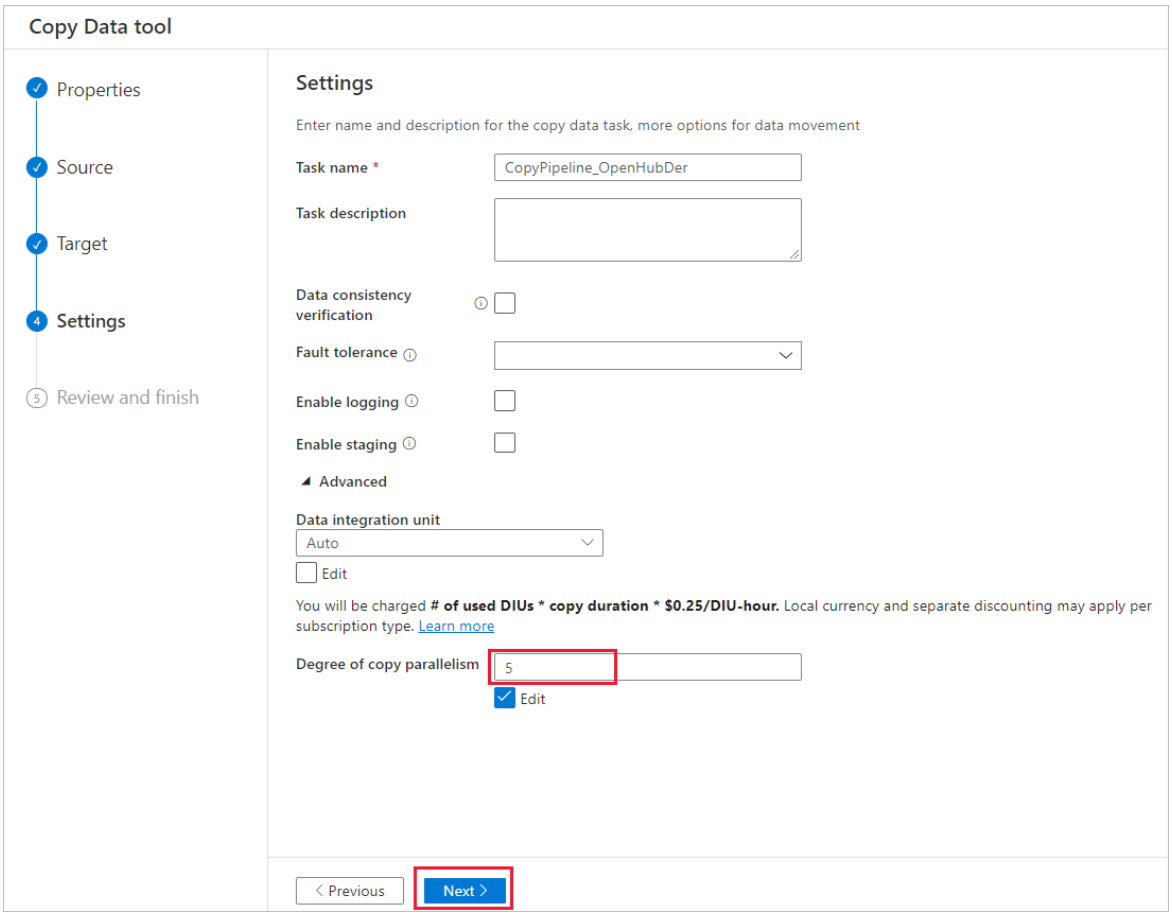
8. On the **Destination data store** page, select the newly created connection in the **Connection** section, and enter **copyfromopenhub** as the output folder name. Then select **Next**.



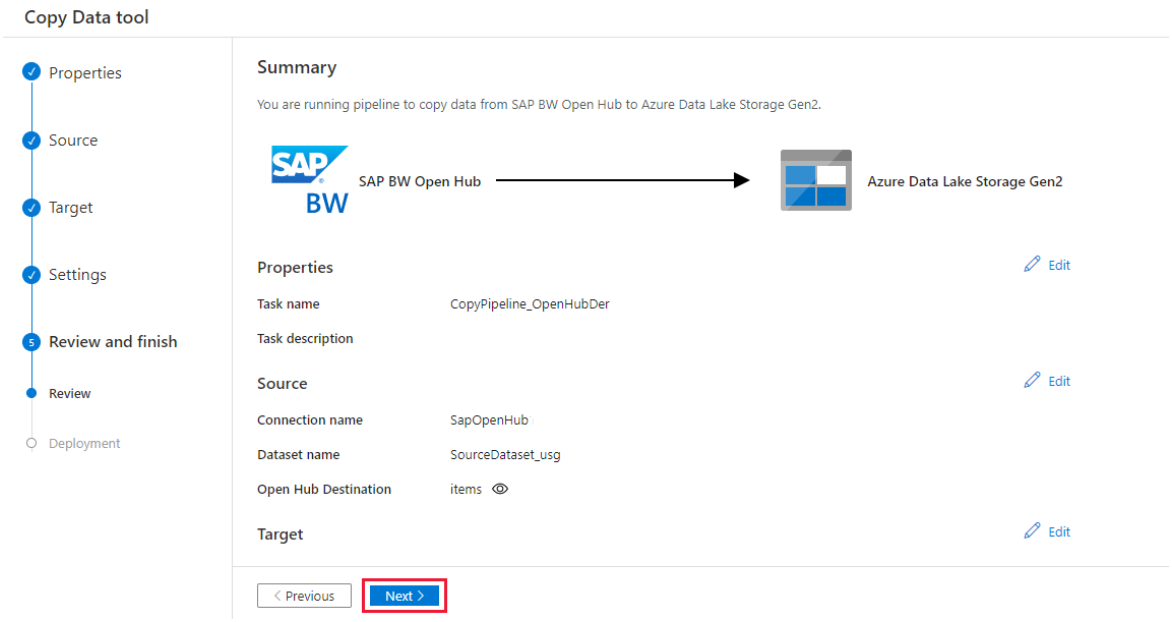
9. On the **File format setting** page, select **Next** to use the default settings.



10. On the **Settings** page, specify a **Task name**, and expand **Advanced**. Enter a value for **Degree of copy parallelism** such as 5 to load from SAP BW in parallel. Then select **Next**.



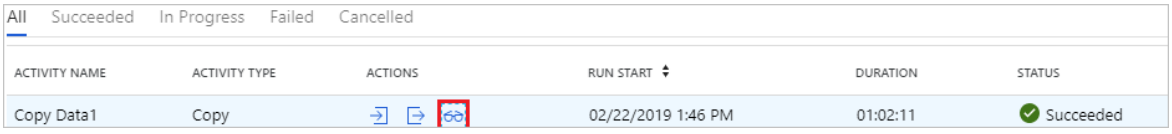
11. On the **Summary** page, review the settings. Then select **Next**.



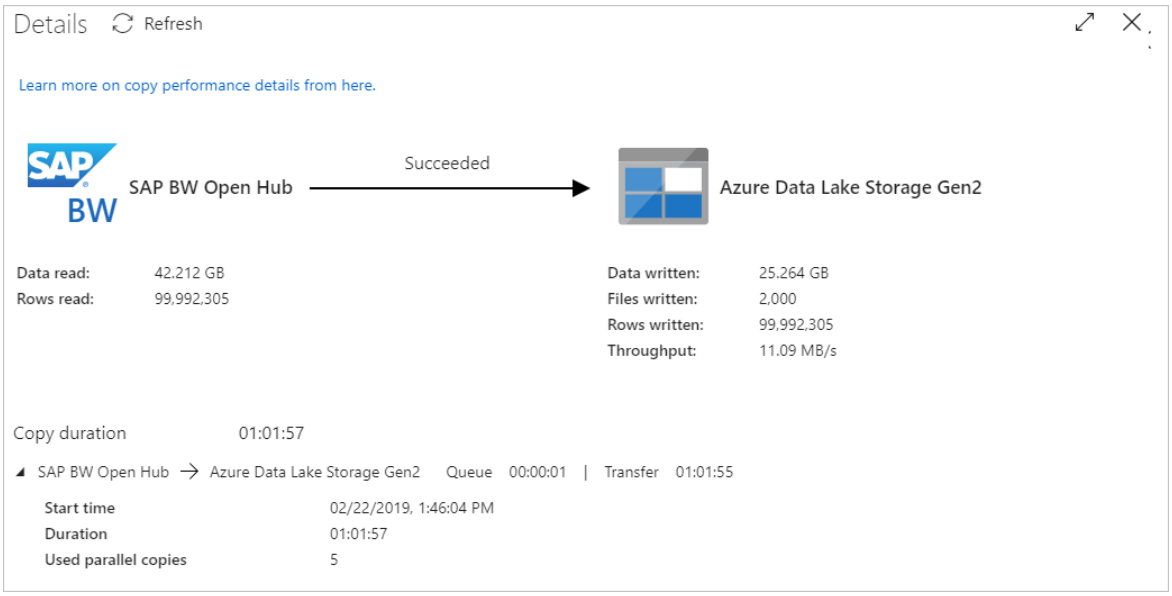
12. On the **Deployment** page, select **Monitor** to monitor the pipeline.

13. Notice that the **Monitor** tab on the left side of the page is automatically selected. You can use links under the **Pipeline name** column in the **Pipeline runs** page to view activity details and to rerun the pipeline.

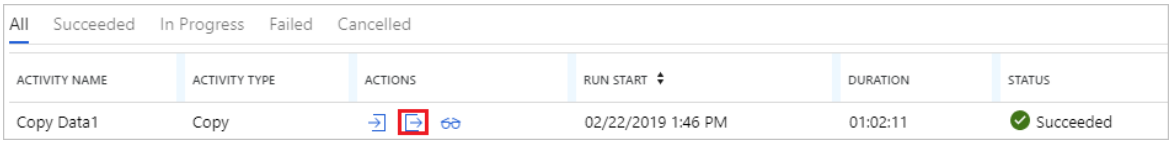
14. To view activity runs that are associated with the pipeline run, select links under the **Pipeline name** column. There's only one activity (copy activity) in the pipeline, so you see only one entry. To switch back to the pipeline-runs view, select the **All pipeline runs** link at the top. Select **Refresh** to refresh the list.

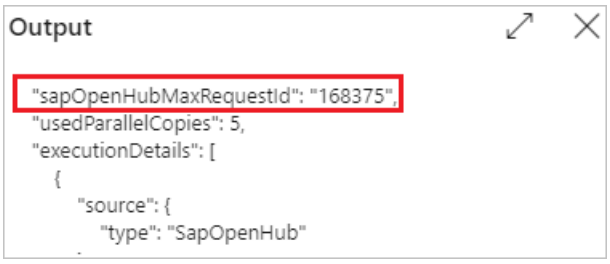


15. To monitor the execution details for each copy activity, select the **Details** link, which is an eyeglasses icon in the same row of each copy activity in the activity-monitoring view. Available details include the data volume copied from the source to the sink, data throughput, execution steps and duration, and configurations used.



16. To view the **maximum Request ID** of each copy activity, go back to the activity-monitoring view and select **Output** in the same row of each copy activity.



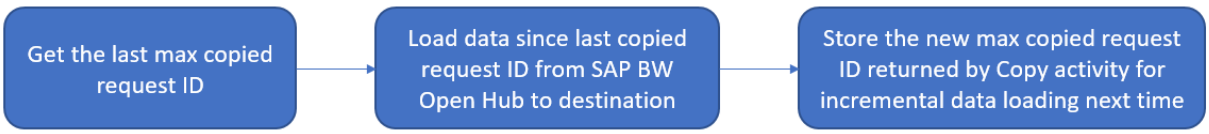


Incremental copy from SAP BW Open Hub

> **TIP**
> See SAP BW Open Hub connector delta extraction flow to learn how the SAP BW Open Hub connector copies incremental data from SAP BW. This article can also help you understand basic connector configuration.

Now, let's continue to configure incremental copy from SAP BW Open Hub.

Incremental copy uses a "high-watermark" mechanism that's based on the **request ID**. That ID is automatically generated in SAP BW Open Hub Destination by the DTP. The following diagram shows this workflow:

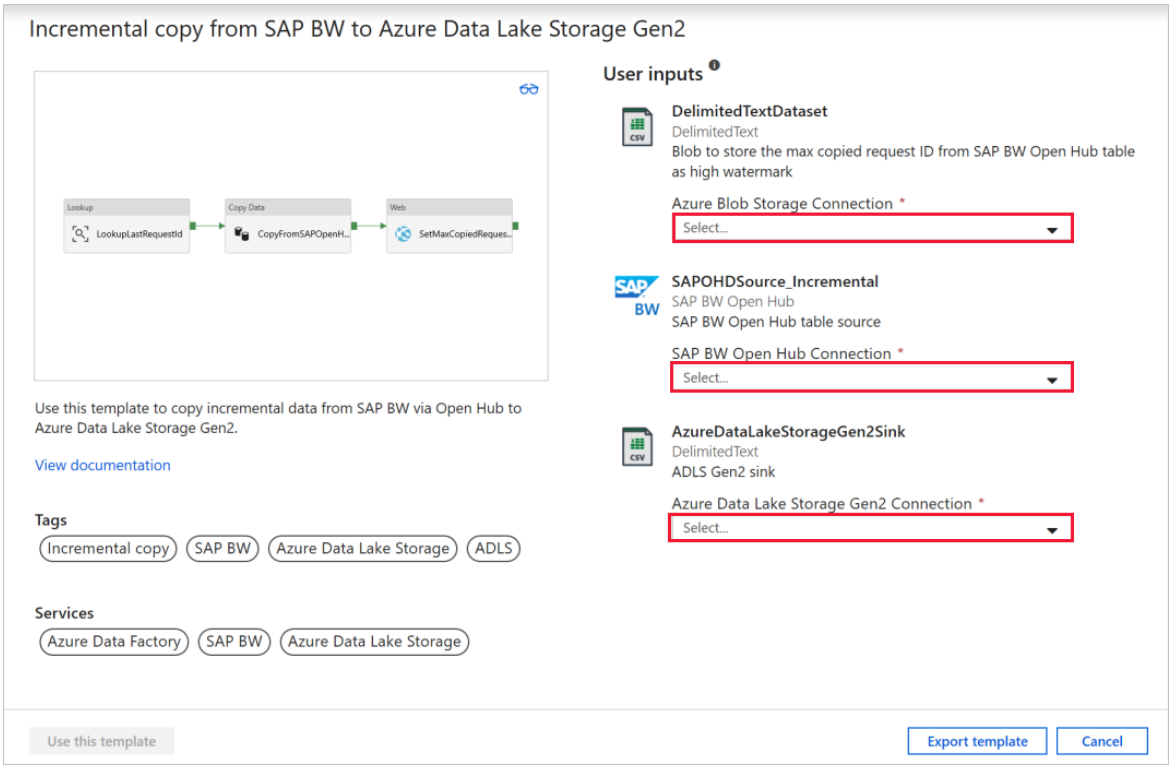


On the home page, select **Pipeline templates** in the **Discover more** section to use the built-in template.

1. Search for **SAP BW** to find and select the **Incremental copy from SAP BW to Azure Data Lake Storage Gen2** template. This template copies data into Azure Data Lake Storage Gen2. You can use a similar workflow to copy to other sink types.

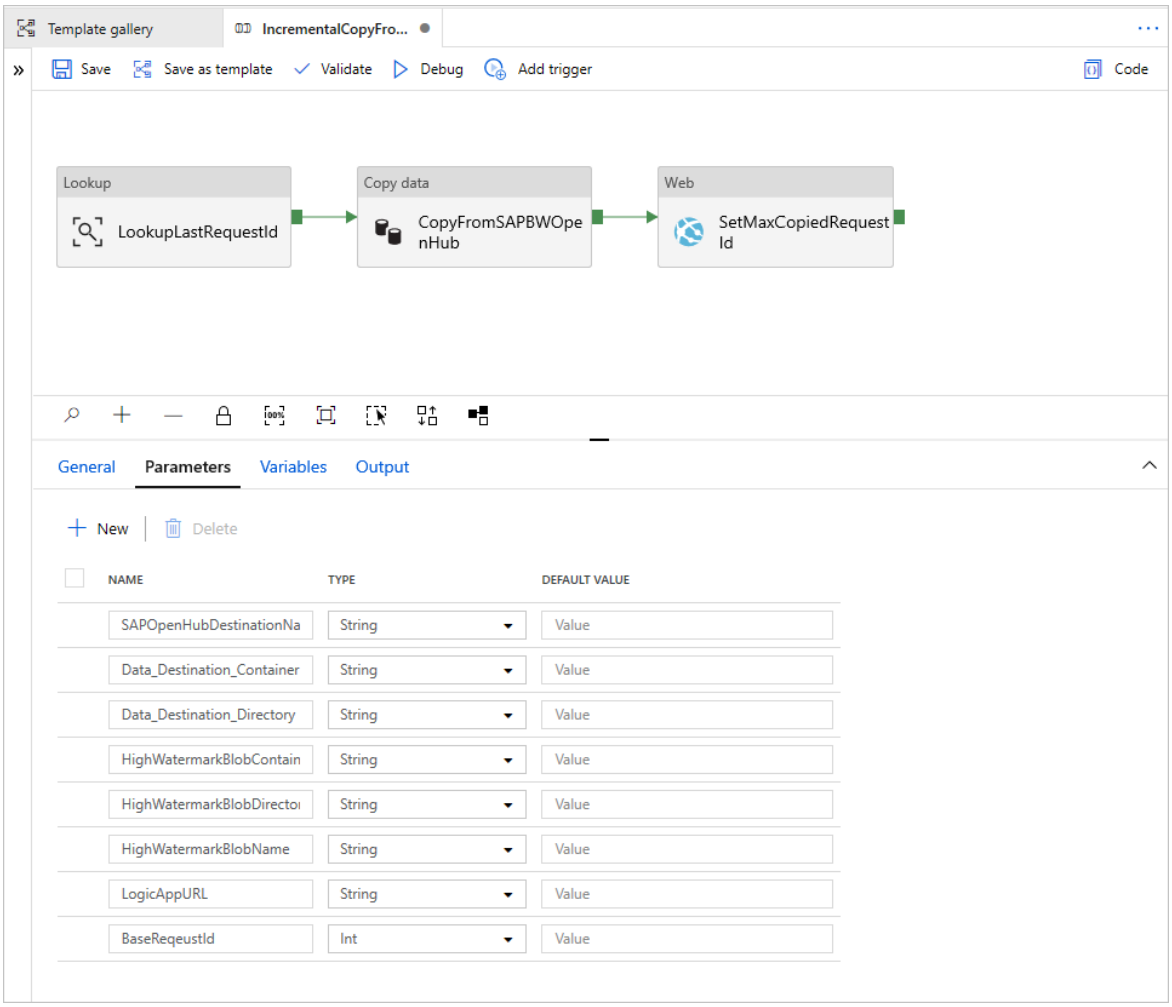
2. On the template's main page, select or create the following three connections, and then select **Use this template** in the lower-right corner of the window.

   - **Azure Blob storage**: In this walkthrough, we use Azure Blob storage to store the high watermark, which is the *max copied request ID*.
   - **SAP BW Open Hub**: This is the source to copy data from. Refer to the previous full-copy walkthrough for detailed configuration.
   - **Azure Data Lake Storage Gen2**: This is the sink to copy data to. Refer to the previous full-copy walkthrough for detailed configuration.



3. This template generates a pipeline with the following three activities and makes them chained on-success: *Lookup*, *Copy Data*, and *Web*.

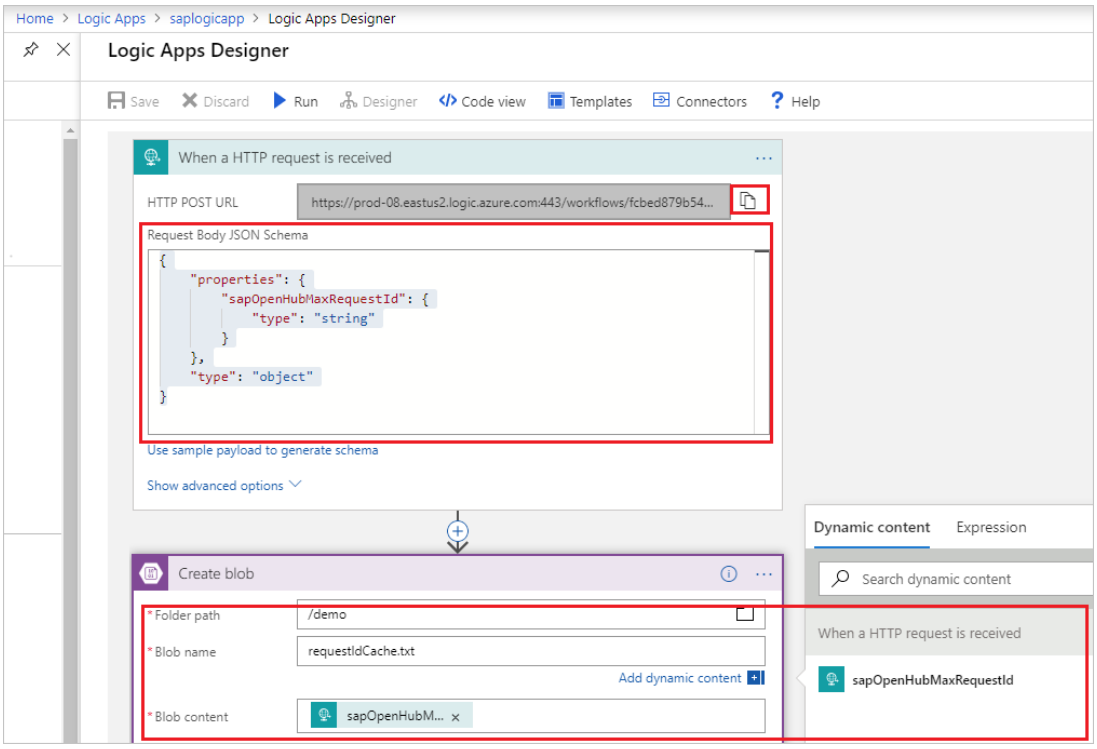
   Go to the pipeline **Parameters** tab. You see all the configurations that you need to provide.



- **SAPOpenHubDestinationName**: Specify the Open Hub table name to copy data from.
- **Data_Destination_Container**: Specify the destination Azure Data Lake Storage Gen2 container to copy data to. If the container doesn't exist, the copy activity creates one during execution.
- **Data_Destination_Directory**: Specify the folder path under the Azure Data Lake Storage Gen2 container to copy data to. If the path doesn't exist, the copy activity creates a path during execution.
- **HighWatermarkBlobContainer**: Specify the container to store the high-watermark value.
- **HighWatermarkBlobDirectory**: Specify the folder path under the container to store the high-watermark value.
- **HighWatermarkBlobName**: Specify the blob name to store the high watermark value, such as `requestIdCache.txt`. In Blob storage, go to the corresponding path of HighWatermarkBlobContainer+HighWatermarkBlobDirectory+HighWatermarkBlobName, such as *container/path/requestIdCache.txt*. Create a blob with content 0.

- **LogicAppURL**: In this template, we use WebActivity to call Azure Logic Apps to set the high-watermark value in Blob storage. Or, you can use Azure SQL Database to store it. Use a stored procedure activity to update the value.

  You must first create a logic app, as the following image shows. Then, paste in the **HTTP POST URL**.



a. Go to the Azure portal. Select a new **Logic Apps** service. Select **+Blank Logic App** to go to **Logic Apps Designer**.

b. Create a trigger of **When an HTTP request is received**. Specify the HTTP request body as follows:

```
{
   "properties": {
      "sapOpenHubMaxRequestId": {
         "type": "string"
      }
   },
   "type": "object"
}
```

c. Add a **Create blob** action. For **Folder path** and **Blob name**, use the same values that you configured previously in *HighWatermarkBlobContainer+HighWatermarkBlobDirectory* and *HighWatermarkBlobName*.

d. Select **Save**. Then, copy the value of **HTTP POST URL** to use in the pipeline.

4. After you provide the pipeline parameters, select **Debug** > **Finish** to invoke a run to validate the configuration. Or, select **Publish** to publish all the changes, and then select **Add trigger** to execute a run.

SAP BW Open Hub Destination configurations

This section introduces configuration of the SAP BW side to use the SAP BW Open Hub connector to copy data.

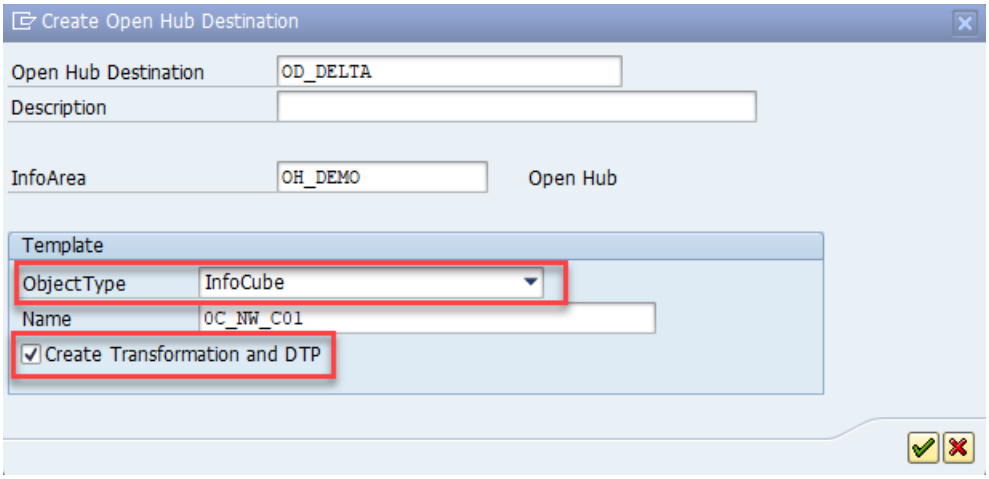
Configure delta extraction in SAP BW

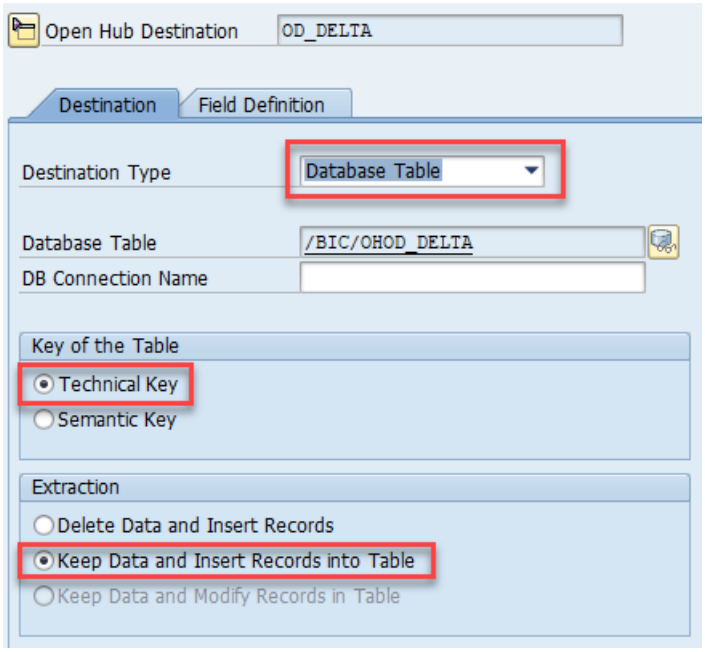
If you need both historical copy and incremental copy or only incremental copy, configure delta extraction in SAP BW.

1. Create the Open Hub Destination. You can create the OHD in SAP Transaction RSA1, which automatically creates the required transformation and data-transfer process. Use the following settings:

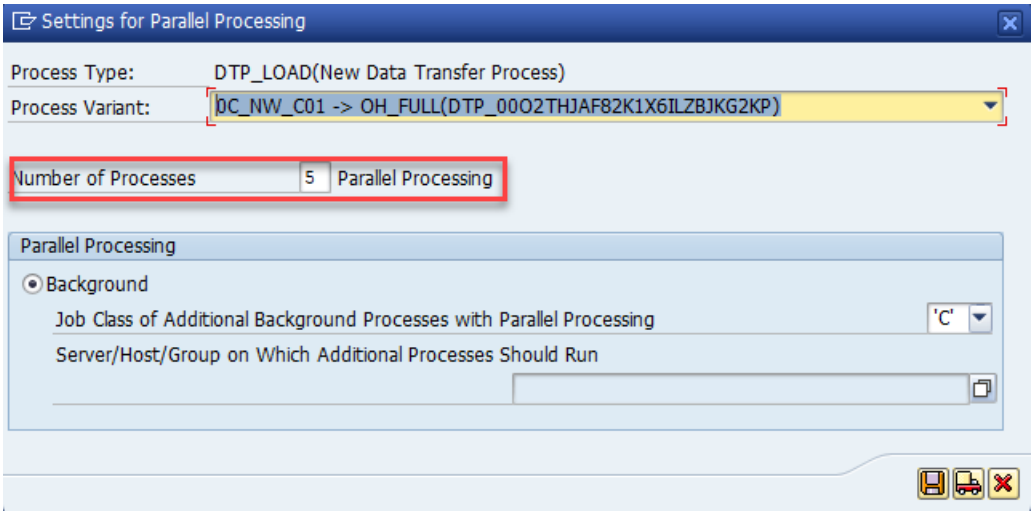
   - **ObjectType**: You can use any object type. Here, we use **InfoCube** as an example.

- **Destination Type**: Select **Database Table**.
- **Key of the Table**: Select **Technical Key**.
- **Extraction**: Select **Keep Data and Insert Records into Table**.





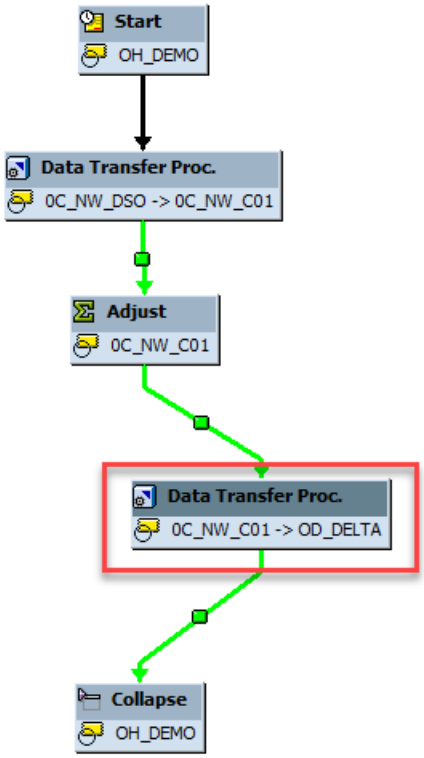
You might increase the number of parallel running SAP work processes for the DTP:



2. Schedule the DTP in process chains.

   A delta DTP for a cube only works if the necessary rows haven't been compressed. Make sure that BW cube compression isn't running before the DTP to the Open Hub table. The easiest way to do this is to integrate the DTP into your existing process chains. In the following example, the DTP (to the OHD) is

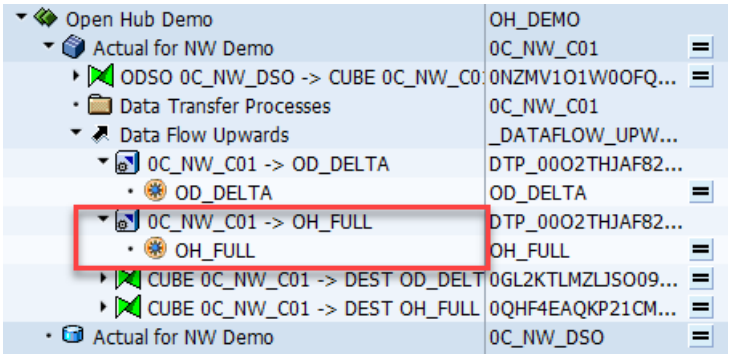
inserted into the process chain between the *Adjust* (aggregate rollup) and *Collapse* (cube compression) steps.



Configure full extraction in SAP BW

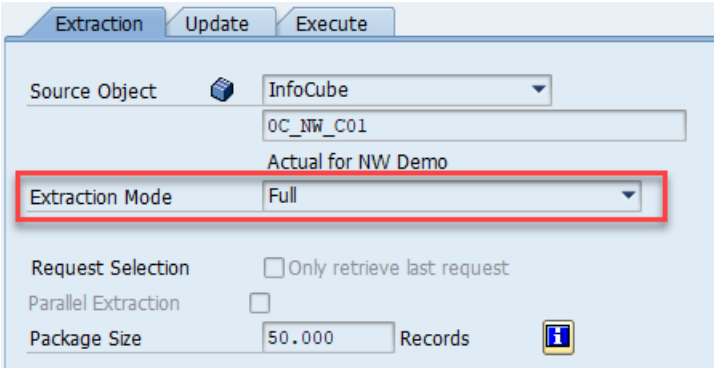
In addition to delta extraction, you might want a full extraction of the same SAP BW InfoProvider. This usually applies if you want to do full copy but not incremental, or you want to resync delta extraction.

You can't have more than one DTP for the same OHD. So, you must create an additional OHD before delta extraction.



For a full load OHD, choose different options than for delta extraction:

- In OHD: Set the **Extraction** option to **Delete Data and Insert Records**. Otherwise, data will be extracted many times when you repeat the DTP in a BW process chain.
- In the DTP: Set **Extraction Mode** to **Full**. You must change the automatically created DTP from **Delta** to **Full** immediately after the OHD is created, as this image shows:



- In the BW Open Hub connector: Turn off **Exclude last request**. Otherwise, nothing will be extracted.

You typically run the full DTP manually. Or, you can create a process chain for the full DTP. It's typically a separate chain that's independent of your existing process chains. In either case, *make sure that the DTP is finished before you start the extraction by using copy*. Otherwise, only partial data will be copied.

Run delta extraction the first time

The first delta extraction is technically a *full extraction*. By default, the SAP BW Open Hub connector excludes the last request when it copies data. For the first delta extraction, no data is extracted by the copy activity until a subsequent DTP generates delta data in the table with a separate request ID. There are two ways to avoid this scenario:

- Turn off the **Exclude last request** option for the first delta extraction. Make sure that the first delta DTP is finished before you start the delta extraction the first time.
- Use the procedure for resyncing the delta extraction, as described in the next section.

Resync delta extraction

The following scenarios change the data in SAP BW cubes but are not considered by the delta DTP:

- SAP BW selective deletion (of rows by using any filter condition)
- SAP BW request deletion (of faulty requests)

An SAP Open Hub Destination isn't a data-mart-controlled data target (in all SAP BW support packages since 2015). So, you can delete data from a cube without changing the data in the OHD. You must then resync the data of the cube with the service:

1. Run a full extraction in the service (by using a full DTP in SAP).
2. Delete all rows in the Open Hub table for the delta DTP.
3. Set the status of the delta DTP to **Fetched**.

After this, all subsequent delta DTPs and delta extractions work as expected.

To set the status of the delta DTP to **Fetched**, you can use the following option to run the delta DTP manually:

No Data Transfer; Delta Status in Source: Fetched

Next steps

Learn about SAP BW Open Hub connector support:

SAP Business Warehouse Open Hub connector

Copy data from SAP Business Warehouse using Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from an SAP Business Warehouse (BW). It builds on the copy activity overview article that presents a general overview of copy activity.

> **TIP**
> To learn the service's overall support on SAP data integration scenario, see SAP data integration using Azure Data Factory whitepaper with detailed introduction on each SAP connector, comparison and guidance.

Supported capabilities

This SAP Business Warehouse connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	②
Lookup activity	②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

Specifically, this SAP Business Warehouse connector supports:

- SAP Business Warehouse **version 7.x**.
- Copying data from **InfoCubes and QueryCubes** (including BEx queries) using MDX queries.
- Copying data using basic authentication.

> **NOTE**
> The SAP Business Warehouse connector does not currently support parameters with MDX. If filtering with MDX parameters is required you can consider using the alternative SAP Open Hub connector instead.

Prerequisites

To use this SAP Business Warehouse connector, you need to:

- Set up a Self-hosted Integration Runtime. See Self-hosted Integration Runtime article for details.
- Install the **SAP NetWeaver library** on the Integration Runtime machine. You can get the SAP Netweaver library from your SAP administrator, or directly from the SAP Software Download Center. Search for the **SAP Note #1025361** to get the download location for the most recent version. Make sure that you pick the **64-**

bit SAP NetWeaver library which matches your Integration Runtime installation. Then install all files included in the SAP NetWeaver RFC SDK according to the SAP Note. The SAP NetWeaver library is also included in the SAP Client Tools installation.

> **TIP**
>
> To troubleshoot connectivity issue to SAP BW, make sure:
>
> - All dependency libraries extracted from the NetWeaver RFC SDK are in place in the %windir%\system32 folder. Usually it has icudt34.dll, icuin34.dll, icuuc34.dll, libicudecnumber.dll, librfc32.dll, libsapucum.dll, sapcrypto.dll, sapcryto_old.dll, sapnwrfc.dll.
> - The needed ports used to connect to SAP Server are enabled on the Self-hosted IR machine, which usually are port 3300 and 3201.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

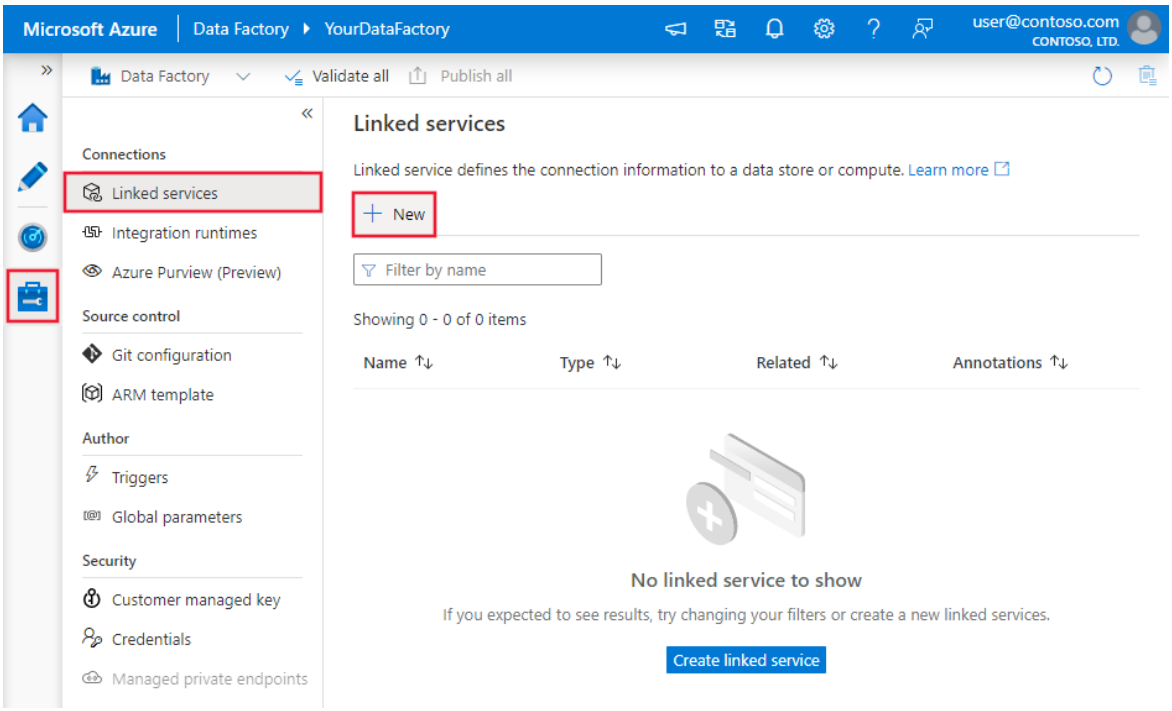
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to SAP BW using UI

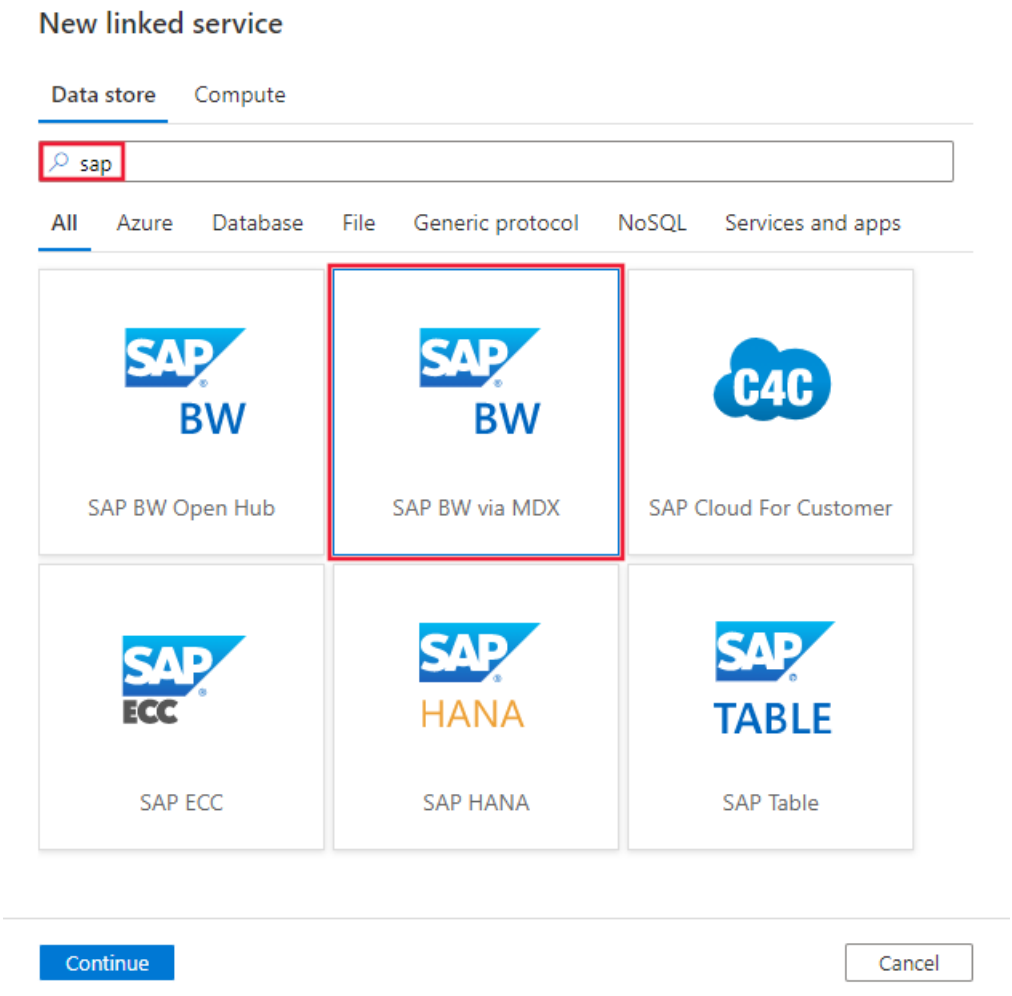
Use the following steps to create a linked service to SAP BW in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

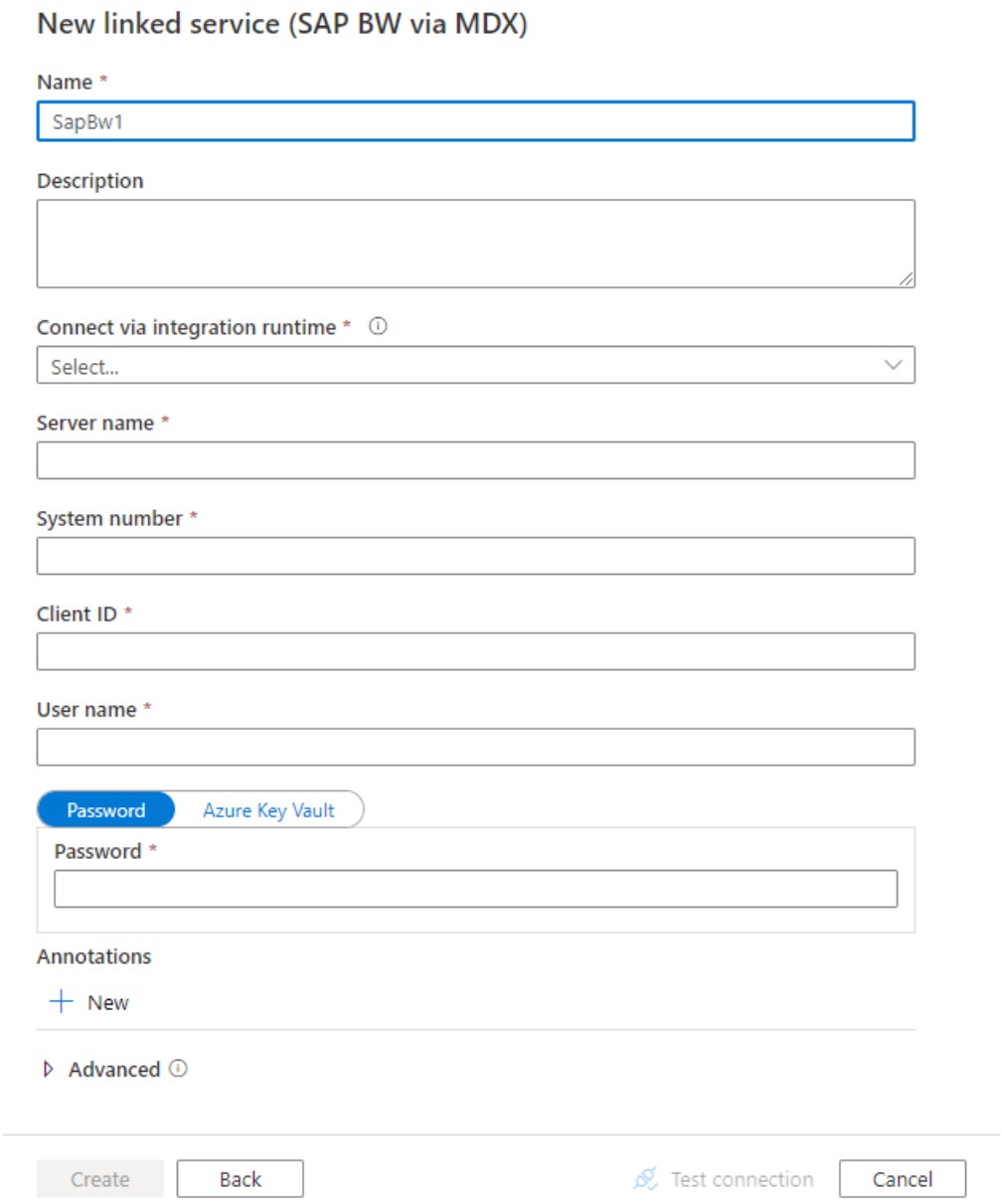
   - Azure Data Factory
   - Azure Synapse



2. Search for SAP and select the SAP BW via MDX connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to SAP Business Warehouse connector.

Linked service properties

The following properties are supported for SAP Business Warehouse (BW) linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **SapBw**	Yes
server	Name of the server on which the SAP BW instance resides.	Yes
systemNumber	System number of the SAP BW system. Allowed value: two-digit decimal number represented as a string.	Yes

PROPERTY	DESCRIPTION	REQUIRED
clientId	Client ID of the client in the SAP W system. Allowed value: three-digit decimal number represented as a string.	Yes
userName	Name of the user who has access to the SAP server.	Yes
password	Password for the user. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. A Self-hosted Integration Runtime is required as mentioned in Prerequisites.	Yes

Example:

```
{
    "name": "SapBwLinkedService",
    "properties": {
        "type": "SapBw",
        "typeProperties": {
            "server": "<server name>",
            "systemNumber": "<system number>",
            "clientId": "<client id>",
            "userName": "<SAP user>",
            "password": {
                "type": "SecureString",
                "value": "<Password for SAP user>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by SAP BW dataset.

To copy data from SAP BW, set the type property of the dataset to **SapBwCube**. While there are no type-specific properties supported for the SAP BW dataset of type RelationalTable.

Example:

```
{
    "name": "SAPBWDataset",
    "properties": {
        "type": "SapBwCube",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<SAP BW linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

If you were using `RelationalTable` typed dataset, it is still supported as-is, while you are suggested to use the new one going forward.

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by SAP BW source.

SAP BW as source

To copy data from SAP BW, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **SapBwSource**	Yes
query	Specifies the MDX query to read data from the SAP BW instance.	Yes

Example:

```
"activities":[
    {
        "name": "CopyFromSAPBW",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SAP BW input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SapBwSource",
                "query": "<MDX query for SAP BW>"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

If you were using `RelationalSource` typed source, it is still supported as-is, while you are suggested to use the new one going forward.

Data type mapping for SAP BW

When copying data from SAP BW, the following mappings are used from SAP BW data types to interim data types used internally within the service. See Schema and data type mappings to learn about how copy activity maps the source schema and data type to the sink.

SAP BW DATA TYPE	INTERIM SERVICE DATA TYPE
ACCP	Int
CHAR	String
CLNT	String
CURR	Decimal
CUKY	String
DEC	Decimal
FLTP	Double
INT1	Byte
INT2	Int16

SAP BW DATA TYPE	INTERIM SERVICE DATA TYPE
INT4	Int
LANG	String
LCHR	String
LRAW	Byte[]
PREC	Int16
QUAN	Decimal
RAW	Byte[]
RAWSTRING	Byte[]
STRING	String
UNIT	String
DATS	String
NUMC	String
TIMS	String

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from or to SAP Cloud for Customer (C4C) using Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from/to SAP Cloud for Customer (C4C). It builds on the copy activity overview article that presents a general overview of copy activity.

> **TIP**
> To learn the service's overall support on SAP data integration scenario, see SAP data integration using Azure Data Factory whitepaper with detailed introduction on each SAP connector, comparison and guidance.

Supported capabilities

This SAP Cloud for Customer connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Specifically, this connector enables the service to copy data from/to SAP Cloud for Customer including the SAP Cloud for Sales, SAP Cloud for Service, and SAP Cloud for Social Engagement solutions.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

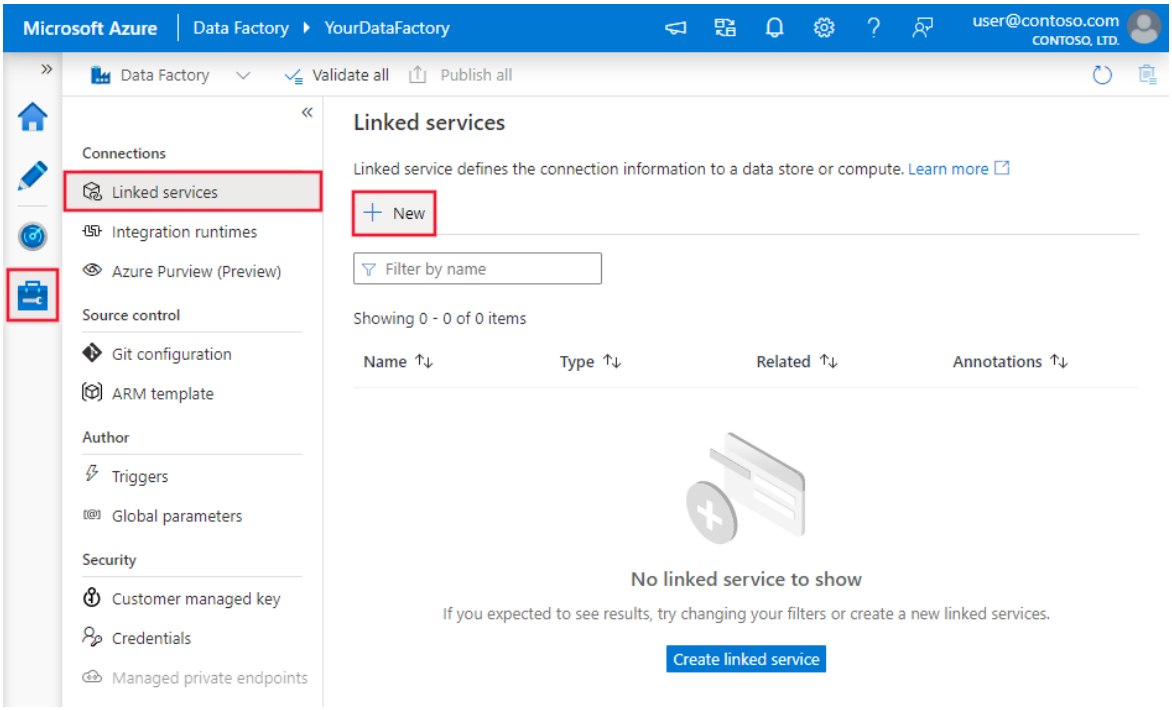
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to SAP Cloud for Customer using UI

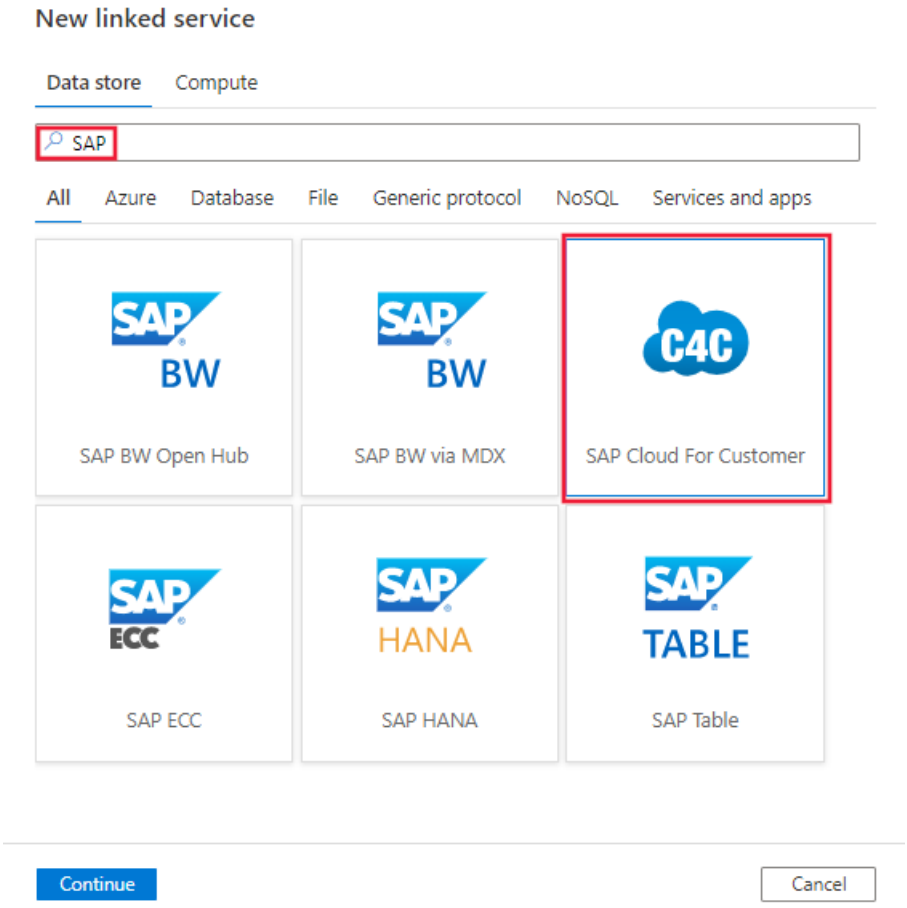
Use the following steps to create a linked service to SAP Cloud for Customer in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

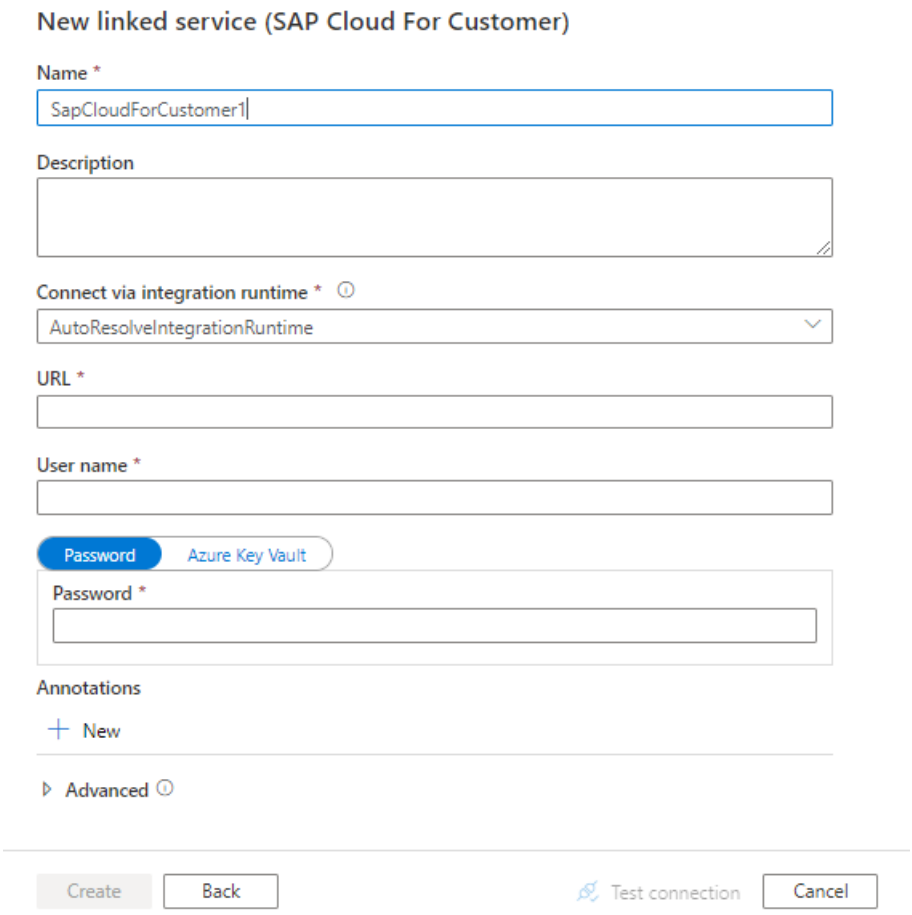
   - Azure Data Factory
   - Azure Synapse



2. Search for SAP and select the SAP Cloud for Customer connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to SAP Cloud for Customer connector.

Linked service properties

The following properties are supported for SAP Cloud for Customer linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **SapCloudForCustomer**.	Yes
url	The URL of the SAP C4C OData service.	Yes
username	Specify the user name to connect to the SAP C4C.	Yes
password	Specify the password for the user account you specified for the username. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectVia	The Integration Runtime to be used to connect to the data store. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "SAPC4CLinkedService",
    "properties": {
        "type": "SapCloudForCustomer",
        "typeProperties": {
            "url": "https://<tenantname>.crm.ondemand.com/sap/c4c/odata/v1/c4codata/" ,
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by SAP Cloud for Customer dataset.

To copy data from SAP Cloud for Customer, set the type property of the dataset to **SapCloudForCustomerResource**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **SapCloudForCustomerResource**	Yes
path	Specify path of the SAP C4C OData entity.	Yes

Example:

```
{
    "name": "SAPC4CDataset",
    "properties": {
        "type": "SapCloudForCustomerResource",
        "typeProperties": {
            "path": "<path e.g. LeadCollection>"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<SAP C4C linked service>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by SAP Cloud for Customer source.

SAP C4C as source

To copy data from SAP Cloud for Customer, set the source type in the copy activity to **SapCloudForCustomerSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **SapCloudForCustomerSource**	Yes
query	Specify the custom OData query to read data.	No
httpRequestTimeout	The timeout (the **TimeSpan** value) for the HTTP request to get a response. This value is the timeout to get a response, not the timeout to read response data. If not specified, the default value is **00:30:00** (30 minutes).	No

Sample query to get data for a specific day:

```
"query": "$filter=CreatedOn ge datetimeoffset'2017-07-31T10:02:06.4202620Z' and CreatedOn le
datetimeoffset'2017-08-01T10:02:06.4202620Z'"
```

Example:

```
"activities":[
    {
        "name": "CopyFromSAPC4C",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SAP C4C input dataset>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SapCloudForCustomerSource",
                "query": "<custom query e.g. $top=10>"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

SAP C4C as sink

To copy data to SAP Cloud for Customer, set the sink type in the copy activity to **SapCloudForCustomerSink**. The following properties are supported in the copy activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **SapCloudForCustomerSink**	Yes
writeBehavior	The write behavior of the operation. Could be “Insert”, “Update”.	No. Default “Insert”.
writeBatchSize	The batch size of write operation. The batch size to get best performance may be different for different table or server.	No. Default 10.
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyToSapC4c",
        "type": "Copy",
        "inputs": [{
            "type": "DatasetReference",
            "referenceName": "<dataset type>"
        }],
        "outputs": [{
            "type": "DatasetReference",
            "referenceName": "SapC4cDataset"
        }],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "SapCloudForCustomerSink",
                "writeBehavior": "Insert",
                "writeBatchSize": 30
            },
            "parallelCopies": 10,
            "dataIntegrationUnits": 4,
            "enableSkipIncompatibleRow": true,
            "redirectIncompatibleRowSettings": {
                "linkedServiceName": {
                    "referenceName": "ErrorLogBlobLinkedService",
                    "type": "LinkedServiceReference"
                },
                "path": "incompatiblerows"
            }
        }
    }
]
```

Data type mapping for SAP Cloud for Customer

When copying data from SAP Cloud for Customer, the following mappings are used from SAP Cloud for Customer data types to interim data types used internally within the service. See Schema and data type mappings to learn about how copy activity maps the source schema and data type to the sink.

SAP C4C ODATA DATA TYPE	INTERIM SERVICE DATA TYPE
Edm.Binary	Byte[]
Edm.Boolean	Bool
Edm.Byte	Byte[]
Edm.DateTime	DateTime
Edm.Decimal	Decimal
Edm.Double	Double
Edm.Single	Single
Edm.Guid	Guid

SAP C4C ODATA DATA TYPE	INTERIM SERVICE DATA TYPE
Edm.Int16	Int16
Edm.Int32	Int32
Edm.Int64	Int64
Edm.SByte	Int16
Edm.String	String
Edm.Time	TimeSpan
Edm.DateTimeOffset	DateTimeOffset

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from SAP ECC using Azure Data Factory or Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the copy activity in Azure Data Factory to copy data from SAP Enterprise Central Component (ECC). For more information, see Copy activity overview.

> **TIP**
> To learn the overall support on SAP data integration scenario, see SAP data integration using Azure Data Factory whitepaper with detailed introduction on each SAP connector, comparison and guidance.

Supported capabilities

This SAP ECC connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

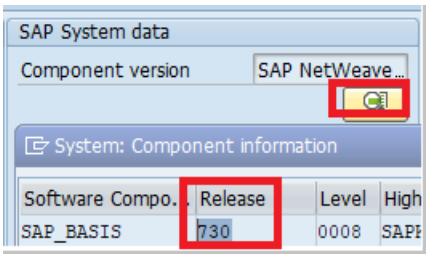
For a list of data stores that are supported as sources or sinks, see the Supported data stores table.

Specifically, this SAP ECC connector supports:

- Copying data from SAP ECC on SAP NetWeaver version 7.0 and later.
- Copying data from any objects exposed by SAP ECC OData services, such as:
  - SAP tables or views.
  - Business Application Programming Interface [BAPI] objects.
  - Data extractors.
  - Data or intermediate documents (IDOCs) sent to SAP Process Integration (PI) that can be received as OData via relative adapters.
- Copying data by using basic authentication.

The version 7.0 or later refers to SAP NetWeaver version instead of SAP ECC version. For example,SAP ECC 6.0 EHP 7 in general has NetWeaver version >=7.4. In case you are unsure about your environment, here are the steps to confirm the version from your SAP system:

1. Use SAP GUI to connect to the SAP System.
2. Go to **System** -> **Status**.
3. Check the release of the SAP_BASIS, ensure it is equal to or larger than 701.



> **TIP**
> To copy data from SAP ECC via an SAP table or view, use the SAP table connector, which is faster and more scalable.

Prerequisites

To use this SAP ECC connector, you need to expose the SAP ECC entities via OData services through SAP Gateway. More specifically:

- **Set up SAP Gateway**. For servers with SAP NetWeaver versions later than 7.4, SAP Gateway is already installed. For earlier versions, you must install the embedded SAP Gateway or the SAP Gateway hub system before exposing SAP ECC data through OData services. To set up SAP Gateway, see the installation guide.
- **Activate and configure the SAP OData service**. You can activate the OData service through TCODE SICF in seconds. You can also configure which objects need to be exposed. For more information, see the step-by-step guidance.

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

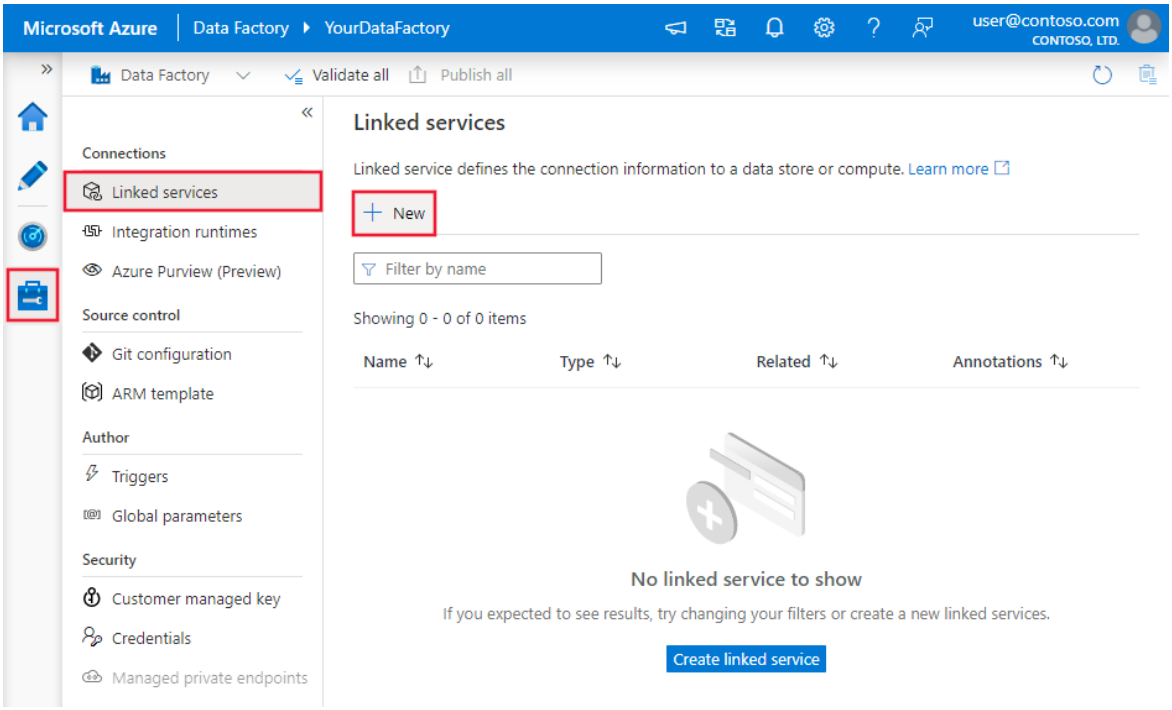
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to SAP ECC using UI

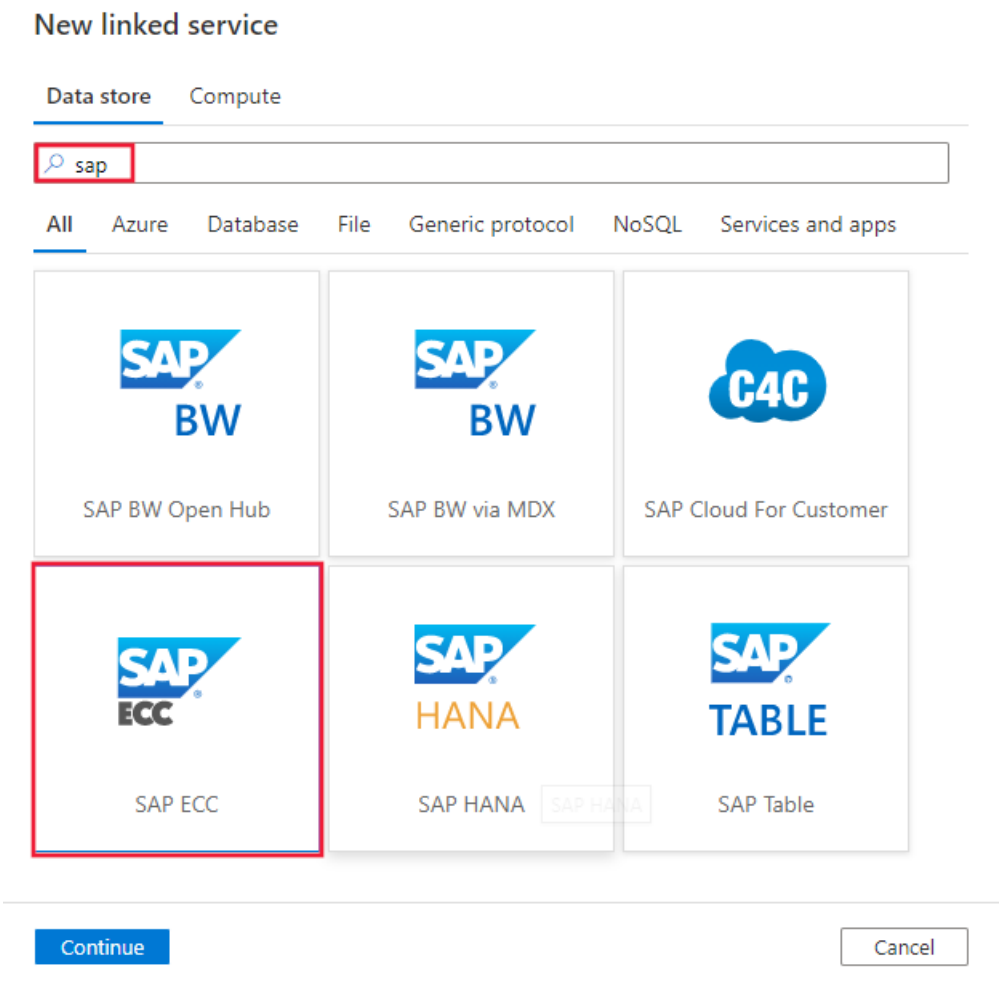
Use the following steps to create a linked service to SAP ECC in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

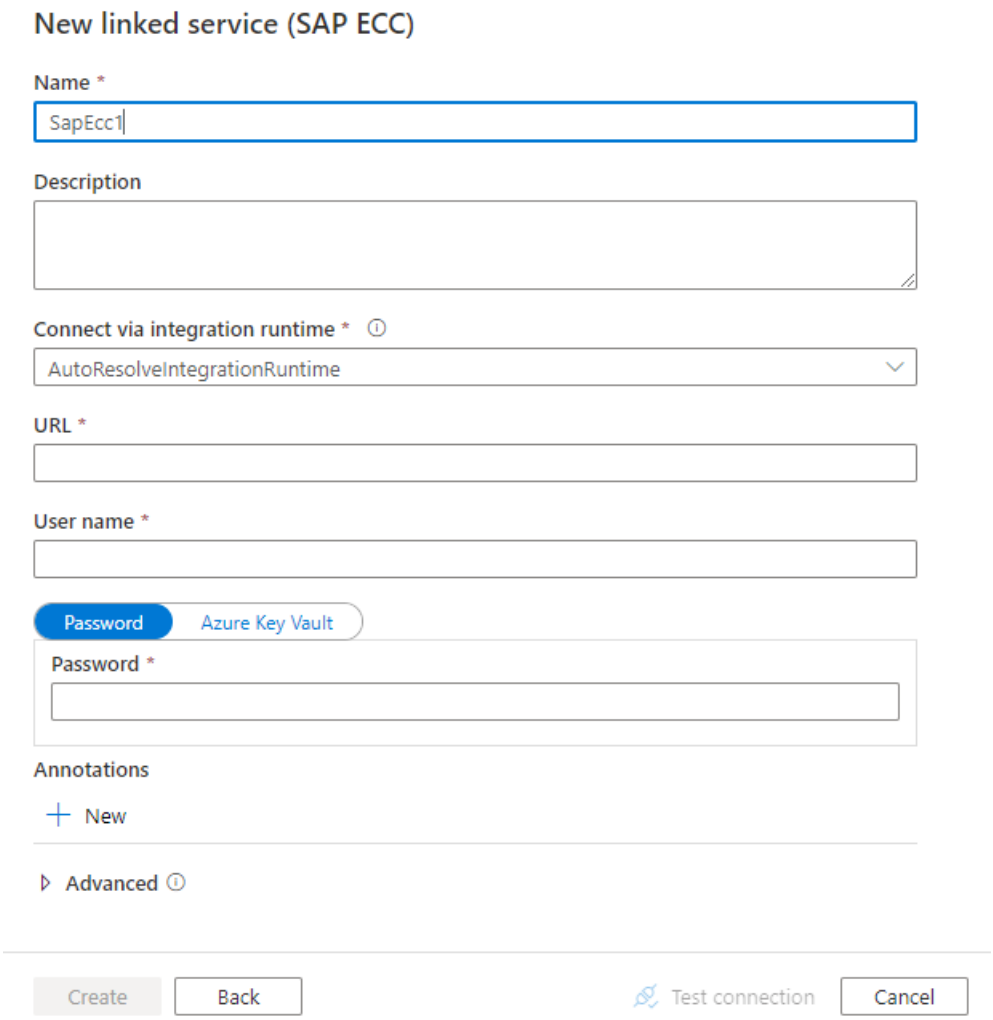
   - Azure Data Factory
   - Azure Synapse



2. Search for SAP and select the SAP ECC connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define the entities specific to the SAP ECC connector.

Linked service properties

The following properties are supported for the SAP ECC linked service:

PROPERTY	DESCRIPTION	REQUIRED
`type`	The `type` property must be set to `SapEcc`.	Yes
`url`	The URL of the SAP ECC OData service.	Yes
`username`	The username used to connect to SAP ECC.	No
`password`	The plaintext password used to connect to SAP ECC.	No

PROPERTY	DESCRIPTION	REQUIRED
`connectVia`	The integration runtime to be used to connect to the data store. Learn more from Prerequisites section. If you don't specify a runtime, the default Azure integration runtime is used.	No

Example

```
{
    "name": "SapECCLinkedService",
    "properties": {
        "type": "SapEcc",
        "typeProperties": {
            "url": "<SAP ECC OData URL, e.g.,
http://eccsvrname:8000/sap/opu/odata/sap/zgw100_dd02l_so_srv/>",
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        }
    },
    "connectVia": {
        "referenceName": "<name of integration runtime>",
        "type": "IntegrationRuntimeReference"
    }
}
```

Dataset properties

For a full list of the sections and properties available for defining datasets, see Datasets. The following section provides a list of the properties supported by the SAP ECC dataset.

To copy data from SAP ECC, set the `type` property of the dataset to `SapEccResource`.

The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
`path`	Path of the SAP ECC OData entity.	Yes

Example

```
{
    "name": "SapEccDataset",
    "properties": {
        "type": "SapEccResource",
        "typeProperties": {
            "path": "<entity path, e.g., dd04tentitySet>"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<SAP ECC linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of the sections and properties available for defining activities, see Pipelines. The following section provides a list of the properties supported by the SAP ECC source.

SAP ECC as a source

To copy data from SAP ECC, set the `type` property in the `source` section of the copy activity to `SapEccSource`.

The following properties are supported in the copy activity's `source` section:

PROPERTY	DESCRIPTION	REQUIRED
`type`	The `type` property of the copy activity's `source` section must be set to `SapEccSource`.	Yes

`query`	The OData query options to filter the data. For example:
	`"$select=Name,Description&$top=10"`
	The SAP ECC connector copies data from the combined URL:
	`<URL specified in the linked service>/<path specified in the dataset>?<query specified in the copy activity's source section>`
	For more information, see OData URL components.
		No

`sapDataColumnDelimiter`	The single character that is used as delimiter passed to SAP RFC to split the output data.	No
`httpRequestTimeout`	The timeout (the **TimeSpan** value) for the HTTP request to get a response. This value is the timeout to get a response, not the timeout to read response data. If not specified, the default value is **00:30:00** (30 minutes).	No

Example

```
"activities":[
    {
        "name": "CopyFromSAPECC",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SAP ECC input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SapEccSource",
                "query": "$top=10"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Data type mappings for SAP ECC

When you're copying data from SAP ECC, the following mappings are used from OData data types for SAP ECC data to interim data types the service uses internally. To learn how the copy activity maps the source schema and data type to the sink, see Schema and data type mappings.

ODATA DATA TYPE	INTERIM SERVICE DATA TYPE
`Edm.Binary`	`String`
`Edm.Boolean`	`Bool`
`Edm.Byte`	`String`
`Edm.DateTime`	`DateTime`
`Edm.Decimal`	`Decimal`
`Edm.Double`	`Double`
`Edm.Single`	`Single`
`Edm.Guid`	`String`
`Edm.Int16`	`Int16`
`Edm.Int32`	`Int32`

ODATA DATA TYPE	INTERIM SERVICE DATA TYPE
`Edm.Int64`	`Int64`
`Edm.SByte`	`Int16`
`Edm.String`	`String`
`Edm.Time`	`TimeSpan`
`Edm.DateTimeOffset`	`DateTimeOffset`

> **NOTE**
> Complex data types aren't currently supported.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of the data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from SAP HANA using Azure Data Factory or Synapse Analytics

8/19/2022 • 9 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from an SAP HANA database. It builds on the copy activity overview article that presents a general overview of copy activity.

> **TIP**
> To learn about overall support for the SAP data integration scenario, see SAP data integration whitepaper with detailed introduction on each SAP connector, comparison and guidance.

Supported capabilities

This SAP HANA connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	②
Lookup activity	②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores supported as sources/sinks by the copy activity, see the Supported data stores table.

Specifically, this SAP HANA connector supports:

- Copying data from any version of SAP HANA database.
- Copying data from **HANA information models** (such as Analytic and Calculation views) and **Row/Column tables**.
- Copying data using **Basic** or **Windows** authentication.
- Parallel copying from a SAP HANA source. See the Parallel copy from SAP HANA section for details.

> **TIP**
> To copy data **into** SAP HANA data store, use generic ODBC connector. See SAP HANA sink section with details. Note the linked services for SAP HANA connector and ODBC connector are with different type thus cannot be reused.

Prerequisites

To use this SAP HANA connector, you need to:

- Set up a Self-hosted Integration Runtime. See Self-hosted Integration Runtime article for details.
- Install the SAP HANA ODBC driver on the Integration Runtime machine. You can download the SAP HANA ODBC driver from the SAP Software Download Center. Search with the keyword **SAP HANA CLIENT for**

Windows.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

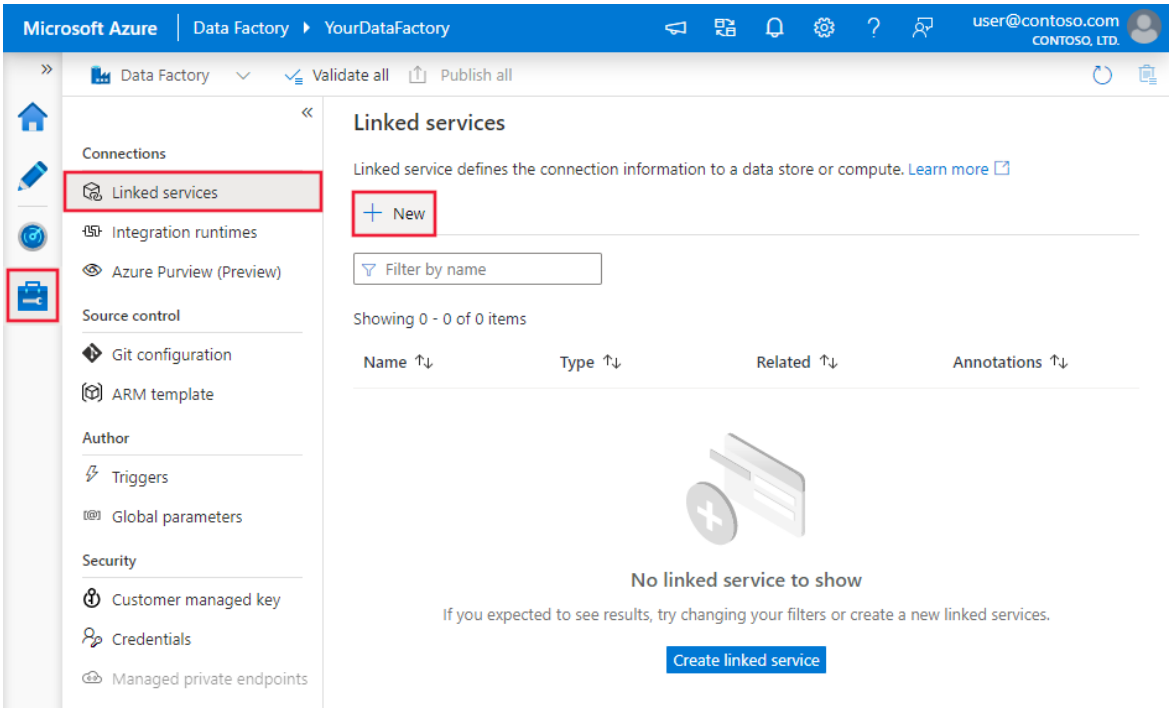
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to SAP HANA using UI

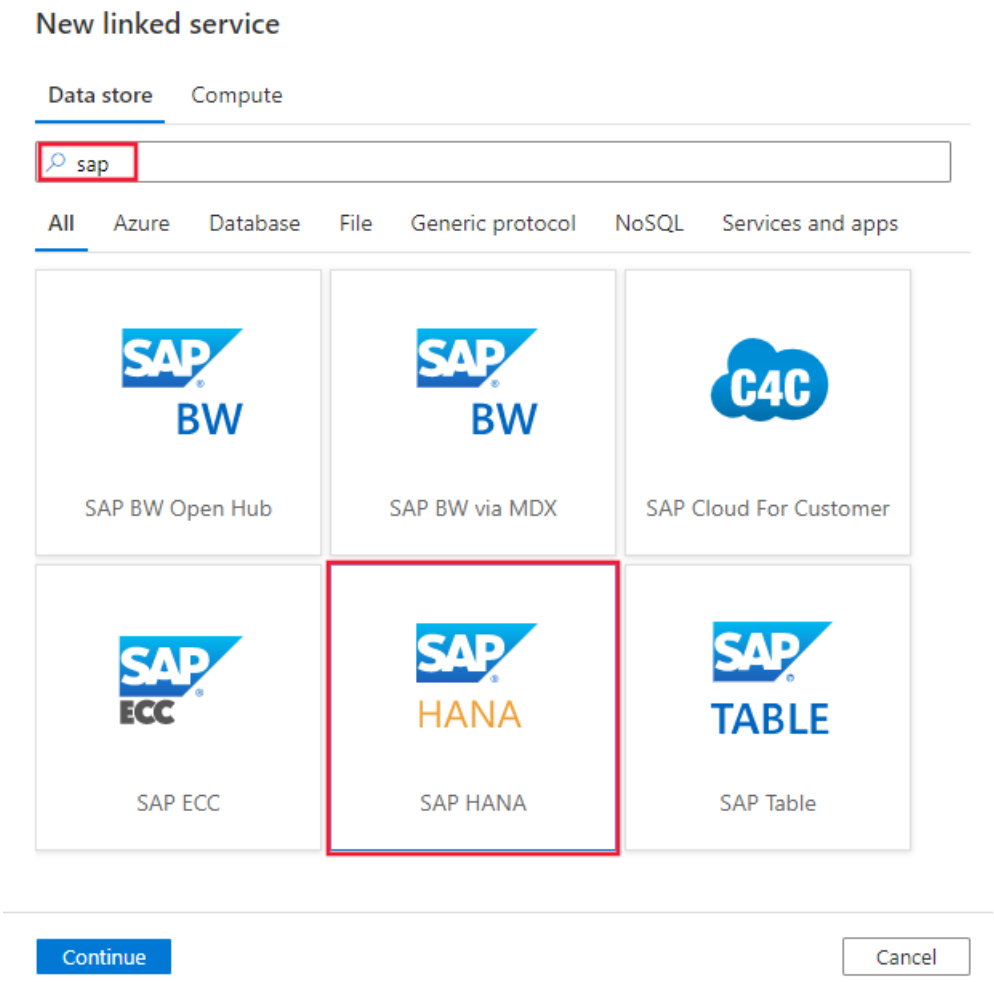
Use the following steps to create a linked service to SAP HANA in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

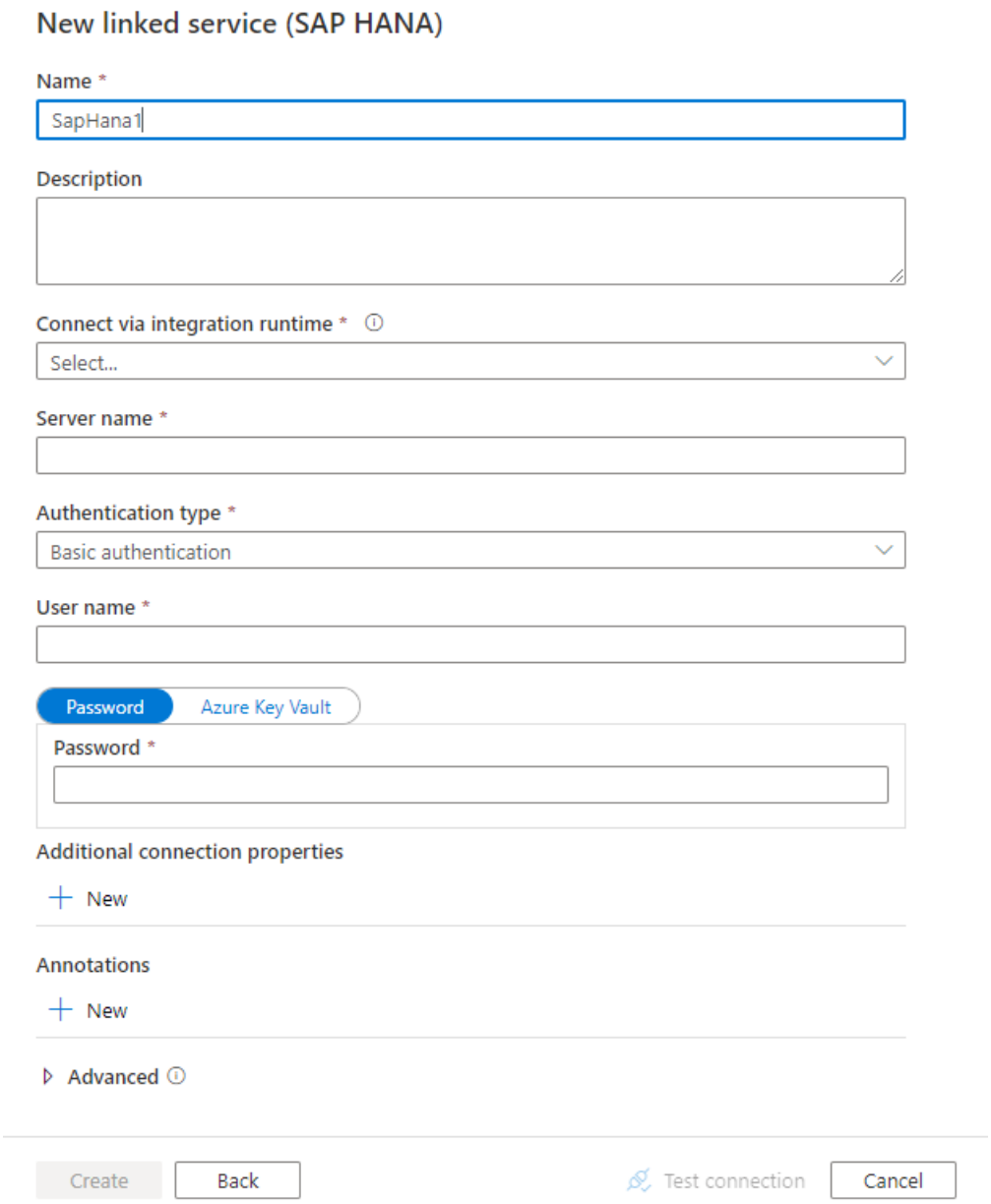
   - Azure Data Factory
   - Azure Synapse



2. Search for SAP and select the SAP HANA connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to SAP HANA connector.

Linked service properties

The following properties are supported for SAP HANA linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **SapHana**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	Specify information that's needed to connect to the SAP HANA by using either **basic authentication** or **Windows authentication**. Refer to the following samples. In connection string, server/port is mandatory (default port is 30015), and username and password is mandatory when using basic authentication. For additional advanced settings, refer to SAP HANA ODBC Connection Properties You can also put password in Azure Key Vault and pull the password configuration out of the connection string. Refer to Store credentials in Azure Key Vault article with more details.	Yes
userName	Specify user name when using Windows authentication. Example: `user@domain.com`	No
password	Specify password for the user account. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
connectVia	The Integration Runtime to be used to connect to the data store. A Self-hosted Integration Runtime is required as mentioned in Prerequisites.	Yes

Example: use basic authentication

```
{
    "name": "SapHanaLinkedService",
    "properties": {
        "type": "SapHana",
        "typeProperties": {
            "connectionString": "SERVERNODE=<server>:<port (optional)>;UID=<userName>;PWD=<Password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: use Windows authentication

```
{
    "name": "SapHanaLinkedService",
    "properties": {
        "type": "SapHana",
        "typeProperties": {
            "connectionString": "SERVERNODE=<server>:<port (optional)>;",
            "userName": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

If you were using SAP HANA linked service with the following payload, it is still supported as-is, while you are suggested to use the new one going forward.

Example:

```
{
    "name": "SapHanaLinkedService",
    "properties": {
        "type": "SapHana",
        "typeProperties": {
            "server": "<server>:<port (optional)>",
            "authenticationType": "Basic",
            "userName": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by SAP HANA dataset.

To copy data from SAP HANA, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **SapHanaTable**	Yes
schema	Name of the schema in the SAP HANA database.	No (if "query" in activity source is specified)

PROPERTY	DESCRIPTION	REQUIRED
table	Name of the table in the SAP HANA database.	No (if "query" in activity source is specified)

Example:

```
{
    "name": "SAPHANADataset",
    "properties": {
        "type": "SapHanaTable",
        "typeProperties": {
            "schema": "<schema name>",
            "table": "<table name>"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<SAP HANA linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

If you were using `RelationalTable` typed dataset, it is still supported as-is, while you are suggested to use the new one going forward.

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by SAP HANA source.

SAP HANA as source

> **TIP**
> To ingest data from SAP HANA efficiently by using data partitioning, learn more from Parallel copy from SAP HANA section.

To copy data from SAP HANA, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **SapHanaSource**	Yes
query	Specifies the SQL query to read data from the SAP HANA instance.	Yes

PROPERTY	DESCRIPTION	REQUIRED
partitionOptions	Specifies the data partitioning options used to ingest data from SAP HANA. Learn more from Parallel copy from SAP HANA section. Allow values are:**None**(default),**PhysicalPartitionsOfTable**, **SapHanaDynamicRange**. Learn more from Parallel copy from SAP HANA section. `PhysicalPartitionsOfTable` can only be used when copying data from a table but not query. When a partition option is enabled (that is, not `None`), the degree of parallelism to concurrently load data from SAP HANA is controlled by the `parallelCopies` setting on the copy activity.	False
partitionSettings	Specify the group of the settings for data partitioning. Apply when partition option is `SapHanaDynamicRange`.	False
partitionColumnName	Specify the name of the source column that will be used by partition for parallel copy. If not specified, the index or the primary key of the table is auto-detected and used as the partition column. Apply when the partition option is `SapHanaDynamicRange`. If you use a query to retrieve the source data, hook `?AdfHanaDynamicRangePartitionCondition` in WHERE clause. See example in Parallel copy from SAP HANA section.	Yes when using `SapHanaDynamicRange` partition.
packetSize	Specifies the network packet size (in Kilobytes) to split data to multiple blocks. If you have large amount of data to copy, increasing packet size can increase reading speed from SAP HANA in most cases. Performance testing is recommended when adjusting the packet size.	No. Default value is 2048 (2MB).

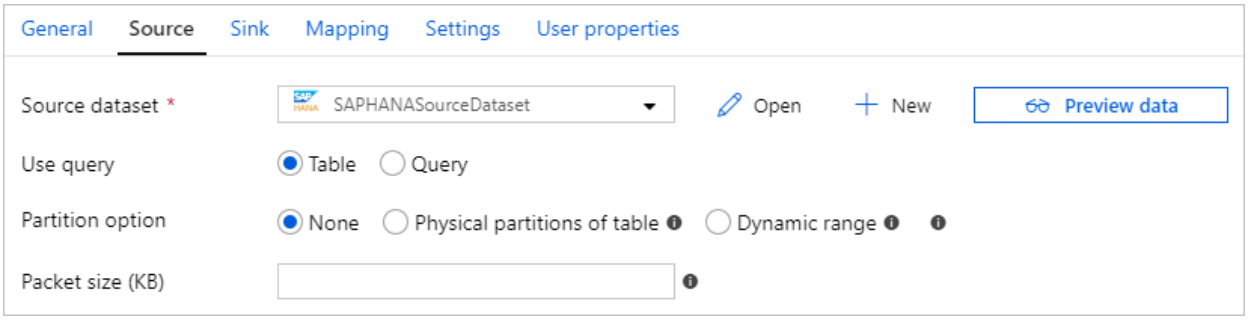
Example:

```
"activities":[
    {
        "name": "CopyFromSAPHANA",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SAP HANA input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SapHanaSource",
                "query": "<SQL query for SAP HANA>"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

If you were using `RelationalSource` typed copy source, it is still supported as-is, while you are suggested to use the new one going forward.

Parallel copy from SAP HANA

The SAP HANA connector provides built-in data partitioning to copy data from SAP HANA in parallel. You can find data partitioning options on the **Source** table of the copy activity.



When you enable partitioned copy, the service runs parallel queries against your SAP HANA source to retrieve data by partitions. The parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your SAP HANA.

You are suggested to enable parallel copy with data partitioning especially when you ingest large amount of data from your SAP HANA. The following are suggested configurations for different scenarios. When copying data into file-based data store, it's recommended to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

SCENARIO	SUGGESTED SETTINGS

SCENARIO	SUGGESTED SETTINGS
Full load from large table.	**Partition option**: Physical partitions of table. During execution, the service automatically detects the physical partition type of the specified SAP HANA table, and choose the corresponding partition strategy: - **Range Partitioning**: Get the partition column and partition ranges defined for the table, then copy the data by range. - **Hash Partitioning**: Use hash partition key as partition column, then partition and copy the data based on ranges calculated by the service. - **Round-Robin Partitioning** or **No Partition**: Use primary key as partition column, then partition and copy the data based on ranges calculated by the service.

Load large amount of data by using a custom query.	**Partition option**: Dynamic range partition.
	Query:
	`SELECT * FROM <TABLENAME> WHERE ? AdfHanaDynamicRangePartitionCondition AND <your_additional_where_clause>`
	.
	Partition column: Specify the column used to apply dynamic range partition.
	
	During execution, the service first calculates the value ranges of the specified partition column, by evenly distributes the rows in a number of buckets according to the number of distinct partition column values the parallel copy setting, then replaces `?AdfHanaDynamicRangePartitionCondition` with filtering the partition column value range for each partition, and sends to SAP HANA.
	
	If you want to use multiple columns as partition column, you can concatenate the values of each column as one column in the query and specify it as the partition column, like
	`SELECT * FROM (SELECT *, CONCAT(<KeyColumn1>, <KeyColumn2>) AS PARTITIONCOLUMN FROM <TABLENAME>) WHERE ?AdfHanaDynamicRangePartitionCondition`
	.

Example: query with physical partitions of a table

```
"source": {
    "type": "SapHanaSource",
    "partitionOption": "PhysicalPartitionsOfTable"
}
```

Example: query with dynamic range partition

```
"source": {
    "type": "SapHanaSource",
    "query":"SELECT * FROM <TABLENAME> WHERE ?AdfHanaDynamicRangePartitionCondition AND
<your_additional_where_clause>",
    "partitionOption": "SapHanaDynamicRange",
    "partitionSettings": {
        "partitionColumnName": "<Partition_column_name>"
    }
}
```

Data type mapping for SAP HANA

When copying data from SAP HANA, the following mappings are used from SAP HANA data types to interim data types used internally within the service. See Schema and data type mappings to learn about how copy activity maps the source schema and data type to the sink.

SAP HANA DATA TYPE	INTERIM SERVICE DATA TYPE
ALPHANUM	String
BIGINT	Int64
BINARY	Byte[]
BINTEXT	String
BLOB	Byte[]
BOOL	Byte
CLOB	String
DATE	DateTime
DECIMAL	Decimal
DOUBLE	Double
FLOAT	Double
INTEGER	Int32
NCLOB	String
NVARCHAR	String
REAL	Single
SECONDDATE	DateTime
SHORTTEXT	String
SMALLDECIMAL	Decimal
SMALLINT	Int16
STGEOMETRYTYPE	Byte[]
STPOINTTYPE	Byte[]
TEXT	String
TIME	TimeSpan

SAP HANA DATA TYPE	INTERIM SERVICE DATA TYPE
TINYINT	Byte
VARCHAR	String
TIMESTAMP	DateTime
VARBINARY	Byte[]

SAP HANA sink

Currently, the SAP HANA connector is not supported as sink, while you can use generic ODBC connector with SAP HANA driver to write data into SAP HANA.

Follow the Prerequisites to set up Self-hosted Integration Runtime and install SAP HANA ODBC driver first. Create an ODBC linked service to connect to your SAP HANA data store as shown in the following example, then create dataset and copy activity sink with ODBC type accordingly. Learn more from ODBC connector article.

```
{
    "name": "SAPHANAViaODBCLinkedService",
    "properties": {
        "type": "Odbc",
        "typeProperties": {
            "connectionString": "Driver={HDBODBC};servernode=<HANA server>.clouddatahub-int.net:30015",
            "authenticationType": "Basic",
            "userName": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from an SAP table using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 11 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the copy activity in Azure Data Factory and Azure Synapse Analytics pipelines to copy data from an SAP table. For more information, see Copy activity overview.

> **TIP**
> To learn the overall support on SAP data integration scenario, see SAP data integration using Azure Data Factory whitepaper with detailed introduction on each SAP connector, comparsion and guidance.

Supported capabilities

This SAP table connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	②
Lookup activity	②

① Azure integration runtime ② Self-hosted integration runtime

For a list of the data stores that are supported as sources or sinks by the copy activity, see the Supported data stores table.

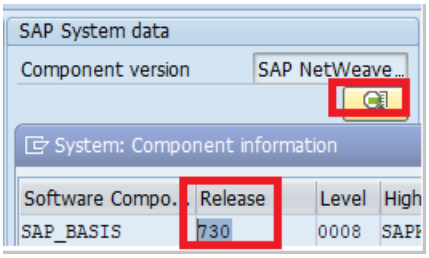
Specifically, this SAP table connector supports:

- Copying data from an SAP table in:
  - SAP ERP Central Component (SAP ECC) version 7.01 or later (in a recent SAP Support Package Stack released after 2015).
  - SAP Business Warehouse (SAP BW) version 7.01 or later (in a recent SAP Support Package Stack released after 2015).
  - SAP S/4HANA.
  - Other products in SAP Business Suite version 7.01 or later (in a recent SAP Support Package Stack released after 2015).
- Copying data from both an SAP transparent table, a pooled table, a clustered table, and a view.
- Copying data by using basic authentication or Secure Network Communications (SNC), if SNC is configured.
- Connecting to an SAP application server or SAP message server.
- Retrieving data via default or custom RFC.

The version 7.01 or later refers to SAP NetWeaver version instead of SAP ECC version. For example,SAP ECC 6.0 EHP 7 in general has NetWeaver version >=7.4. In case you are unsure about your environment, here are the

steps to confirm the version from your SAP system:

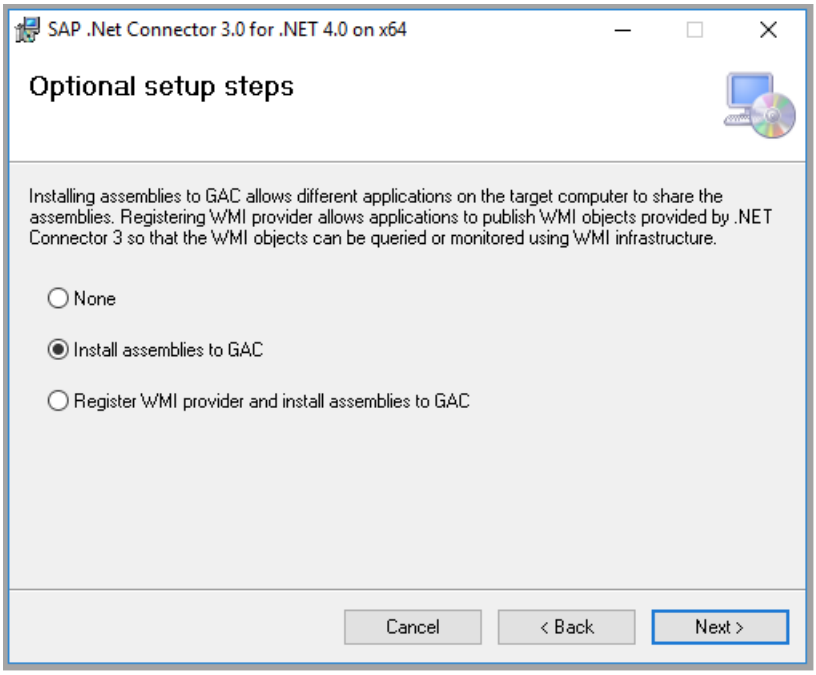
1. Use SAP GUI to connect to the SAP System.
2. Go to **System** -> **Status**.
3. Check the release of the SAP_BASIS, ensure it is equal to or larger than 701.



Prerequisites

To use this SAP table connector, you need to:

- Set up a self-hosted integration runtime (version 3.17 or later). For more information, see Create and configure a self-hosted integration runtime.
- Download the 64-bit SAP Connector for Microsoft .NET 3.0 from SAP's website, and install it on the self-hosted integration runtime machine. During installation, make sure you select the **Install Assemblies to GAC** option in the **Optional setup steps** window.



- The SAP user who's being used in the SAP table connector must have the following permissions:
  - Authorization for using Remote Function Call (RFC) destinations.
  - Permissions to the Execute activity of the S_SDSAUTH authorization object. You can refer to SAP Note 460089 on the majority authorization objects. Certain RFCs are required by the underlying NCo connector, for example RFC_FUNCTION_SEARCH.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal

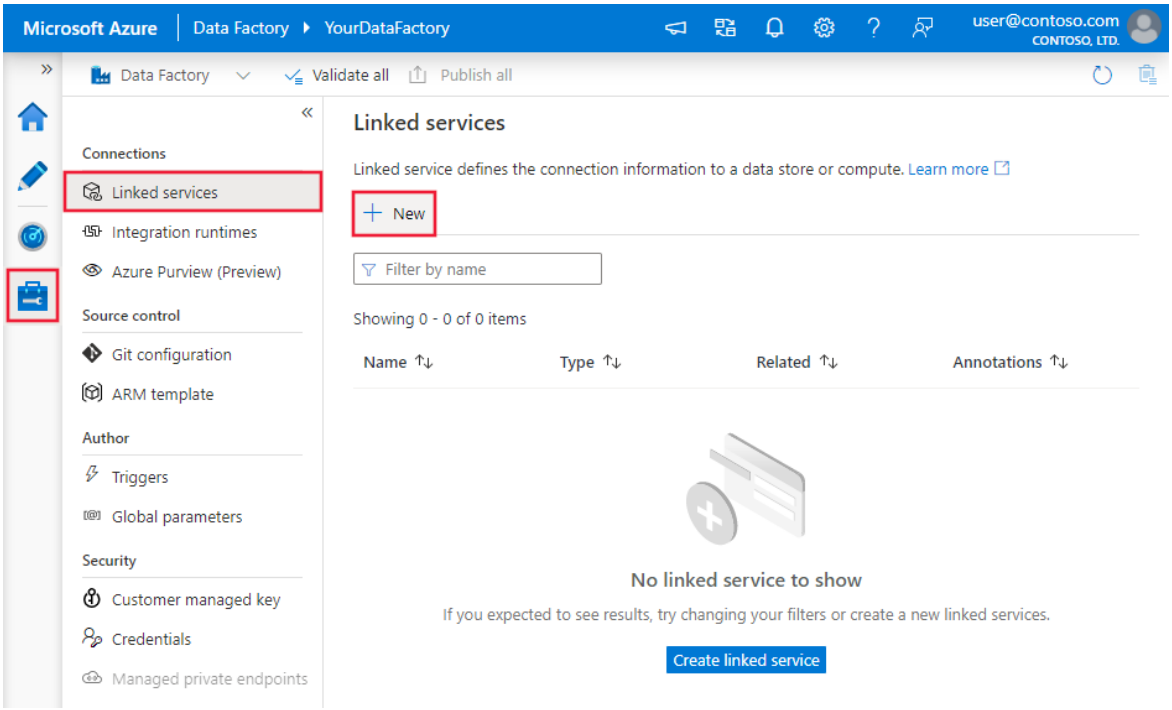
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to an SAP table using UI

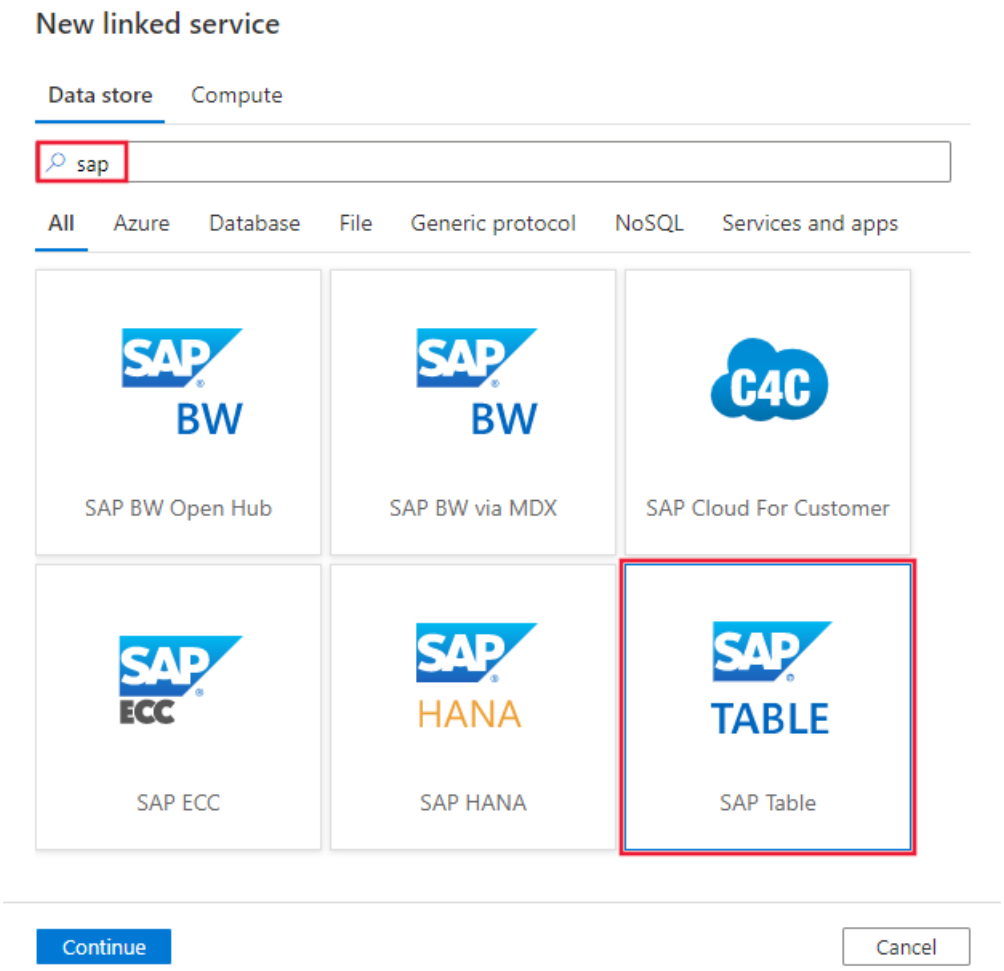
Use the following steps to create a linked service to an SAP table in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

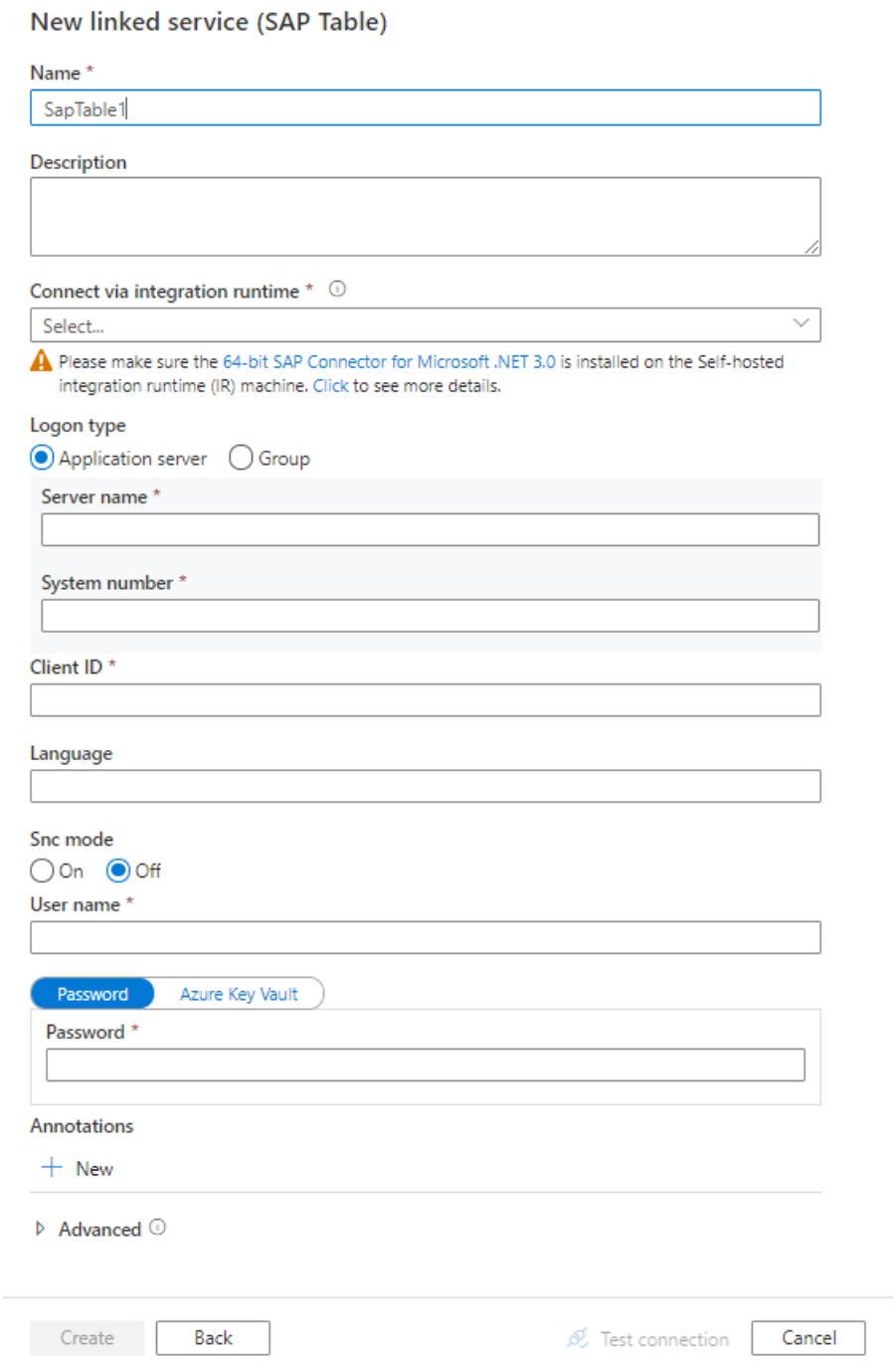
   - Azure Data Factory
   - Azure Synapse



2. Search for SAP and select the SAP table connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define the entities specific to the SAP table connector.

Linked service properties

The following properties are supported for the SAP BW Open Hub linked service:

PROPERTY	DESCRIPTION	REQUIRED
`type`	The `type` property must be set to `SapTable`.	Yes

PROPERTY	DESCRIPTION	REQUIRED
`server`	The name of the server on which the SAP instance is located. Use to connect to an SAP application server.	No
`systemNumber`	The system number of the SAP system. Use to connect to an SAP application server. Allowed value: A two-digit decimal number represented as a string.	No
`messageServer`	The host name of the SAP message server. Use to connect to an SAP message server.	No
`messageServerService`	The service name or port number of the message server. Use to connect to an SAP message server.	No
`systemId`	The ID of the SAP system where the table is located. Use to connect to an SAP message server.	No
`logonGroup`	The logon group for the SAP system. Use to connect to an SAP message server.	No
`clientId`	The ID of the client in the SAP system. Allowed value: A three-digit decimal number represented as a string.	Yes
`language`	The language that the SAP system uses. Default value is `EN`.	No
`userName`	The name of the user who has access to the SAP server.	Yes
`password`	The password for the user. Mark this field with the `SecureString` type to store it securely, or reference a secret stored in Azure Key Vault.	Yes
`sncMode`	The SNC activation indicator to access the SAP server where the table is located. Use if you want to use SNC to connect to the SAP server. Allowed values are `0` (off, the default) or `1` (on).	No

PROPERTY	DESCRIPTION	REQUIRED
`sncMyName`	The initiator's SNC name to access the SAP server where the table is located. Applies when `sncMode` is on.	No
`sncPartnerName`	The communication partner's SNC name to access the SAP server where the table is located. Applies when `sncMode` is on.	No
`sncLibraryPath`	The external security product's library to access the SAP server where the table is located. Applies when `sncMode` is on.	No
`sncQop`	The SNC Quality of Protection level to apply. Applies when `sncMode` is On. Allowed values are `1` (Authentication), `2` (Integrity), `3` (Privacy), `8` (Default), `9` (Maximum).	No
`connectVia`	The integration runtime to be used to connect to the data store. A self-hosted integration runtime is required, as mentioned earlier in Prerequisites.	Yes

Example 1: Connect to an SAP application server

```
{
    "name": "SapTableLinkedService",
    "properties": {
        "type": "SapTable",
        "typeProperties": {
            "server": "<server name>",
            "systemNumber": "<system number>",
            "clientId": "<client ID>",
            "userName": "<SAP user>",
            "password": {
                "type": "SecureString",
                "value": "<Password for SAP user>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 2: Connect to an SAP message server

```
{
    "name": "SapTableLinkedService",
    "properties": {
        "type": "SapTable",
        "typeProperties": {
            "messageServer": "<message server name>",
            "messageServerService": "<service name or port>",
            "systemId": "<system ID>",
            "logonGroup": "<logon group>",
            "clientId": "<client ID>",
            "userName": "<SAP user>",
            "password": {
                "type": "SecureString",
                "value": "<Password for SAP user>"
            }
        },
        "connectVia": {
            "referenceName": "<name of integration runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 3: Connect by using SNC

```
{
    "name": "SapTableLinkedService",
    "properties": {
        "type": "SapTable",
        "typeProperties": {
            "server": "<server name>",
            "systemNumber": "<system number>",
            "clientId": "<client ID>",
            "userName": "<SAP user>",
            "password": {
                "type": "SecureString",
                "value": "<Password for SAP user>"
            },
            "sncMode": 1,
            "sncMyName": "<SNC myname>",
            "sncPartnerName": "<SNC partner name>",
            "sncLibraryPath": "<SNC library path>",
            "sncQop": "8"
        },
        "connectVia": {
            "referenceName": "<name of integration runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of the sections and properties for defining datasets, see Datasets. The following section provides a list of the properties supported by the SAP table dataset.

To copy data from and to the SAP BW Open Hub linked service, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED

PROPERTY	DESCRIPTION	REQUIRED
`type`	The `type` property must be set to `SapTableResource`.	Yes
`tableName`	The name of the SAP table to copy data from.	Yes

Example

```
{
    "name": "SAPTableDataset",
    "properties": {
        "type": "SapTableResource",
        "typeProperties": {
            "tableName": "<SAP table name>"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<SAP table linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of the sections and properties for defining activities, see Pipelines. The following section provides a list of the properties supported by the SAP table source.

SAP table as source

To copy data from an SAP table, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
`type`	The `type` property must be set to `SapTableSource`.	Yes
`rowCount`	The number of rows to be retrieved.	No
`rfcTableFields`	The fields (columns) to copy from the SAP table. For example, `column0, column1`.	No
`rfcTableOptions`	The options to filter the rows in an SAP table. For example, `COLUMN0 EQ 'SOMEVALUE'`. See also the SAP query operator table later in this article.	No

PROPERTY	DESCRIPTION	REQUIRED
`customRfcReadTableFunctionModule`	A custom RFC function module that can be used to read data from an SAP table. You can use a custom RFC function module to define how the data is retrieved from your SAP system and returned to the service. The custom function module must have an interface implemented (import, export, tables) that's similar to `/SAPDS/RFC_READ_TABLE2`, which is the default interface used by the service.	No
`partitionOption`	The partition mechanism to read from an SAP table. The supported options include: • `None` • `PartitionOnInt` (normal integer or integer values with zero padding on the left, such as `0000012345`) • `PartitionOnCalendarYear` (4 digits in the format "YYYY") • `PartitionOnCalendarMonth` (6 digits in the format "YYYYMM") • `PartitionOnCalendarDate` (8 digits in the format "YYYYMMDD") • `PartitionOntime` (6 digits in the format "HHMMSS", such as `235959`)	No
`partitionColumnName`	The name of the column used to partition the data.	No
`partitionUpperBound`	The maximum value of the column specified in `partitionColumnName` that will be used to continue with partitioning.	No
`partitionLowerBound`	The minimum value of the column specified in `partitionColumnName` that will be used to continue with partitioning. (Note: `partitionLowerBound` cannot be "0" when partition option is `PartitionOnInt`)	No
`maxPartitionsNumber`	The maximum number of partitions to split the data into. The default value is 1.	No

PROPERTY	DESCRIPTION	REQUIRED
`sapDataColumnDelimiter`	The single character that is used as delimiter passed to SAP RFC to split the output data.	No

TIP

If your SAP table has a large volume of data, such as several billion rows, use `partitionOption` and `partitionSetting` to split the data into smaller partitions. In this case, the data is read per partition, and each data partition is retrieved from your SAP server via a single RFC call.

Taking `partitionOption` as `partitionOnInt` as an example, the number of rows in each partition is calculated with this formula: (total rows falling between `partitionUpperBound` and `partitionLowerBound`)/ `maxPartitionsNumber` .

To load data partitions in parallel to speed up copy, the parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your SAP table. We strongly recommend making `maxPartitionsNumber` a multiple of the value of the `parallelCopies` property. When copying data into file-based data store, it's also recommanded to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

TIP

The `BASXML` is enabled by default for this SAP Table connector within the service.

In `rfcTableOptions` , you can use the following common SAP query operators to filter the rows:

OPERATOR	DESCRIPTION
`EQ`	Equal to
`NE`	Not equal to
`LT`	Less than
`LE`	Less than or equal to
`GT`	Greater than
`GE`	Greater than or equal to
`IN`	As in `TABCLASS IN ('TRANSP', 'INTTAB')`
`LIKE`	As in `LIKE 'Emma%'`

Example

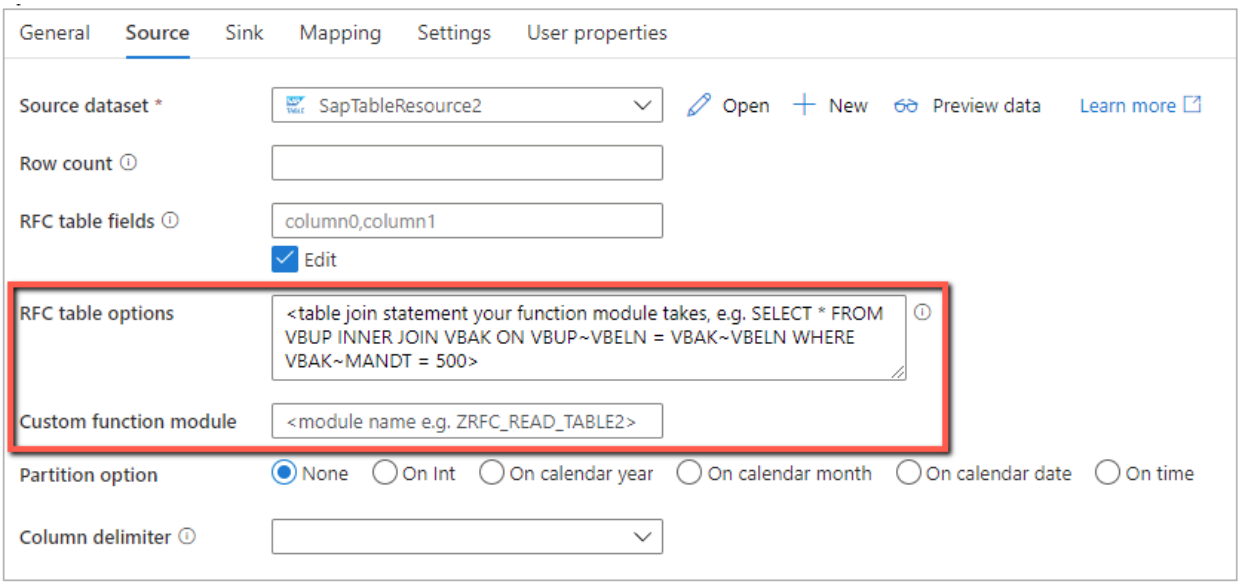
```
"activities":[
    {
        "name": "CopyFromSAPTable",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SAP table input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SapTableSource",
                "partitionOption": "PartitionOnInt",
                "partitionSettings": {
                     "partitionColumnName": "<partition column name>",
                     "partitionUpperBound": "2000",
                     "partitionLowerBound": "1",
                     "maxPartitionsNumber": 500
                 }
            },
            "sink": {
                "type": "<sink type>"
            },
            "parallelCopies": 4
        }
    }
]
```

Join SAP tables

Currently SAP Table connector only supports one single table with the default function module. To get the joined data of multiple tables, you can leverage the customRfcReadTableFunctionModule property in the SAP Table connector following steps below:

- Write a custom function module, which can take a query as OPTIONS and apply your own logic to retrieve the data.
- For the "Custom function module", enter the name of your custom function module.
- For the "RFC table options", specify the table join statement to feed into your function module as OPTIONS, such as "`<TABLE1>` INNER JOIN `<TABLE2>` ON COLUMN0".

Below is an example:



TIP

You can also consider having the joined data aggregated in the VIEW, which is supported by SAP Table connector. You can also try to extract related tables to get onboard onto Azure (e.g. Azure Storage, Azure SQL Database), then use Data Flow to proceed with further join or filter.

Create custom function module

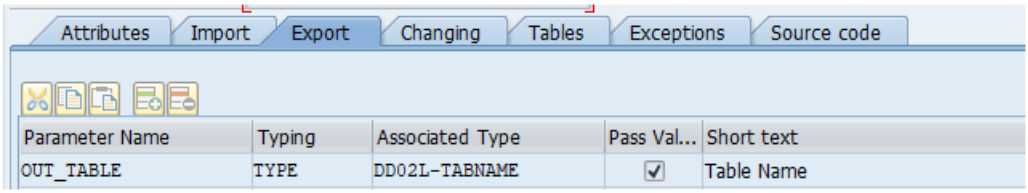
For SAP table, currently we support customRfcReadTableFunctionModule property in the copy source, which allows you to leverage your own logic and process data.

As a quick guidance, here are some requirements to get started with the "Custom function module":

- Definition:

```
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(QUERY_TABLE) TYPE  DD02L-TABNAME
*"     VALUE(DELIMITER) TYPE  SONV-FLAG OPTIONAL
*"     VALUE(NO_DATA) TYPE  SONV-FLAG OPTIONAL
*"     VALUE(ROWSKIPS) TYPE  SOID-ACCNT OPTIONAL
*"     VALUE(ROWCOUNT) TYPE  SOID-ACCNT OPTIONAL
*"  EXPORTING
*"     VALUE(OUT_TABLE) TYPE  DD02L-TABNAME
*"  TABLES
*"      OPTIONS STRUCTURE  /SAPDS/RFC_DB_OPT
*"      FIELDS STRUCTURE  RFC_DB_FLD
*"      TBLOUT128 STRUCTURE  /SAPDS/TAB128
*"      TBLOUT512 STRUCTURE  /SAPDS/TAB512
*"      TBLOUT2048 STRUCTURE  /SAPDS/TAB2048
*"      TBLOUT8192 STRUCTURE  /SAPDS/TAB8192
*"      TBLOUT30000 STRUCTURE  /SAPDS/TAB30K
*"  EXCEPTIONS
*"      TABLE_NOT_AVAILABLE
*"      TABLE_WITHOUT_DATA
*"      OPTION_NOT_VALID
*"      FIELD_NOT_VALID
*"      NOT_AUTHORIZED
*"      DATA_BUFFER_EXCEEDED
*"----------------------------------------------------------------------
```

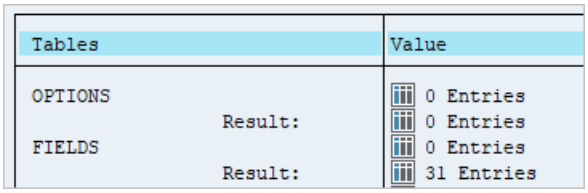
- Export data into one of the tables below:



Parameter Name	Typing	Associated Type	Optional	Short text
OPTIONS	LIKE	/SAPDS/RFC_DB_OPT	☐	RFC Table Read: Select Options / WH...
FIELDS	LIKE	RFC_DB_FLD	☐	RFC Table Read: Description of Fields ...
TBLOUT128	LIKE	/SAPDS/TAB128	☐	DataServices: Table with 128 char field
TBLOUT512	LIKE	/SAPDS/TAB512	☐	DataServices: Table with 512 chars
TBLOUT2048	LIKE	/SAPDS/TAB2048	☐	DataServices: Table with 2048 chars
TBLOUT8192	LIKE	/SAPDS/TAB8192	☐	DataServices: Table with 8192 chars
TBLOUT30000	LIKE	/SAPDS/TAB30K	☐	DataServices: Table with 30000 chars

Below are illustrations of how SAP table connector works with custom function module:

1. Build connection with SAP server via SAP NCO.
2. Invoke "Custom function module" with the parameters set as below:
   - QUERY_TABLE: the table name you set in the SAP Table dataset;
   - Delimiter: the delimiter you set in the SAP Table Source;
   - ROWCOUNT/Option/Fields: the Rowcount/Aggregated Option/Fields you set in the Table source.
3. Get the result and parse the data in below ways:
   a. Parse the value in the Fields table to get the schemas.



   b. Get the values of the output table to see which table contains these values.

   c. Get the values in the OUT_TABLE, parse the data and then write it into the sink.

Data type mappings for an SAP table

When you're copying data from an SAP table, the following mappings are used from the SAP table data types to interim data types used within the service. To learn how the copy activity maps the source schema and data type to the sink, see Schema and data type mappings.

SAP ABAP TYPE	SERVICE INTERIM DATA TYPE
`C` (String)	`String`
`I` (Integer)	`Int32`
`F` (Float)	`Double`
`D` (Date)	`String`

SAP ABAP TYPE	SERVICE INTERIM DATA TYPE
`T` (Time)	`String`
`P` (BCD Packed, Currency, Decimal, Qty)	`Decimal`
`N` (Numeric)	`String`
`X` (Binary and Raw)	`String`

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of the data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from ServiceNow using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in Azure Data Factory and Synapse Analytics pipelines to copy data from ServiceNow. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This ServiceNow connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity. Therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

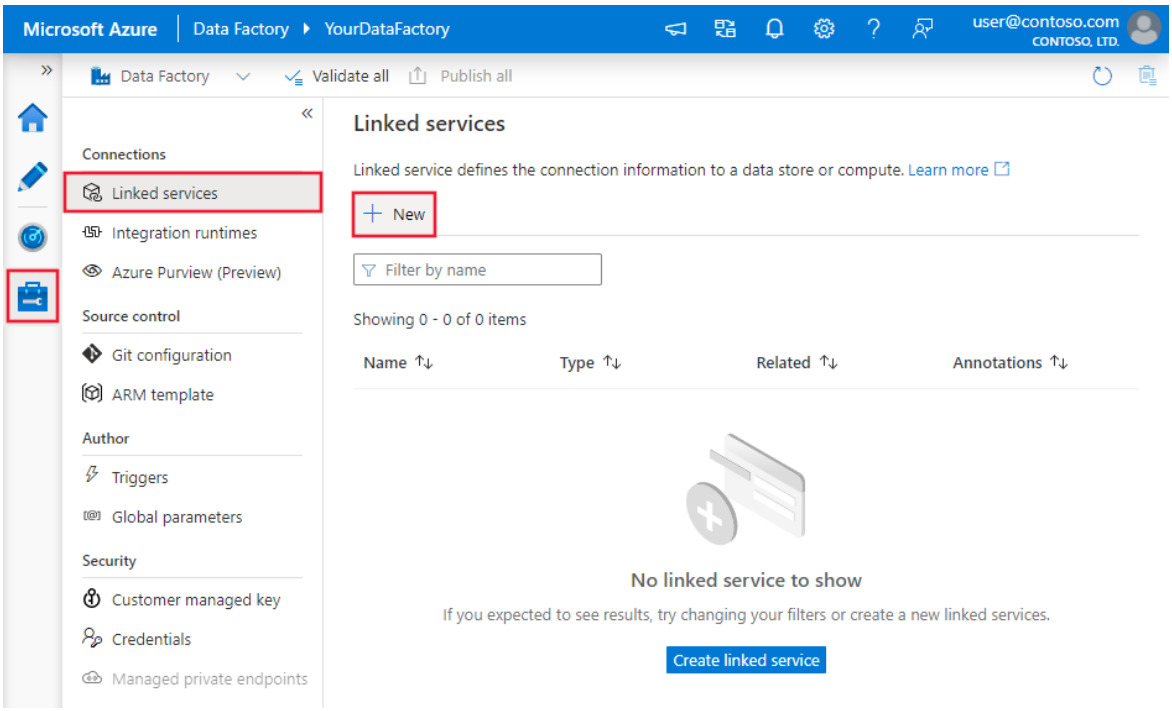
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to ServiceNow using UI

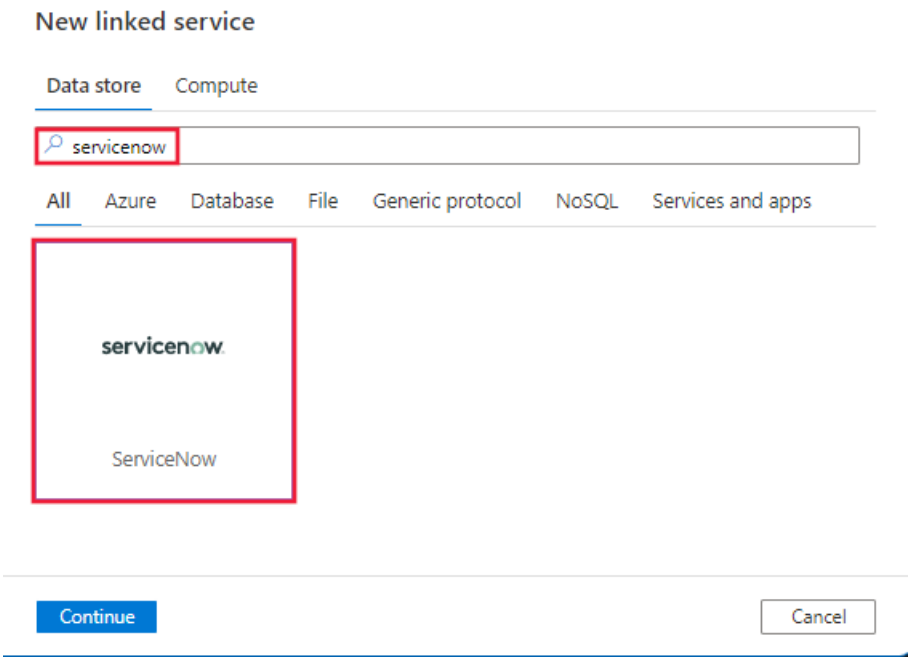
Use the following steps to create a linked service to ServiceNow in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

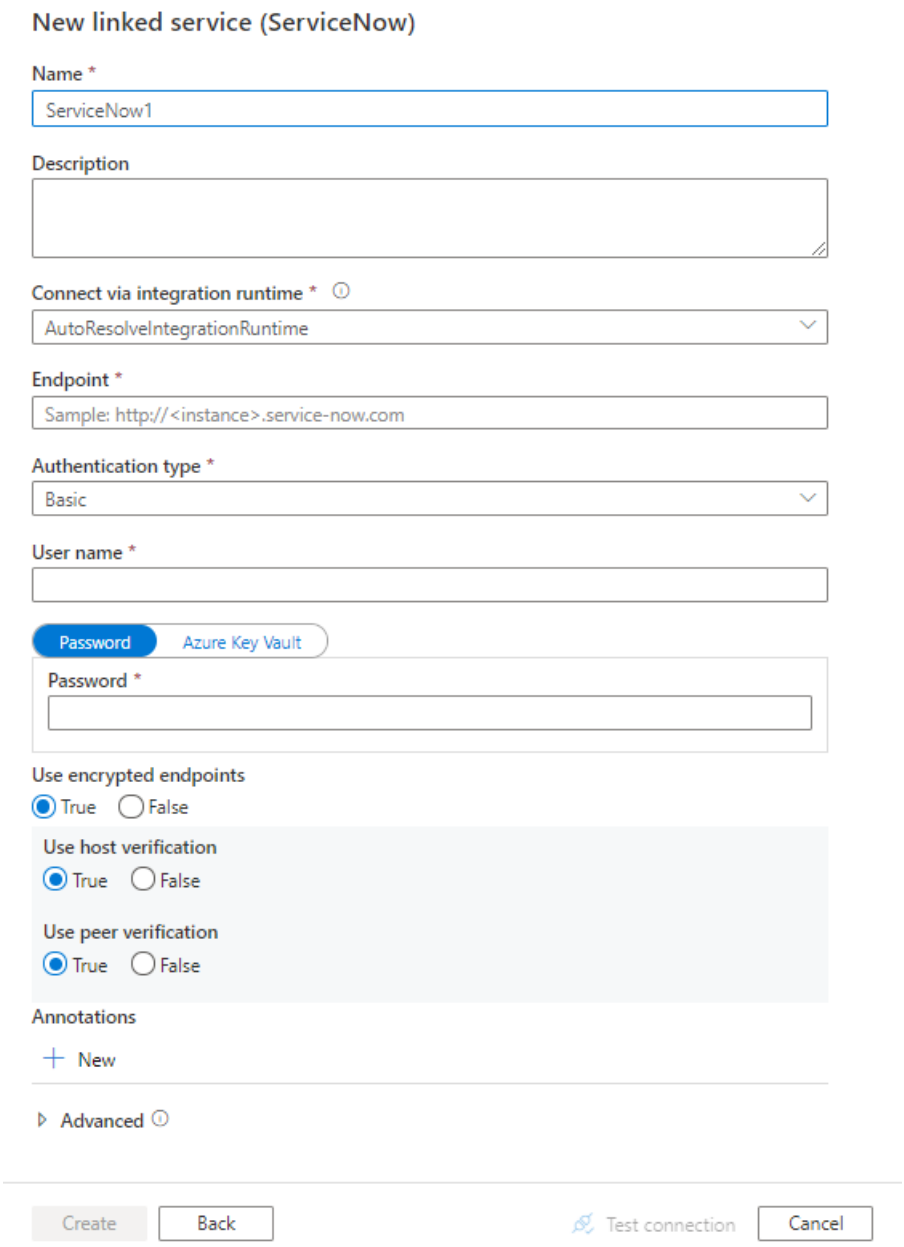
   - Azure Data Factory
   - Azure Synapse



2. Search for ServiceNow and select the ServiceNow connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to ServiceNow connector.

Linked service properties

The following properties are supported for ServiceNow linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **ServiceNow**	Yes
endpoint	The endpoint of the ServiceNow server (`http://<instance>.service-now.com`).	Yes

PROPERTY	DESCRIPTION	REQUIRED
authenticationType	The authentication type to use. Allowed values are: **Basic**, **OAuth2**	Yes
username	The user name used to connect to the ServiceNow server for Basic and OAuth2 authentication.	Yes
password	The password corresponding to the user name for Basic and OAuth2 authentication. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
clientId	The client ID for OAuth2 authentication.	No
clientSecret	The client secret for OAuth2 authentication. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
    "name": "ServiceNowLinkedService",
    "properties": {
        "type": "ServiceNow",
        "typeProperties": {
            "endpoint" : "http://<instance>.service-now.com",
            "authenticationType" : "Basic",
            "username" : "<username>",
            "password": {
                 "type": "SecureString",
                 "value": "<password>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by ServiceNow dataset.

To copy data from ServiceNow, set the type property of the dataset to **ServiceNowObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **ServiceNowObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "ServiceNowDataset",
    "properties": {
        "type": "ServiceNowObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<ServiceNow linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by ServiceNow source.

ServiceNow as source

To copy data from ServiceNow, set the source type in the copy activity to **ServiceNowSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **ServiceNowSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM Actual.alm_asset"`.	No (if "tableName" in dataset is specified)

Note the following when specifying the schema and column for ServiceNow in query, and **refer to Performance tips on copy performance implication**.

- **Schema:** specify the schema as `Actual` or `Display` in the ServiceNow query, which you can look at it as the parameter of `sysparm_display_value` as true or false when calling ServiceNow REST APIs.
- **Column:** the column name for actual value under `Actual` schema is `[column name]_value`, while for display value under `Display` schema is `[column name]_display_value`. Note the column name need map to the schema being used in the query.

Sample query: `SELECT col_value FROM Actual.alm_asset` OR `SELECT col_display_value FROM Display.alm_asset`

Example:

```
"activities":[
    {
        "name": "CopyFromServiceNow",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<ServiceNow input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "ServiceNowSource",
                "query": "SELECT * FROM Actual.alm_asset"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Performance tips

Schema to use

ServiceNow has 2 different schemas, one is **"Actual"** which returns actual data, the other is **"Display"** which returns the display values of data.

If you have a filter in your query, use "Actual" schema which has better copy performance. When querying against "Actual" schema, ServiceNow natively support filter when fetching the data to only return the filtered resultset, whereas when querying "Display" schema, ADF retrieve all the data and apply filter internally.

Index

ServiceNow table index can help improve query performance, refer to Create a table index.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy and transform data in SFTP server using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 21 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity to copy data from and to the secure FTP (SFTP) server, and use Data Flow to transform data in SFTP server. To learn more read the introductory article for Azure Data Factory or Azure Synapse Analytics.

Supported capabilities

This SFTP connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Mapping data flow (source/sink)	①
Lookup activity	① ②
GetMetadata activity	① ②
Delete activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

Specifically, the SFTP connector supports:

- Copying files from and to the SFTP server by using **Basic**, **SSH public key** or **multi-factor** authentication.
- Copying files as is or by parsing or generating files with the supported file formats and compression codecs.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

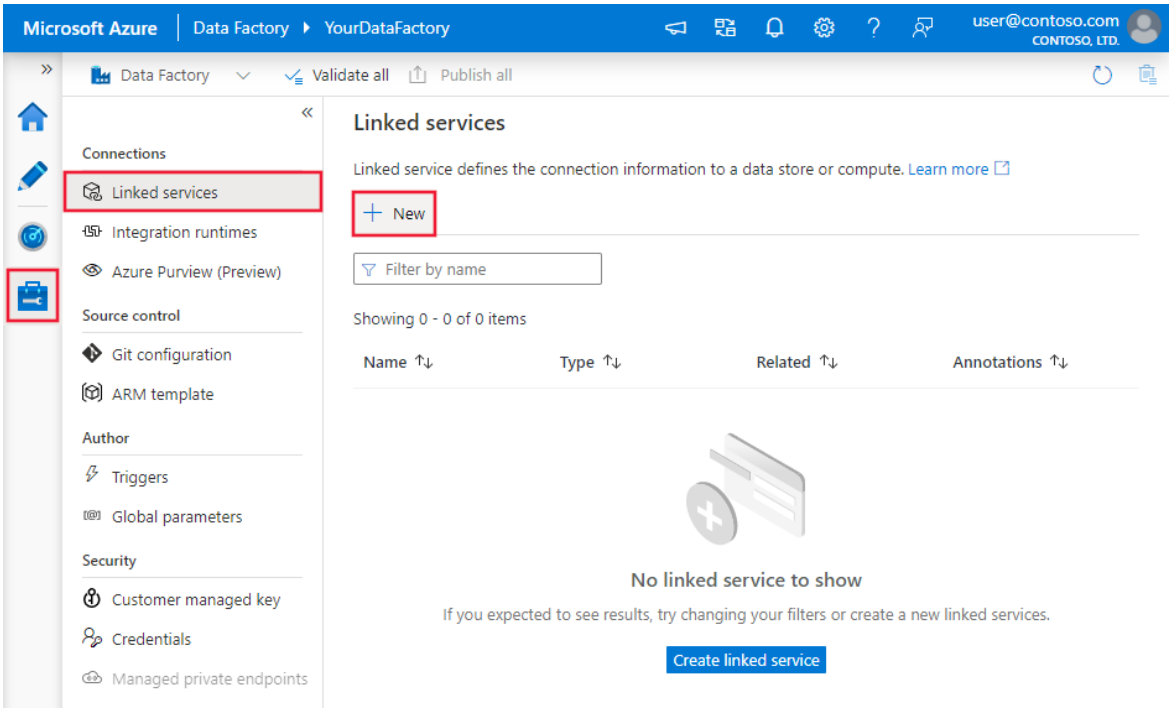
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create an SFTP linked service using UI

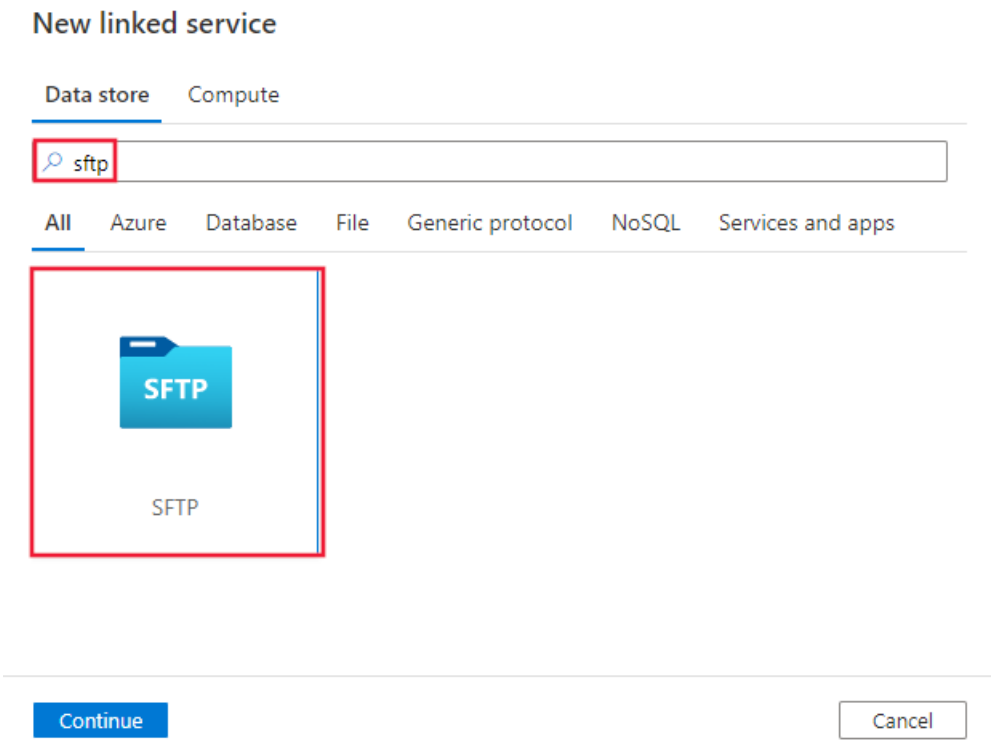
Use the following steps to create an SFTP linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

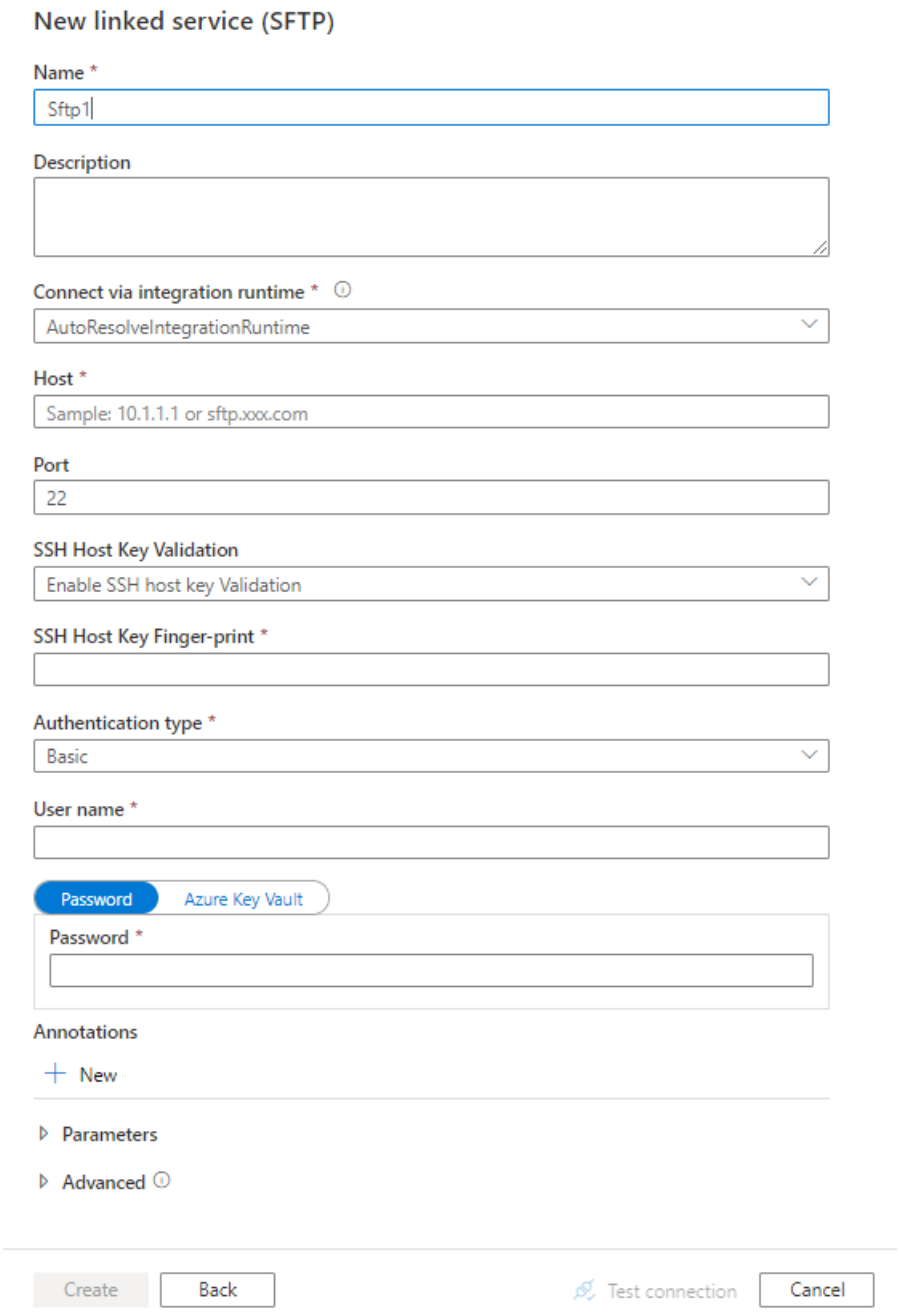
   - Azure Data Factory
   - Azure Synapse



2. Search for SFTP and select the SFTP connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define entities specific to SFTP.

Linked service properties

The following properties are supported for the SFTP linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to *Sftp*.	Yes
host	The name or IP address of the SFTP server.	Yes

PROPERTY	DESCRIPTION	REQUIRED
port	The port on which the SFTP server is listening. The allowed value is an integer, and the default value is *22*.	No
skipHostKeyValidation	Specify whether to skip host key validation. Allowed values are *true* and *false* (default).	No
hostKeyFingerprint	Specify the fingerprint of the host key.	Yes, if the "skipHostKeyValidation" is set to false.
authenticationType	Specify the authentication type. Allowed values are *Basic*, *SshPublicKey* and *MultiFactor*. For more properties, see the Use basic authentication section. For JSON examples, see the Use SSH public key authentication section.	Yes
connectVia	The integration runtime to be used to connect to the data store. To learn more, see the Prerequisites section. If the integration runtime isn't specified, the service uses the default Azure Integration Runtime.	No

Use basic authentication

To use basic authentication, set the *authenticationType* property to *Basic*, and specify the following properties in addition to the SFTP connector generic properties that were introduced in the preceding section:

PROPERTY	DESCRIPTION	REQUIRED
userName	The user who has access to the SFTP server.	Yes
password	The password for the user (userName). Mark this field as a SecureString to store it securely, or reference a secret stored in an Azure key vault.	Yes

Example:

```
{
    "name": "SftpLinkedService",
    "properties": {
        "type": "Sftp",
        "typeProperties": {
            "host": "<sftp server>",
            "port": 22,
            "skipHostKeyValidation": false,
            "hostKeyFingerPrint": "ssh-rsa 2048 xx:00:00:00:xx:00:x0:0x:0x:0x:0x:00:00:x0:x0:00",
            "authenticationType": "Basic",
            "userName": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of integration runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Use SSH public key authentication

To use SSH public key authentication, set "authenticationType" property as **SshPublicKey**, and specify the following properties besides the SFTP connector generic ones introduced in the last section:

PROPERTY	DESCRIPTION	REQUIRED
userName	The user who has access to the SFTP server.	Yes
privateKeyPath	Specify the absolute path to the private key file that the integration runtime can access. This applies only when the self-hosted type of integration runtime is specified in "connectVia."	Specify either `privateKeyPath` or `privateKeyContent`.
privateKeyContent	Base64 encoded SSH private key content. SSH private key should be OpenSSH format. Mark this field as a SecureString to store it securely, or reference a secret stored in an Azure key vault.	Specify either `privateKeyPath` or `privateKeyContent`.
passPhrase	Specify the pass phrase or password to decrypt the private key if the key file or the key content is protected by a pass phrase. Mark this field as a SecureString to store it securely, or reference a secret stored in an Azure key vault.	Yes, if the private key file or the key content is protected by a pass phrase.

NOTE

The SFTP connector supports an RSA/DSA OpenSSH key. Make sure that your key file content starts with "-----BEGIN [RSA/DSA] PRIVATE KEY-----". If the private key file is a PPK-format file, use the PuTTY tool to convert from PPK to OpenSSH format.

Example 1: SshPublicKey authentication using private key filePath

```
{
    "name": "SftpLinkedService",
    "properties": {
        "type": "Sftp",
        "typeProperties": {
            "host": "<sftp server>",
            "port": 22,
            "skipHostKeyValidation": true,
            "authenticationType": "SshPublicKey",
            "userName": "xxx",
            "privateKeyPath": "D:\\privatekey_openssh",
            "passPhrase": {
                "type": "SecureString",
                "value": "<pass phrase>"
            }
        },
        "connectVia": {
            "referenceName": "<name of integration runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example 2: SshPublicKey authentication using private key content

```
{
    "name": "SftpLinkedService",
    "type": "Linkedservices",
    "properties": {
        "type": "Sftp",
        "typeProperties": {
            "host": "<sftp server>",
            "port": 22,
            "skipHostKeyValidation": true,
            "authenticationType": "SshPublicKey",
            "userName": "<username>",
            "privateKeyContent": {
                "type": "SecureString",
                "value": "<base64 string of the private key content>"
            },
            "passPhrase": {
                "type": "SecureString",
                "value": "<pass phrase>"
            }
        },
        "connectVia": {
            "referenceName": "<name of integration runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Use multi-factor authentication

To use multi-factor authentication which is a combination of basic and SSH public key authentications, specify the user name, password and the private key info described in above sections.

Example: multi-factor authentication

```
{
    "name": "SftpLinkedService",
    "properties": {
        "type": "Sftp",
        "typeProperties": {
            "host": "<host>",
            "port": 22,
            "authenticationType": "MultiFactor",
            "userName": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            },
            "privateKeyContent": {
                "type": "SecureString",
                "value": "<base64 encoded private key content>"
            },
            "passPhrase": {
                "type": "SecureString",
                "value": "<passphrase for private key>"
            }
        },
        "connectVia": {
            "referenceName": "<name of integration runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties that are available for defining datasets, see the Datasets article.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for SFTP under `location` settings in the format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The *type* property under `location` in dataset must be set to *SftpLocation*.	Yes
folderPath	The path to the folder. If you want to use a wildcard to filter the folder, skip this setting and specify the path in activity source settings.	No

PROPERTY	DESCRIPTION	REQUIRED
fileName	The file name under the specified folderPath. If you want to use a wildcard to filter files, skip this setting and specify the file name in activity source settings.	No

Example:

```
{
    "name": "DelimitedTextDataset",
    "properties": {
        "type": "DelimitedText",
        "linkedServiceName": {
            "referenceName": "<SFTP linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, auto retrieved during authoring > ],
        "typeProperties": {
            "location": {
                "type": "SftpLocation",
                "folderPath": "root/folder/subfolder"
            },
            "columnDelimiter": ",",
            "quoteChar": "\"",
            "firstRowAsHeader": true,
            "compressionCodec": "gzip"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties that are available for defining activities, see the Pipelines article. This section provides a list of properties that are supported by the SFTP source.

SFTP as source

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

The following properties are supported for SFTP under the `storeSettings` settings in the format-based Copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The *type* property under `storeSettings` must be set to *SftpReadSettings*.	Yes

PROPERTY	DESCRIPTION	REQUIRED
Locate the files to copy		
OPTION 1: static path	Copy from the folder/file path that's specified in the dataset. If you want to copy all files from a folder, additionally specify `wildcardFileName` as `*`.	
OPTION 2: wildcard - wildcardFolderPath	The folder path with wildcard characters to filter source folders. Allowed wildcards are `*` (matches zero or more characters) and `?` (matches zero or a single character); use `^` to escape if your actual folder name has a wildcard or this escape char inside. For more examples, see Folder and file filter examples.	No
OPTION 2: wildcard - wildcardFileName	The file name with wildcard characters under the specified folderPath/wildcardFolderPath to filter source files. Allowed wildcards are `*` (matches zero or more characters) and `?` (matches zero or a single character); use `^` to escape if your actual file name has wildcard or this escape char inside. For more examples, see Folder and file filter examples.	Yes
OPTION 3: a list of files - fileListPath	Indicates to copy a specified file set. Point to a text file that includes a list of files you want to copy (one file per line, with the relative path to the path configured in the dataset). When you use this option, don't specify the file name in the dataset. For more examples, see File list examples.	No
Additional settings		
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. When recursive is set to true and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are *true* (default) and *false*. This property doesn't apply when you configure `fileListPath`.	No

PROPERTY	DESCRIPTION	REQUIRED
deleteFilesAfterCompletion	Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store. This property is only valid in binary files copy scenario. The default value: false.	No
modifiedDatetimeStart	Files are filtered based on the attribute *Last Modified*. The files are selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to the UTC time zone in the format of *2018-12-01T05:00:00Z*. The properties can be NULL, which means that no file attribute filter is applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is NULL, it means that the files whose last modified attribute is greater than or equal to the datetime value are selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is NULL, it means that the files whose last modified attribute is less than the datetime value are selected. This property doesn't apply when you configure `fileListPath`.	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns. Allowed values are **false** (default) and **true**.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns. If it is not specified, by default, - When you use file path in dataset or list of files on source, partition root path is the path configured in dataset. - When you use wildcard folder filter, partition root path is the sub-path before the first wildcard. For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27": - If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns `month` and `day` with value "08" and "27" respectively, in addition to the columns inside the files. - If partition root path is not specified, no extra column will be generated.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No
disableChunking	When copying data from SFTP, the service tries to get the file length first, then divide the file into multiple parts and read them in parallel. Specify whether your SFTP server supports getting file length or seeking to read from a certain offset. Allowed values are **false** (default), **true**.	No

Example:

```
"activities":[
    {
        "name": "CopyFromSFTP",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Delimited text input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "DelimitedTextSource",
                "formatSettings":{
                    "type": "DelimitedTextReadSettings",
                    "skipLineCount": 10
                },
                "storeSettings":{
                    "type": "SftpReadSettings",
                    "recursive": true,
                    "wildcardFolderPath": "myfolder*A",
                    "wildcardFileName": "*.csv",
                    "disableChunking": false
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

SFTP as a sink

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- JSON format
- ORC format
- Parquet format

The following properties are supported for SFTP under `storeSettings` settings in a format-based Copy sink:

PROPERTY	DESCRIPTION	REQUIRED
type	The *type* property under `storeSettings` must be set to *SftpWriteSettings*.	Yes

PROPERTY	DESCRIPTION	REQUIRED
copyBehavior	Defines the copy behavior when the source is files from a file-based data store. Allowed values are: - **PreserveHierarchy (default)**: Preserves the file hierarchy in the target folder. The relative path of the source file to the source folder is identical to the relative path of the target file to the target folder. - **FlattenHierarchy**: All files from the source folder are in the first level of the target folder. The target files have autogenerated names. - **MergeFiles**: Merges all files from the source folder to one file. If the file name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No
useTempFileRename	Indicate whether to upload to temporary files and rename them, or directly write to the target folder or file location. By default, the service first writes to temporary files and then renames them when the upload is finished. This sequence helps to (1) avoid conflicts that might result in a corrupted file if you have other processes writing to the same file, and (2) ensure that the original version of the file exists during the transfer. If your SFTP server doesn't support a rename operation, disable this option and make sure that you don't have a concurrent write to the target file. For more information, see the troubleshooting tip at the end of this table.	No. Default value is *true*.
operationTimeout	The wait time before each write request to SFTP server times out. Default value is 60 min (01:00:00).	No

TIP

If you receive the error "UserErrorSftpPathNotFound," "UserErrorSftpPermissionDenied," or "SftpOperationFail" when you're writing data into SFTP, and the SFTP user you use *does* have the proper permissions, check to see whether your SFTP server support file rename operation is working. If it isn't, disable the **Upload with temp file** (`useTempFileRename`) option and try again. To learn more about this property, see the preceding table. If you use a self-hosted integration runtime for the Copy activity, be sure to use version 4.6 or later.

Example:

```
"activities":[
    {
        "name": "CopyToSFTP",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "BinarySink",
                "storeSettings":{
                    "type": "SftpWriteSettings",
                    "copyBehavior": "PreserveHierarchy"
                }
            }
        }
    }
]
```

Folder and file filter examples

This section describes the behavior that results from using wildcard filters with folder paths and file names.

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`Folder*`	(empty, use default)	false	FolderA **File1.csv** **File2.json** Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	(empty, use default)	true	FolderA **File1.csv** **File2.json** Subfolder1 **File3.csv** **File4.json** **File5.csv** AnotherFolderB File6.csv

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
`Folder*`	`*.csv`	false	FolderA **File1.csv** File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
`Folder*`	`*.csv`	true	FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** AnotherFolderB File6.csv

File list examples

This table describes the behavior that results from using a file list path in the Copy activity source. It assumes that you have the following source folder structure and want to copy the files that are in bold type:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	AZURE DATA FACTORY CONFIGURATION
root FolderA **File1.csv** File2.json Subfolder1 **File3.csv** File4.json **File5.csv** Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	**In the dataset:** - Folder path: `root/FolderA` **In the Copy activity source:** - File list path: `root/Metadata/FileListToCopy.txt` The file list path points to a text file in the same data store that includes a list of files you want to copy (one file per line, with the relative path to the path configured in the dataset).

Mapping data flow properties

When you're transforming data in mapping data flows, you can read and write files from SFTP in the following formats:

- Avro
- Delimited text
- Excel
- JSON
- ORC
- Parquet
- XML

Format specific settings are located in the documentation for that format. For more information, see Source transformation in mapping data flow and Sink transformation in mapping data flow.

> **NOTE**
> SSH host key validation is not supported in mapping data flow now.

> **NOTE**
> To access on premise SFTP sever, you need to use Azure Data Factory or Synapse workspace Managed Virtual Network using a private endpoint. Refer to this tutorial for detailed steps.

Source transformation

The below table lists the properties supported by SFTP source. You can edit these properties in the **Source options** tab. When using inline dataset, you will see additional settings, which are the same as the properties described in dataset properties section.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Wildcard path	Using a wildcard pattern will instruct ADF to loop through each matching folder and file in a single source transformation. This is an effective way to process multiple files within a single flow.	No	String[]	wildcardPaths
Partition Root Path	If you have partitioned folders in your file source with a `key=value` format (for example, `year=2019`), then you can assign the top level of that partition folder tree to a column name in your data flow data stream.	No	String	partitionRootPath
Allow no files found	If true, an error is not thrown if no files are found.	No	`true` or `false`	ignoreNoFilesFound
List of files	This is a file set. Create a text file that includes a list of relative path files to process. Point to this text file.	No	`true` or `false`	fileList

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Column to store file name	Store the name of the source file in a column in your data. Enter a new column name here to store the file name string.	No	String	rowUrlColumn
After completion	Choose to do nothing with the source file after the data flow runs, delete the source file, or move the source file. The paths for the move are relative.	No	Delete: `true` or `false` Move: `['<from>', '<to>']`	purgeFiles moveFiles
Filter by last modified	You can filter which files you process by specifying a date range of when they were last modified. All date-times are in UTC.	No	Timestamp	modifiedAfter modifiedBefore

SFTP source script example

When you use SFTP dataset as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 ignoreNoFilesFound: true,
 purgeFiles: true,
 fileList: true,
 modifiedAfter: (toTimestamp(1647388800000L)),
 modifiedBefore: (toTimestamp(1647561600000L)),
 partitionRootPath: 'partdata',
 wildcardPaths:['partdata/**/*.csv']) ~> SFTPSource
```

Sink transformation

The below table lists the properties supported by SFTP sink. You can edit these properties in the **Settings** tab. When using inline dataset, you will see additional settings, which are the same as the properties described in dataset properties section.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Clear the folder	Determines whether or not the destination folder gets cleared before the data is written.	No	`true` or `false`	truncate

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
File name option	The naming format of the data written. By default, one file per partition in format `part-#####-tid-<guid>` .	No	Pattern: String Per partition: String[] Name file as column data: String Name folder as column data: String Output to single file: `['<fileName>']`	filePattern partitionFileNames rowUrlColumn rowFolderUrlColumn partitionFileNames
Quote all	Determines whether to enclose all values in quotes.	No	`true` or `false`	quoteAll

SFTP sink script example

When you use SFTP dataset as sink type, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
 validateSchema: false,
 filePattern:'loans[n].csv',
 truncate: true,
 skipDuplicateMapInputs: true,
 skipDuplicateMapOutputs: true) ~> SFTPSink
```

Lookup activity properties

For information about Lookup activity properties, see Lookup activity.

GetMetadata activity properties

For information about GetMetadata activity properties, see GetMetadata activity.

Delete activity properties

For information about Delete activity properties, see Delete activity.

Legacy models

> **NOTE**
> The following models are still supported as is for backward compatibility. We recommend that you use the previously discussed new model, because the authoring UI has switched to generating the new model.

Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED
type	The *type* property of the dataset must be set to *FileShare*.	Yes

PROPERTY	DESCRIPTION	REQUIRED
folderPath	The path to the folder. A wildcard filter is supported. Allowed wildcards are `*` (matches zero or more characters) and `?` (matches zero or a single character); use `^` to escape if your actual file name has a wildcard or this escape char inside. Examples: rootfolder/subfolder/, see more examples in Folder and file filter examples.	Yes
fileName	**Name or wildcard filter** for the files under the specified "folderPath". If you don't specify a value for this property, the dataset points to all files in the folder. For filter, the allowed wildcards are `*` (matches zero or more characters) and `?` (matches zero or a single character). - Example 1: `"fileName": "*.csv"` - Example 2: `"fileName": "???20180427.txt"` Use `^` to escape if your actual folder name has wildcard or this escape char inside.	No
modifiedDatetimeStart	Files are filtered based on the attribute *Last Modified*. The files are selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to UTC time zone in the format of *2018-12-01T05:00:00Z*. The overall performance of data movement will be affected by enabling this setting when you want to do file filter from large numbers of files. The properties can be NULL, which means that no file attribute filter is applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is NULL, it means that the files whose last modified attribute is greater than or equal to the datetime value are selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is NULL, it means that the files whose last modified attribute is less than the datetime value are selected.	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeEnd	Files are filtered based on the attribute *Last Modified*. The files are selected if their last modified time is greater than or equal to `modifiedDatetimeStart` and less than `modifiedDatetimeEnd`. The time is applied to UTC time zone in the format of *2018-12-01T05:00:00Z*. The overall performance of data movement will be affected by enabling this setting when you want to do file filter from large numbers of files. The properties can be NULL, which means that no file attribute filter is applied to the dataset. When `modifiedDatetimeStart` has a datetime value but `modifiedDatetimeEnd` is NULL, it means that the files whose last modified attribute is greater than or equal to the datetime value are selected. When `modifiedDatetimeEnd` has a datetime value but `modifiedDatetimeStart` is NULL, it means that the files whose last modified attribute is less than the datetime value are selected.	No
format	If you want to copy files as is between file-based stores (binary copy), skip the format section in both input and output dataset definitions. If you want to parse files with a specific format, the following file format types are supported: *TextFormat*, *JsonFormat*, *AvroFormat*, *OrcFormat*, and *ParquetFormat*. Set the *type* property under format to one of these values. For more information, see Text format, Json format, Avro format, Orc format, and Parquet format sections.	No (only for binary copy scenario)
compression	Specify the type and level of compression for the data. For more information, see Supported file formats and compression codecs. Supported types are *GZip*, *Deflate*, *BZip2*, and *ZipDeflate*. Supported levels are *Optimal* and *Fastest*.	No

TIP

To copy all files under a folder, specify *folderPath* only.
To copy a single file with a specified name, specify *folderPath* with the folder part and *fileName* with the file name.
To copy a subset of files under a folder, specify *folderPath* with the folder part and *fileName* with the wildcard filter.

> **NOTE**
>
> If you were using *fileFilter* property for the file filter, it is still supported as is, but we recommend that you use the new filter capability added to *fileName* from now on.

Example:

```
{
    "name": "SFTPDataset",
    "type": "Datasets",
    "properties": {
        "type": "FileShare",
        "linkedServiceName":{
            "referenceName": "<SFTP linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "folderPath": "folder/subfolder/",
            "fileName": "*",
            "modifiedDatetimeStart": "2018-12-01T05:00:00Z",
            "modifiedDatetimeEnd": "2018-12-01T06:00:00Z",
            "format": {
                "type": "TextFormat",
                "columnDelimiter": ",",
                "rowDelimiter": "\n"
            },
            "compression": {
                "type": "GZip",
                "level": "Optimal"
            }
        }
    }
}
```

Legacy Copy activity source model

PROPERTY	DESCRIPTION	REQUIRED
type	The *type* property of the Copy activity source must be set to *FileSystemSource*	Yes
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. When recursive is set to *true* and the sink is a file-based store, empty folders and subfolders won't be copied or created at the sink. Allowed values are *true* (default) and *false*	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```
"activities":[
    {
        "name": "CopyFromSFTP",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SFTP input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "FileSystemSource",
                "recursive": true
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Next steps

For a list of data stores that are supported as sources and sinks by the Copy activity, see supported data stores.

Copy data from SharePoint Online List by using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Copy Activity in Azure Data Factory and Azure Synapse pipelines to copy data from SharePoint Online List. The article builds on Copy Activity, which presents a general overview of Copy Activity.

Supported capabilities

This SharePoint Online List connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks, see the Supported data stores table.

Specifically, this SharePoint List Online connector uses service principal authentication and retrieves data via OData protocol.

> **TIP**
> This connector supports copying data from SharePoint Online **List** but not file. Learn how to copy file from Copy file from SharePoint Online section.

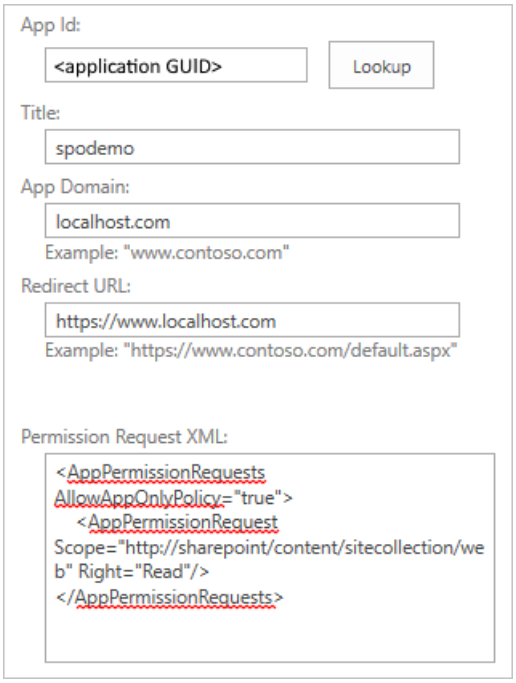
Prerequisites

The SharePoint List Online connector uses service principal authentication to connect to SharePoint. Follow these steps to set it up:

1. Register an application entity in Azure Active Directory (Azure AD) by following Register your application with an Azure AD tenant. Make note of the following values, which you use to define the linked service:
   - Application ID
   - Application key
   - Tenant ID
2. Grant SharePoint Online site permission to your registered application by following the steps below. To do this, you need a site admin role.
   a. Open SharePoint Online site link e.g. `https://[your_site_url]/_layouts/15/appinv.aspx` (replace the site URL).
   b. Search the application ID you registered, fill the empty fields, and click "Create".

- App Domain: `localhost.com`
- Redirect URL: `https://www.localhost.com`
- Permission Request XML:

```
<AppPermissionRequests AllowAppOnlyPolicy="true">
    <AppPermissionRequest Scope="http://sharepoint/content/sitecollection/web"
Right="Read"/>
</AppPermissionRequests>
```



c. Click "Trust It" for this app.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

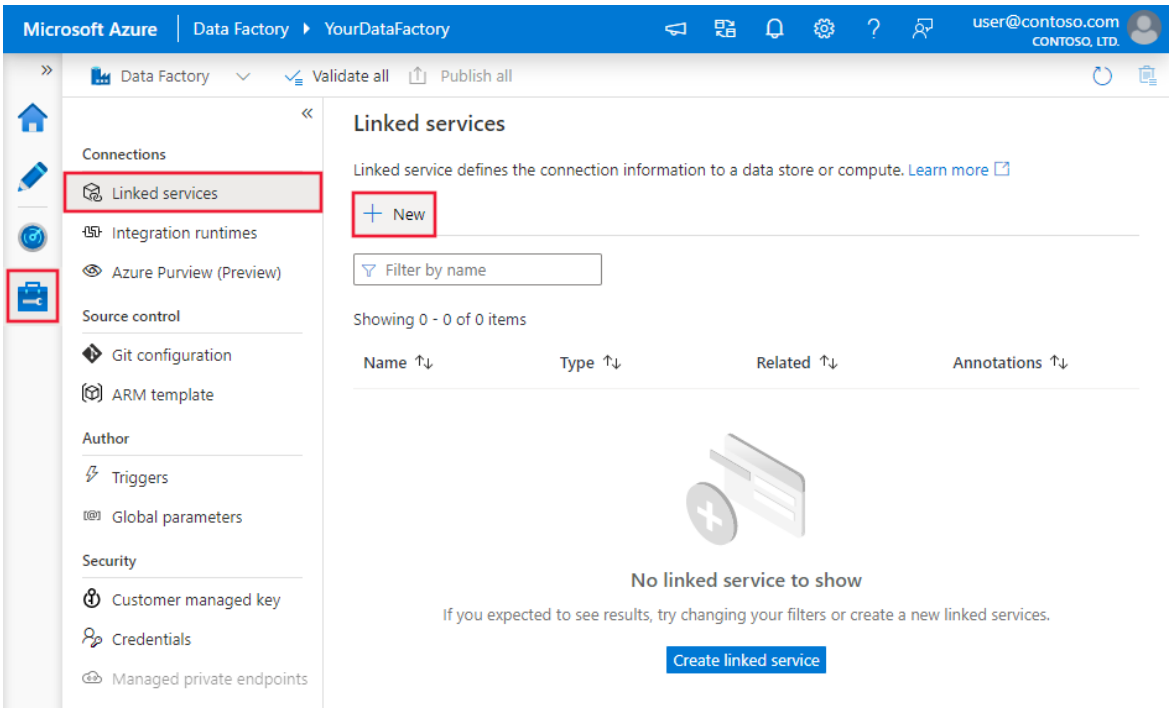
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to a SharePoint Online List using UI

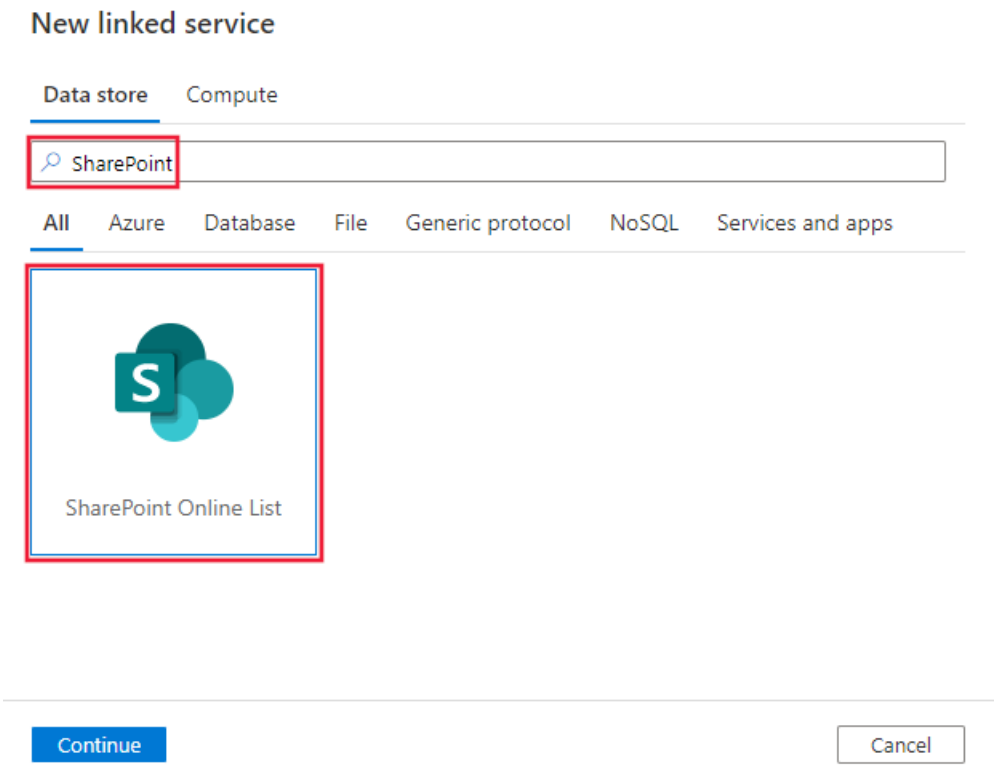
Use the following steps to create a linked service to a SharePoint Online List in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

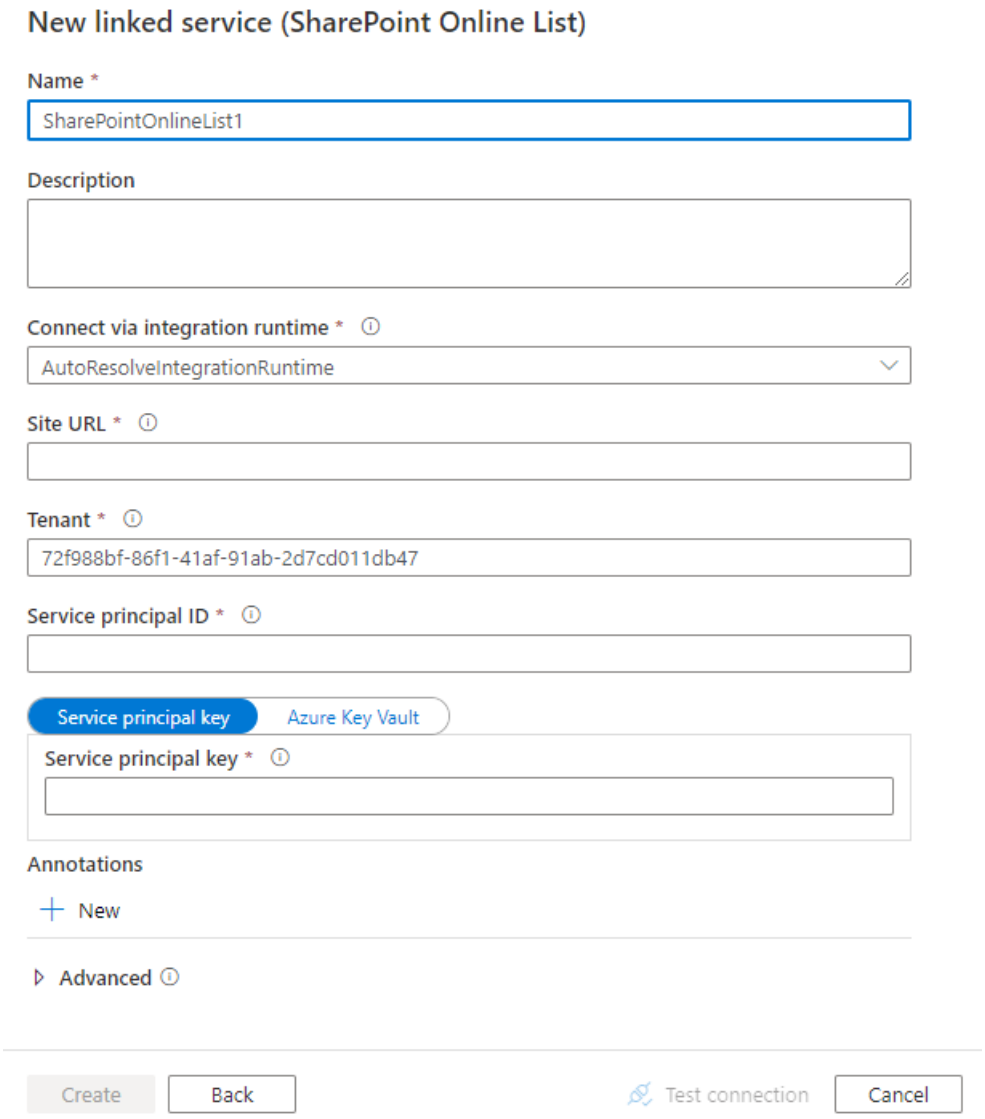
   - Azure Data Factory
   - Azure Synapse



2. Search for SharePoint and select the SharePoint Online List connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties you can use to define entities that are specific to SharePoint Online List connector.

Linked service properties

The following properties are supported for a SharePoint Online List linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to:**SharePointOnlineList**.	Yes
siteUrl	The SharePoint Online site url, e.g. `https://contoso.sharepoint.com/sites/siteName` .	Yes
servicePrincipalId	The Application (client) ID of the application registered in Azure Active Directory.	Yes

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalKey	The application's key. Mark this field as a **SecureString** to store it securely, or reference a secret stored in Azure Key Vault.	Yes
tenantId	The tenant ID under which your application resides.	Yes
connectVia	The Integration Runtime to use to connect to the data store. Learn more from Prerequisites, earlier in this article. If not specified, the default Azure Integration Runtime is used.	No

Example:

```
{
    "name": "SharePointOnlineList",
    "properties": {
        "type": "SharePointOnlineList",
        "typeProperties": {
            "siteUrl": "<site URL>",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            },
            "tenantId": "<tenant ID>"
        }
    }
}
```

Dataset properties

For a full list of sections and properties that are available for defining datasets, see Datasets and linked services. The following section provides a list of the properties supported by the SAP table dataset.

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the dataset must be set to **SharePointOnlineLResource**.	Yes
listName	The name of the SharePoint Online List.	Yes

Example

```
{
    "name": "SharePointOnlineListDataset",
    "properties":
    {
        "type": "SharePointOnlineListResource",
        "linkedServiceName": {
            "referenceName": "<SharePoint Online List linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties":
        {
            "listName":"<name of the list>"
        }
    }
}
```

Copy Activity properties

For a full list of sections and properties that are available for defining activities, see Pipelines. The following section provides a list of the properties supported by the SharePoint Online List source.

SharePoint Online List as source

To copy data from SharePoint Online List, the following properties are supported in the Copy Activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The **type** property of the Copy Activity source must be set to **SharePointOnlineListSource**.	Yes
query	Custom OData query options for filtering data. Example: `"$top=10&$select=Title,Number"`.	No
httpRequestTimeout	The timeout (in second) for the HTTP request to get a response. Default is 300 (5 minutes).	No

Example

```
"activities":[
    {
        "name": "CopyFromSharePointOnlineList",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SharePoint Online List input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source":{
                "type":"SharePointOnlineListSource",
                "query":"<ODataquerye.g.$top=10&$select=Title,Number>"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

NOTE

It isn't possible to select more than one *choice* data type for a SharePoint Online List source.

Data type mapping for SharePoint Online List

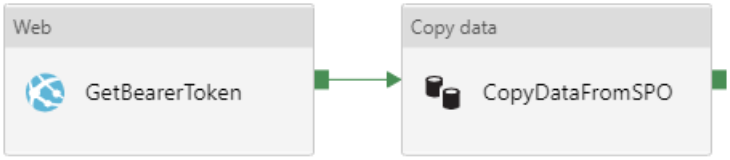
When you copy data from SharePoint Online List, the following mappings are used between SharePoint Online List data types and interim data types used by the service internally.

SHAREPOINT ONLINE DATA TYPE	ODATA DATA TYPE	INTERIM DATA TYPE
Single line of text	Edm.String	String
Multiple lines of text	Edm.String	String
Choice (menu to choose from)	Edm.String	String
Number (1, 1.0, 100)	Edm.Double	Double
Currency ($, ¥, €)	Edm.Double	Double
Date and Time	Edm.DateTime	DateTime
Lookup (information already on this site)	Edm.Int32	Int32
Yes/No (check box)	Edm.Boolean	Boolean
Person or Group	Edm.Int32	Int32

SHAREPOINT ONLINE DATA TYPE	ODATA DATA TYPE	INTERIM DATA TYPE
Hyperlink or Picture	Edm.String	String
Calculated (calculation based on other columns)	Edm.String / Edm.Double / Edm.DateTime / Edm.Boolean	String / Double / DateTime / Boolean
Attachment	Not supported	
Task Outcome	Not supported	
External Data	Not supported	
Managed Metadata	Not supported	

Copy file from SharePoint Online

You can copy file from SharePoint Online by using **Web activity** to authenticate and grab access token from SPO, then passing to subsequent **Copy activity** to copy data with **HTTP connector as source**.



1. Follow the Prerequisites section to create AAD application and grant permission to SharePoint Online.
2. Create a **Web Activity** to get the access token from SharePoint Online:
   - **URL**: `https://accounts.accesscontrol.windows.net/[Tenant-ID]/tokens/OAuth/2`. Replace the tenant ID.
   - **Method**: POST
   - **Headers**:
     - Content-Type: application/x-www-form-urlencoded
   - **Body**:
     `grant_type=client_credentials&client_id=[Client-ID]@[Tenant-ID]&client_secret=[Client-Secret]&resource=00000003-0000-0ff1-ce00-000000000000/[Tenant-Name].sharepoint.com@[Tenant-ID]`
     . Replace the client ID (application ID), client secret (application key), tenant ID, and tenant name (of the SharePoint tenant).

   Caution

   Set the Secure Output option to true in Web activity to prevent the token value from being logged in plain text. Any further activities that consume this value should have their Secure Input option set to true.
3. Chain with a **Copy activity** with HTTP connector as source to copy SharePoint Online file content:
   - HTTP linked service:
     - **Base URL**:
       `https://[site-url]/_api/web/GetFileByServerRelativeUrl('[relative-path-to-file]')/$value`.
       Replace the site URL and relative path to file. Make sure to include the SharePoint site URL along with the Domain name, such as
       `https://[sharepoint-domain-name].sharepoint.com/sites/[sharepoint-site]/_api/web/GetFileByServerRelativeUrl('/sites/[sharepoint-site]/[relative-path-to-file]')/$value`
       .
     - **Authentication type:** Anonymous *(to use the Bearer token configured in copy activity source*

later)

- Dataset: choose the format you want. To copy file as-is, select "Binary" type.
- Copy activity source:
  - **Request method**: GET
  - **Additional header**: use the following expression `@{concat('Authorization: Bearer ', activity('<Web-activity-name>').output.access_token)}`, which uses the Bearer token generated by the upstream Web activity as authorization header. Replace the Web activity name.
- Configure the copy activity sink as usual.

> **NOTE**
>
> Even if an Azure AD application has `FullControl` permissions on SharePoint Online, you can't copy files from document libraries with IRM enabled.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores that Copy Activity supports as sources and sinks, see Supported data stores and formats.

Copy data from Shopify using Azure Data Factoryor Synapse Analytics (Preview)

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Shopify. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Shopify connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

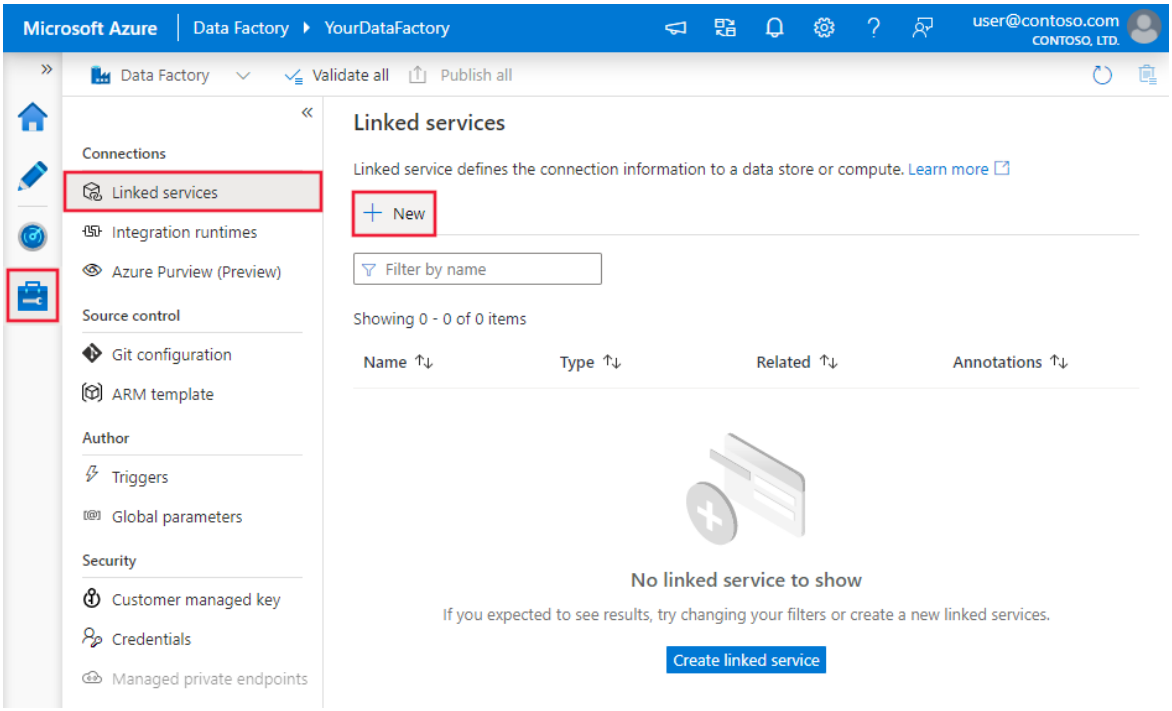
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Shopify using UI

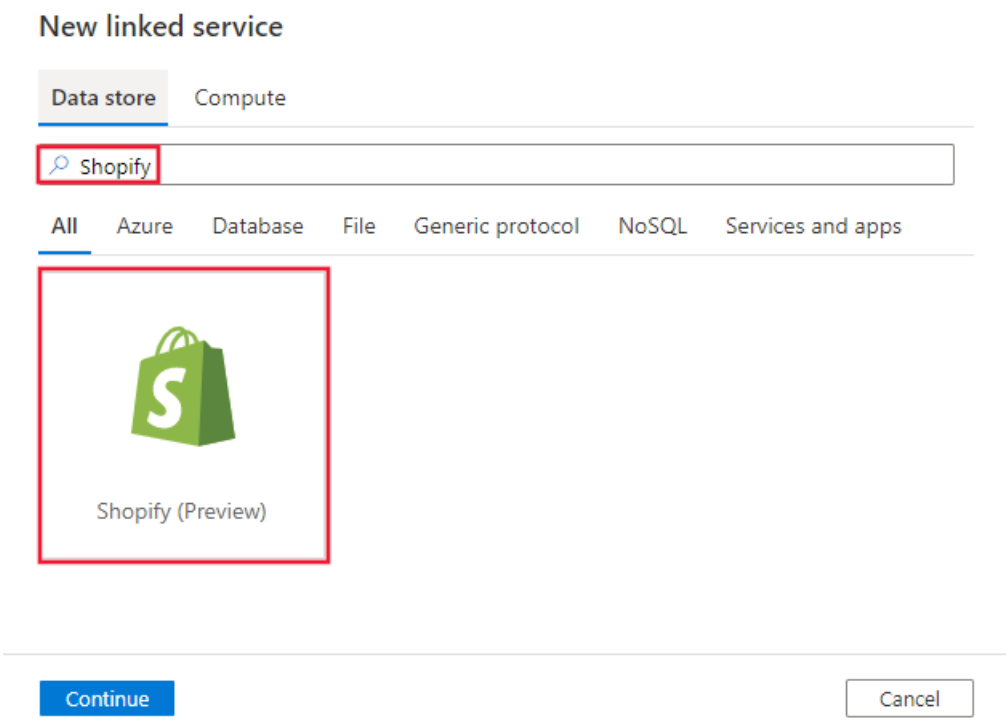
Use the following steps to create a linked service to Shopify in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

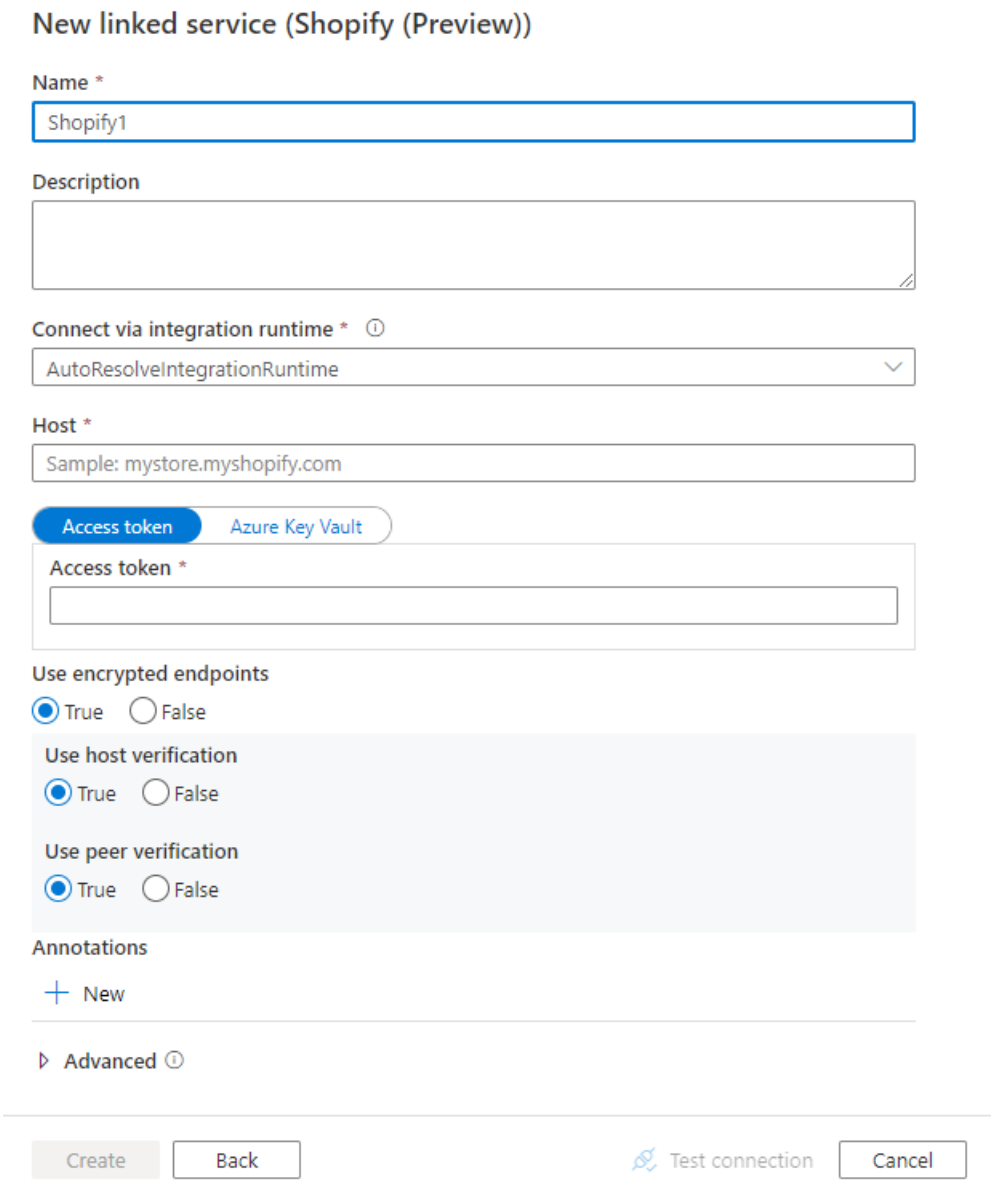
- Azure Data Factory
- Azure Synapse



2. Search for Shopify and select the Shopify connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Shopify connector.

Linked service properties

The following properties are supported for Shopify linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Shopify**	Yes
host	The endpoint of the Shopify server. (that is, mystore.myshopify.com)	Yes

PROPERTY	DESCRIPTION	REQUIRED
accessToken	The API access token that can be used to access Shopify's data. The token does not expire if it is offline mode. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example:

```
{
    "name": "ShopifyLinkedService",
    "properties": {
        "type": "Shopify",
        "typeProperties": {
            "host" : "mystore.myshopify.com",
            "accessToken": {
                 "type": "SecureString",
                 "value": "<accessToken>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Shopify dataset.

To copy data from Shopify, set the type property of the dataset to **ShopifyObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **ShopifyObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "ShopifyDataset",
    "properties": {
        "type": "ShopifyObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Shopify linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Shopify source.

Shopify as source

To copy data from Shopify, set the source type in the copy activity to **ShopifySource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **ShopifySource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM "Products" WHERE Product_Id = '123'"` .	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromShopify",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Shopify input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "ShopifySource",
                "query": "SELECT * FROM \"Products\" WHERE Product_Id = '123'"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Transform data in Smartsheet (Preview) using Azure Data Factory or Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Data Flow to transform data in Smartsheet (Preview). To learn more, read the introductory article for Azure Data Factory or Azure Synapse Analytics.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Smartsheet connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Mapping data flow (source/-)	①

① Azure integration runtime ② Self-hosted integration runtime

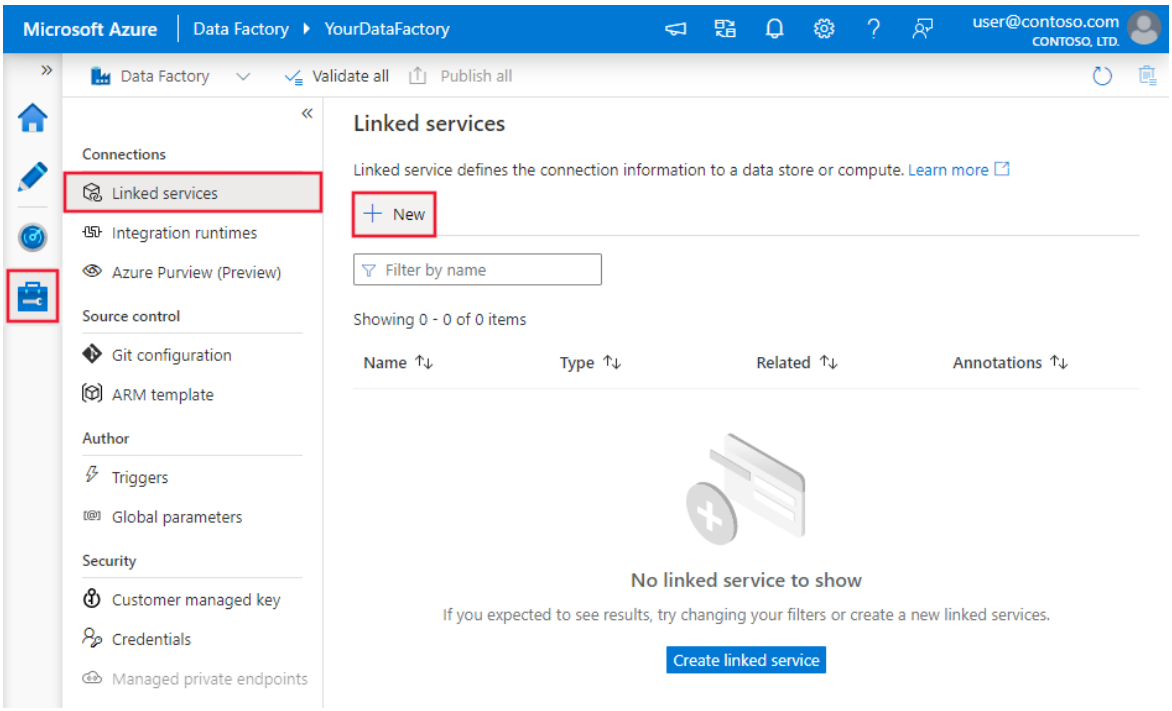
For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Create a Smartsheet linked service using UI

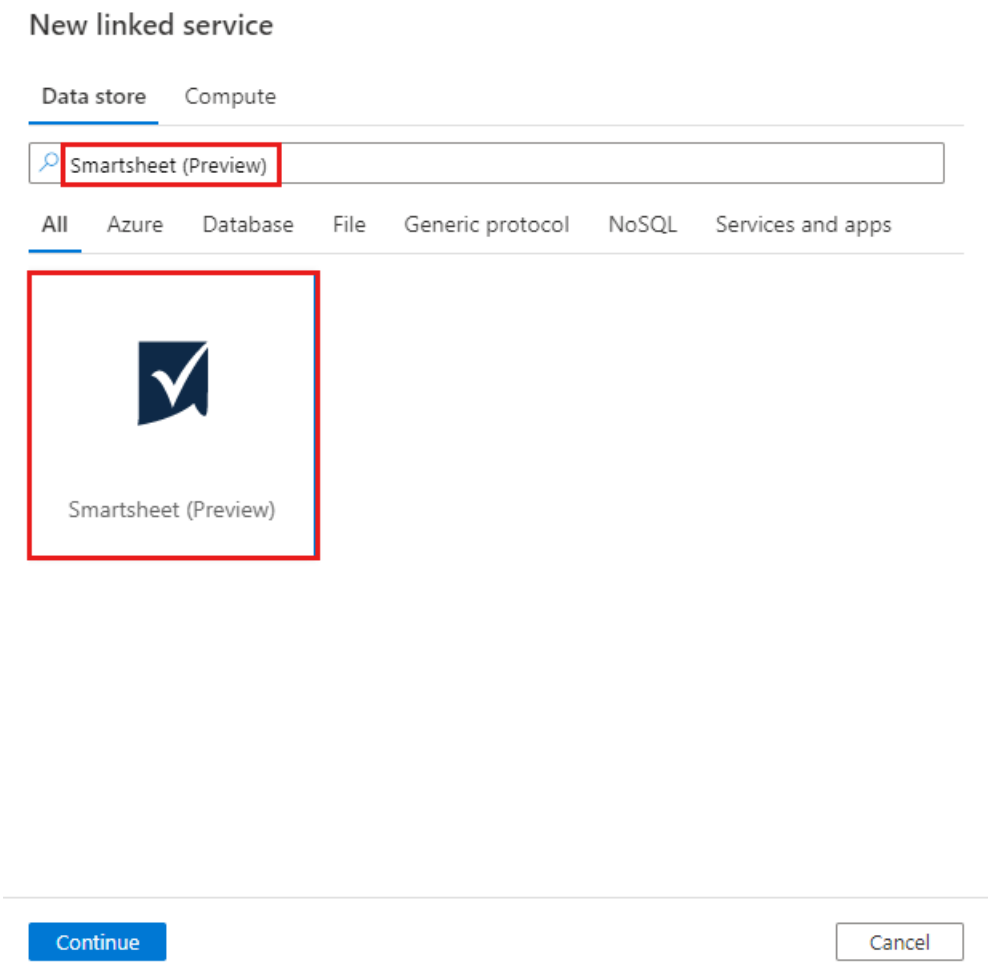
Use the following steps to create a Smartsheet linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then select New:

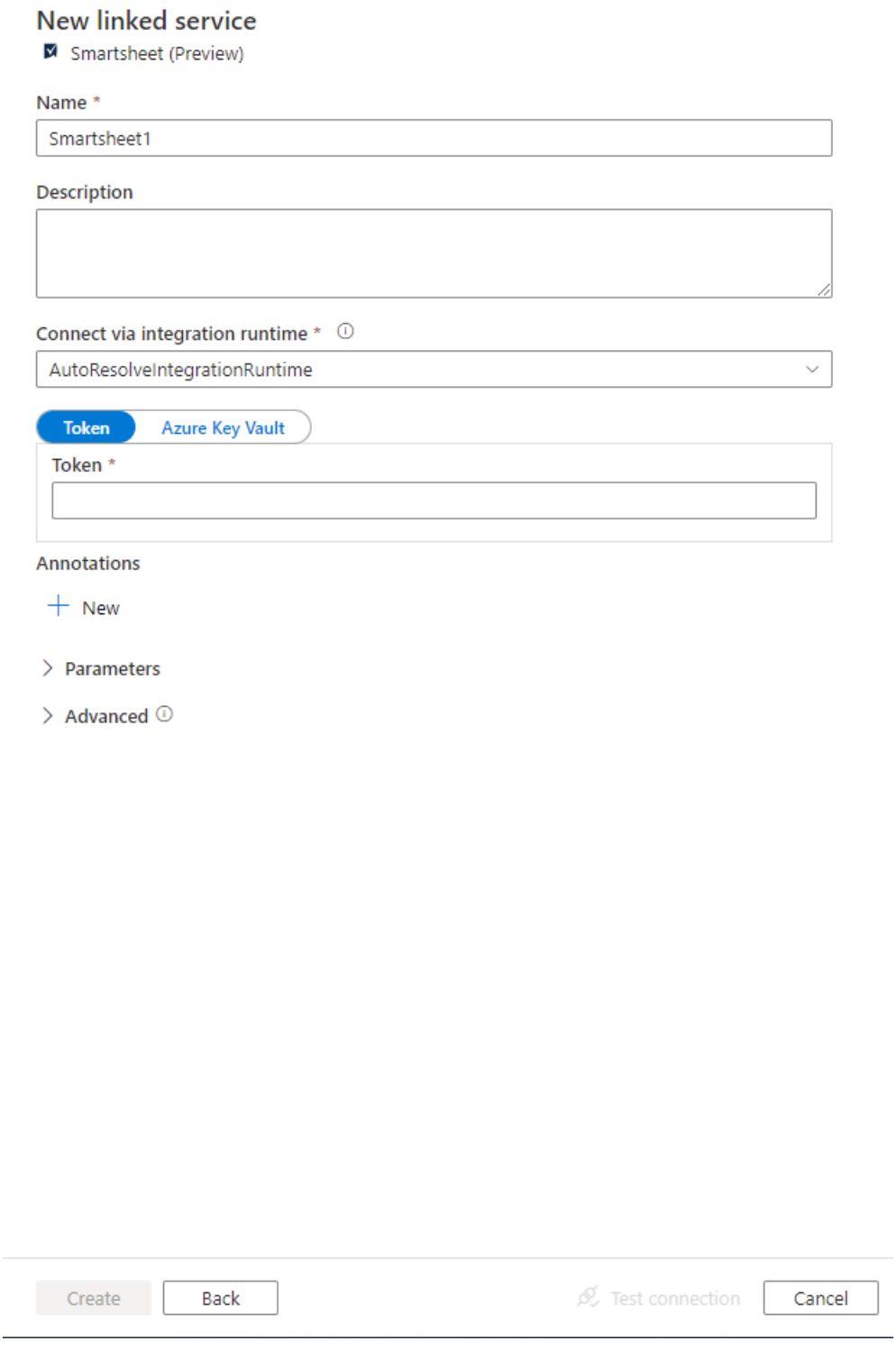
   - Azure Data Factory
   - Azure Synapse



2. Search for Smartsheet (Preview) and select the Smartsheet (Preview) connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide information about properties that are used to define Data Factory and Synapse pipeline entities specific to Smartsheet.

Linked service properties

The following properties are supported for the Smartsheet linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **Smartsheet**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
apiToken	Specify an API token for the Smartsheet. Mark this field as **SecureString** to store it securely. Or, you can reference a secret stored in Azure Key Vault.	Yes

Example:

```
{
    "name": "SmartsheetLinkedService",
    "properties": {
        "type": "Smartsheet",
        "typeProperties": {
            "apiToken": {
                "type": "SecureString",
                "value": "<API token>"
            }
        }
    }
}
```

Mapping data flow properties

When transforming data in mapping data flow, you can read tables from Smartsheet. For more information, see the source transformation in mapping data flows. You can only use an inline dataset as source type.

Source transformation

The below table lists the properties supported by Smartsheet source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Entity type	The type of the data asset in Smartsheet.	Yes when use inline mode	`sheets` `reports`	entityType
Entity name	The name of a sheet or a report in Smartsheet.	Yes when use inline mode	String	entityId

Smartsheet source script examples

```
source(allowSchemaDrift: true,
 validateSchema: false,
 store: 'smartsheet',
 format: 'rest',
 entityId: 'Sheet1',
 entityType: 'sheets') ~> SmartsheetSource
```

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy and transform data in Snowflake using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 15 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy activity in Azure Data Factory and Azure Synapse pipelines to copy data from and to Snowflake, and use Data Flow to transform data in Snowflake. For more information, see the introductory article for Data Factory or Azure Synapse Analytics.

Supported capabilities

This Snowflake connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Mapping data flow (source/sink)	①
Lookup activity	① ②
Script activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For the Copy activity, this Snowflake connector supports the following functions:

- Copy data from Snowflake that utilizes Snowflake's COPY into [location] command to achieve the best performance.
- Copy data to Snowflake that takes advantage of Snowflake's COPY into [table] command to achieve the best performance. It supports Snowflake on Azure.
- If a proxy is required to connect to Snowflake from a self-hosted Integration Runtime, you must configure the environment variables for HTTP_PROXY and HTTPS_PROXY on the Integration Runtime host.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it. Make sure to add the IP addresses that the self-hosted integration runtime uses to the allowed list.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allowed list.

The Snowflake account that is used for Source or Sink should have the necessary `USAGE` access on the Database and Read / Write access on Schema and the Tables/Views under it. In addition, it should also have `CREATE STAGE` on the schema to be able to create the External stage with SAS URI.

The following Account properties values must be set

PROPERTY	DESCRIPTION	REQUIRED	DEFAULT
REQUIRE_STORAGE_INTEGRATION_FOR_STAGE_CREATION	Specifies whether to require a storage integration object as cloud credentials when creating a named external stage (using CREATE STAGE) to access a private cloud storage location.	FALSE	FALSE
REQUIRE_STORAGE_INTEGRATION_FOR_STAGE_OPERATION	Specifies whether to require using a named external stage that references a storage integration object as cloud credentials when loading data from or unloading data to a private cloud storage location.	FALSE	FALSE

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

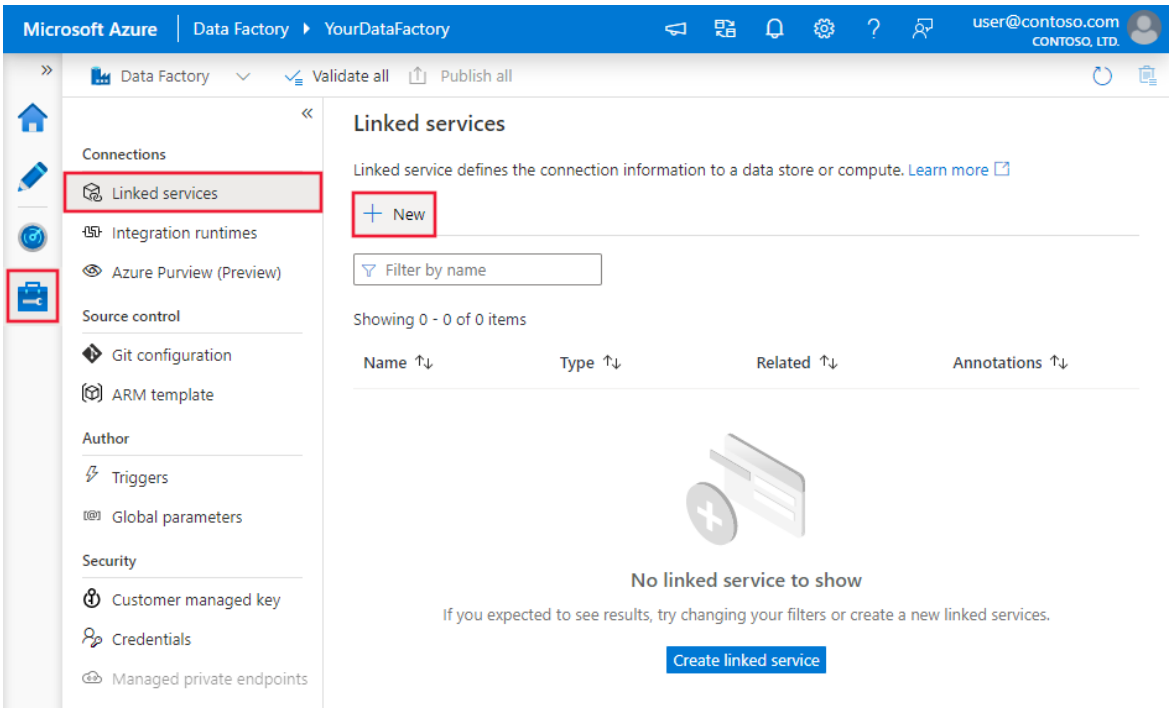
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Snowflake using UI

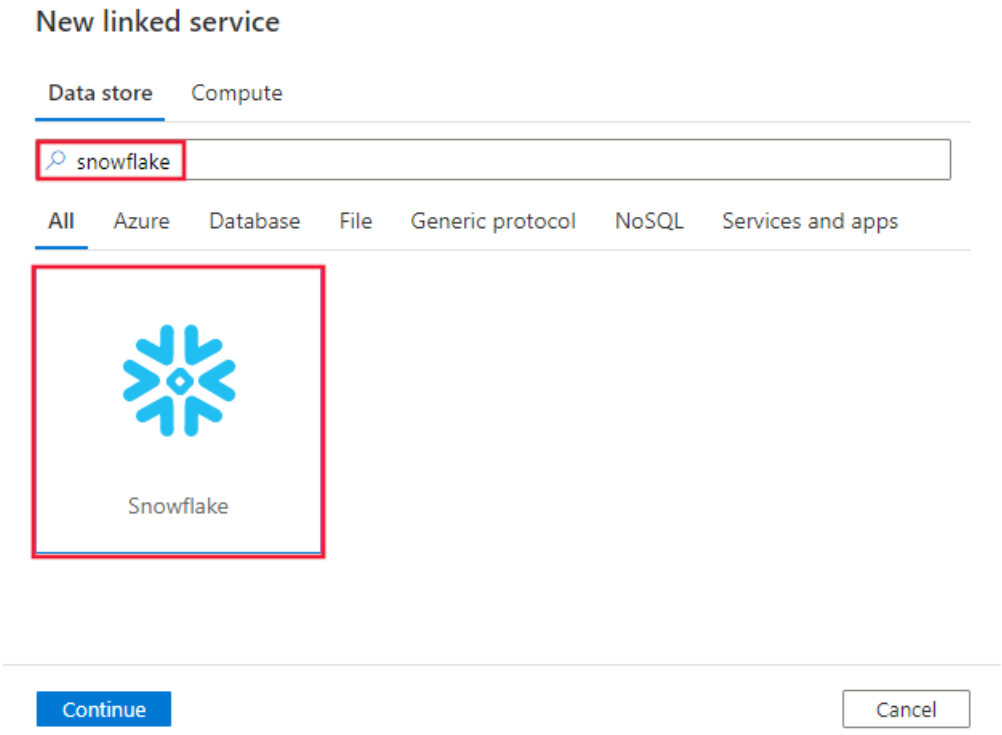
Use the following steps to create a linked service to Snowflake in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

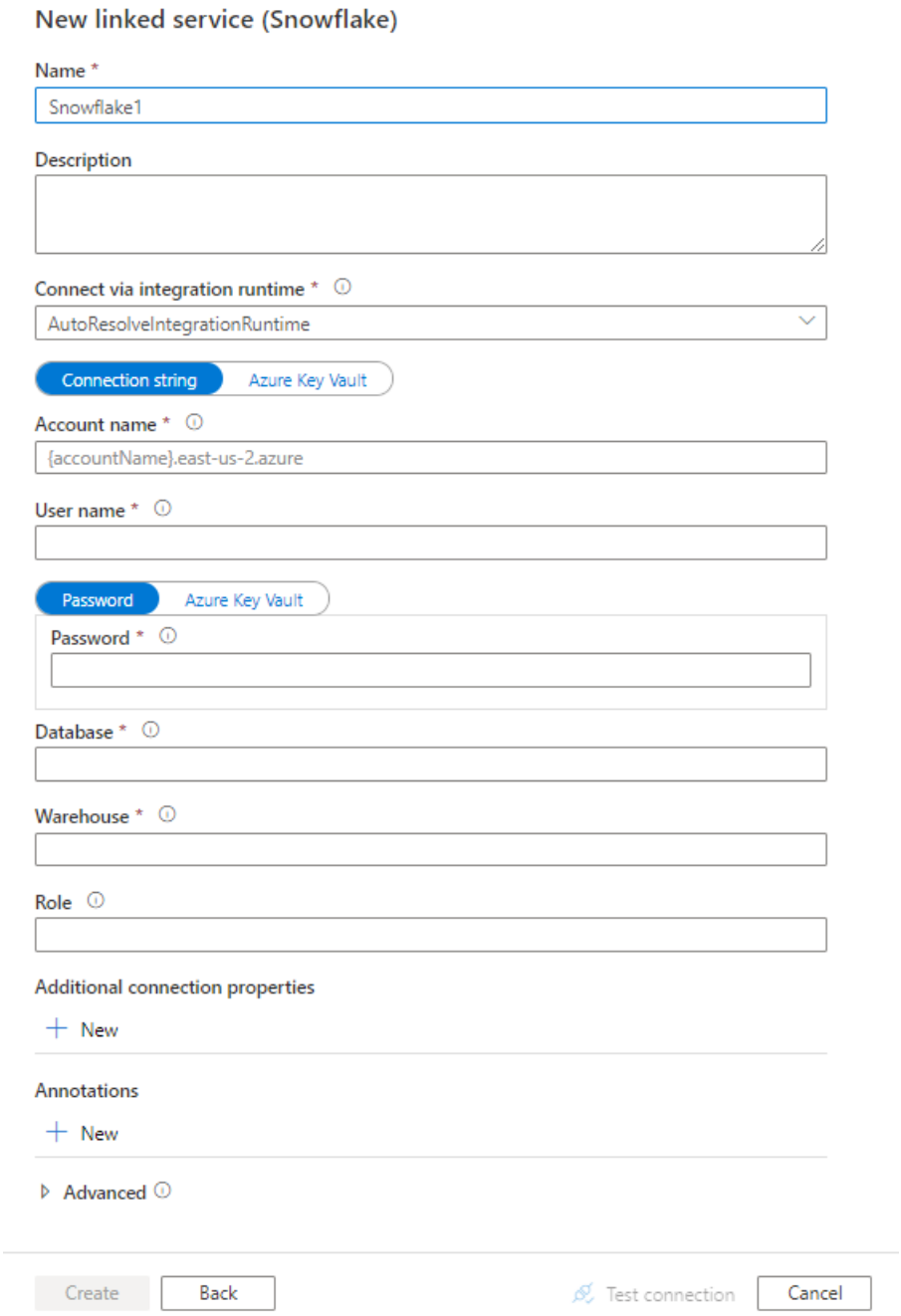
   - Azure Data Factory
   - Azure Synapse



2. Search for Snowflake and select the Snowflake connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that define entities specific to a Snowflake connector.

Linked service properties

The following properties are supported for a Snowflake-linked service.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **Snowflake**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	Specifies the information needed to connect to the Snowflake instance. You can choose to put password or entire connection string in Azure Key Vault. Refer to the examples below the table, as well as the Store credentials in Azure Key Vault article, for more details. Some typical settings: - **Account name:** The full account name of your Snowflake account (including additional segments that identify the region and cloud platform), e.g. xy12345.east-us-2.azure. - **User name:** The login name of the user for the connection. - **Password:** The password for the user. - **Database:** The default database to use once connected. It should be an existing database for which the specified role has privileges. - **Warehouse:** The virtual warehouse to use once connected. It should be an existing warehouse for which the specified role has privileges. - **Role:** The default access control role to use in the Snowflake session. The specified role should be an existing role that has already been assigned to the specified user. The default role is PUBLIC.	Yes
connectVia	The integration runtime that is used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime (if your data store is located in a private network). If not specified, it uses the default Azure integration runtime.	No

Example:

```
{
    "name": "SnowflakeLinkedService",
    "properties": {
        "type": "Snowflake",
        "typeProperties": {
            "connectionString": "jdbc:snowflake://<accountname>.snowflakecomputing.com/?user=
<username>&password=<password>&db=<database>&warehouse=<warehouse>&role=<myRole>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Password in Azure Key Vault:

```
{
    "name": "SnowflakeLinkedService",
    "properties": {
        "type": "Snowflake",
        "typeProperties": {
            "connectionString": "jdbc:snowflake://<accountname>.snowflakecomputing.com/?user=<username>&db=
<database>&warehouse=<warehouse>&role=<myRole>",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article.

The following properties are supported for the Snowflake dataset.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **SnowflakeTable**.	Yes
schema	Name of the schema. Note the schema name is case-sensitive.	No for source, yes for sink
table	Name of the table/view. Note the table name is case-sensitive.	No for source, yes for sink

Example:

```
{
    "name": "SnowflakeDataset",
    "properties": {
        "type": "SnowflakeTable",
        "typeProperties": {
            "schema": "<Schema name for your Snowflake database>",
            "table": "<Table name for your Snowflake database>"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "linkedServiceName": {
            "referenceName": "<name of linked service>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the Snowflake source and sink.

Snowflake as the source

Snowflake connector utilizes Snowflake's COPY into [location] command to achieve the best performance.

If sink data store and format are natively supported by the Snowflake COPY command, you can use the Copy activity to directly copy from Snowflake to sink. For details, see Direct copy from Snowflake. Otherwise, use built-in Staged copy from Snowflake.

To copy data from Snowflake, the following properties are supported in the Copy activity **source** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the Copy activity source must be set to **SnowflakeSource**.	Yes
query	Specifies the SQL query to read data from Snowflake. If the names of the schema, table and columns contain lower case, quote the object identifier in query e.g. `select * from "schema"."myTable"`. Executing stored procedure is not supported.	No
exportSettings	Advanced settings used to retrieve data from Snowflake. You can configure the ones supported by the COPY into command that the service will pass through when you invoke the statement.	No
Under `exportSettings`*:*		
type	The type of export command, set to **SnowflakeExportCopyCommand**.	Yes
additionalCopyOptions	Additional copy options, provided as a dictionary of key-value pairs. Examples: MAX_FILE_SIZE, OVERWRITE. For more information, see Snowflake Copy Options.	No
additionalFormatOptions	Additional file format options that are provided to COPY command as a dictionary of key-value pairs. Examples: DATE_FORMAT, TIME_FORMAT, TIMESTAMP_FORMAT. For more information, see Snowflake Format Type Options.	No

> **NOTE**
>
> Make sure you have permission to execute the following command and access the schema *INFORMATION_SCHEMA* and the table *COLUMNS*.
>
> - `COPY INTO <location>`

Direct copy from Snowflake

If your sink data store and format meet the criteria described in this section, you can use the Copy activity to directly copy from Snowflake to sink. The service checks the settings and fails the Copy activity run if the following criteria is not met:

- The **sink linked service** is **Azure Blob storage** with **shared access signature** authentication. If you want to directly copy data to Azure Data Lake Storage Gen2 in the following supported format, you can create an Azure Blob linked service with SAS authentication against your ADLS Gen2 account, to avoid using staged copy from Snowflake.
- The **sink data format** is of **Parquet**, **delimited text**, or **JSON** with the following configurations:
  - For **Parquet** format, the compression codec is **None**, **Snappy**, or **Lzo**.
  - For **delimited text** format:
    - `rowDelimiter` is **\r\n**, or any single character.
    - `compression` can be **no compression**, **gzip**, **bzip2**, or **deflate**.
    - `encodingName` is left as default or set to **utf-8**.
    - `quoteChar` is **double quote**, **single quote**, or **empty string** (no quote char).
  - For **JSON** format, direct copy only supports the case that source Snowflake table or query result only has single column and the data type of this column is **VARIANT**, **OBJECT**, or **ARRAY**.
    - `compression` can be **no compression**, **gzip**, **bzip2**, or **deflate**.
    - `encodingName` is left as default or set to **utf-8**.
    - `filePattern` in copy activity sink is left as default or set to **setOfObjects**.
- In copy activity source, `additionalColumns` is not specified.
- Column mapping is not specified.

Example:

```
"activities":[
    {
        "name": "CopyFromSnowflake",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Snowflake input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SnowflakeSource",
                "sqlReaderQuery": "SELECT * FROM MYTABLE",
                "exportSettings": {
                    "type": "SnowflakeExportCopyCommand",
                    "additionalCopyOptions": {
                        "MAX_FILE_SIZE": "64000000",
                        "OVERWRITE": true
                    },
                    "additionalFormatOptions": {
                        "DATE_FORMAT": "'MM/DD/YYYY'"
                    }
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Staged copy from Snowflake

When your sink data store or format is not natively compatible with the Snowflake COPY command, as mentioned in the last section, enable the built-in staged copy using an interim Azure Blob storage instance. The staged copy feature also provides you better throughput. The service exports data from Snowflake into staging storage, then copies the data to sink, and finally cleans up your temporary data from the staging storage. See Staged copy for details about copying data by using staging.

To use this feature, create an Azure Blob storage linked service that refers to the Azure storage account as the interim staging. Then specify the `enableStaging` and `stagingSettings` properties in the Copy activity.

> **NOTE**
>
> The staging Azure Blob storage linked service must use shared access signature authentication, as required by the Snowflake COPY command. Make sure you grant proper access permission to Snowflake in the staging Azure Blob storage. To learn more about this, see this article.

Example:

```
"activities":[
    {
        "name": "CopyFromSnowflake",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Snowflake input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SnowflakeSource",
                "sqlReaderQuery": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            },
            "enableStaging": true,
            "stagingSettings": {
                "linkedServiceName": {
                    "referenceName": "MyStagingBlob",
                    "type": "LinkedServiceReference"
                },
                "path": "mystagingpath"
            }
        }
    }
]
```

Snowflake as sink

Snowflake connector utilizes Snowflake's COPY into [table] command to achieve the best performance. It supports writing data to Snowflake on Azure.

If source data store and format are natively supported by Snowflake COPY command, you can use the Copy activity to directly copy from source to Snowflake. For details, see Direct copy to Snowflake. Otherwise, use built-in Staged copy to Snowflake.

To copy data to Snowflake, the following properties are supported in the Copy activity **sink** section.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the Copy activity sink, set to **SnowflakeSink**.	Yes
preCopyScript	Specify a SQL query for the Copy activity to run before writing data into Snowflake in each run. Use this property to clean up the preloaded data.	No

PROPERTY	DESCRIPTION	REQUIRED
importSettings	Advanced settings used to write data into Snowflake. You can configure the ones supported by the COPY into command that the service will pass through when you invoke the statement.	No
Under `importSettings`*:*		
type	The type of import command, set to **SnowflakeImportCopyCommand**.	Yes
additionalCopyOptions	Additional copy options, provided as a dictionary of key-value pairs. Examples: ON_ERROR, FORCE, LOAD_UNCERTAIN_FILES. For more information, see Snowflake Copy Options.	No
additionalFormatOptions	Additional file format options provided to the COPY command, provided as a dictionary of key-value pairs. Examples: DATE_FORMAT, TIME_FORMAT, TIMESTAMP_FORMAT. For more information, see Snowflake Format Type Options.	No

NOTE

Make sure you have permission to execute the following command and access the schema *INFORMATION_SCHEMA* and the table *COLUMNS*.

- `SELECT CURRENT_REGION()`
- `COPY INTO <table>`
- `SHOW REGIONS`
- `CREATE OR REPLACE STAGE`
- `DROP STAGE`

Direct copy to Snowflake

If your source data store and format meet the criteria described in this section, you can use the Copy activity to directly copy from source to Snowflake. The service checks the settings and fails the Copy activity run if the following criteria is not met:

- The **source linked service** is **Azure Blob storage** with **shared access signature** authentication. If you want to directly copy data from Azure Data Lake Storage Gen2 in the following supported format, you can create an Azure Blob linked service with SAS authentication against your ADLS Gen2 account, to avoid using staged copy to Snowflake..
- The **source data format** is **Parquet**, **Delimited text**, or **JSON** with the following configurations:
  - For **Parquet** format, the compression codec is **None**, or **Snappy**.
  - For **delimited text** format:
    - `rowDelimiter` is **\r\n**, or any single character. If row delimiter is not "\r\n", `firstRowAsHeader`

need to be **false**, and `skipLineCount` is not specified.

- `compression` can be **no compression**, **gzip**, **bzip2**, or **deflate**.
- `encodingName` is left as default or set to "UTF-8", "UTF-16", "UTF-16BE", "UTF-32", "UTF-32BE", "BIG5", "EUC-JP", "EUC-KR", "GB18030", "ISO-2022-JP", "ISO-2022-KR", "ISO-8859-1", "ISO-8859-2", "ISO-8859-5", "ISO-8859-6", "ISO-8859-7", "ISO-8859-8", "ISO-8859-9", "WINDOWS-1250", "WINDOWS-1251", "WINDOWS-1252", "WINDOWS-1253", "WINDOWS-1254", "WINDOWS-1255".
- `quoteChar` is **double quote**, **single quote**, or **empty string** (no quote char).

- For **JSON** format, direct copy only supports the case that sink Snowflake table only has single column and the data type of this column is **VARIANT**, **OBJECT**, or **ARRAY**.
  - `compression` can be **no compression**, **gzip**, **bzip2**, or **deflate**.
  - `encodingName` is left as default or set to **utf-8**.
  - Column mapping is not specified.

- In the Copy activity source:
  - `additionalColumns` is not specified.
  - If your source is a folder, `recursive` is set to true.
  - `prefix`, `modifiedDateTimeStart`, `modifiedDateTimeEnd`, and `enablePartitionDiscovery` are not specified.

Example:

```
"activities":[
    {
        "name": "CopyToSnowflake",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Snowflake output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "SnowflakeSink",
                "importSettings": {
                    "type": "SnowflakeImportCopyCommand",
                    "copyOptions": {
                        "FORCE": "TRUE",
                        "ON_ERROR": "SKIP_FILE",
                    },
                    "fileFormatOptions": {
                        "DATE_FORMAT": "YYYY-MM-DD",
                    }
                }
            }
        }
    }
]
```

Staged copy to Snowflake

When your source data store or format is not natively compatible with the Snowflake COPY command, as mentioned in the last section, enable the built-in staged copy using an interim Azure Blob storage instance. The staged copy feature also provides you better throughput. The service automatically converts the data to meet the data format requirements of Snowflake. It then invokes the COPY command to load data into Snowflake. Finally, it cleans up your temporary data from the blob storage. See Staged copy for details about copying data using staging.

To use this feature, create an Azure Blob storage linked service that refers to the Azure storage account as the interim staging. Then specify the `enableStaging` and `stagingSettings` properties in the Copy activity.

> **NOTE**
>
> The staging Azure Blob storage linked service need to use shared access signature authentication as required by the Snowflake COPY command.

Example:

```
"activities":[
    {
        "name": "CopyToSnowflake",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Snowflake output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "SnowflakeSink"
            },
            "enableStaging": true,
            "stagingSettings": {
                "linkedServiceName": {
                    "referenceName": "MyStagingBlob",
                    "type": "LinkedServiceReference"
                },
                "path": "mystagingpath"
            }
        }
    }
]
```

Mapping data flow properties

When transforming data in mapping data flow, you can read from and write to tables in Snowflake. For more information, see the source transformation and sink transformation in mapping data flows. You can choose to use a Snowflake dataset or an inline dataset as source and sink type.

Source transformation

The below table lists the properties supported by Snowflake source. You can edit these properties in the **Source options** tab. The connector utilizes Snowflake internal data transfer.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Table	If you select Table as input, data flow will fetch all the data from the table specified in the Snowflake dataset or in the source options when using inline dataset.	No	String	*(for inline dataset only)* tableName schemaName
Query	If you select Query as input, enter a query to fetch data from Snowflake. This setting overrides any table that you've chosen in dataset. If the names of the schema, table and columns contain lower case, quote the object identifier in query e.g. `select * from "schema"."myTable"`.	No	String	query

Snowflake source script examples

When you use Snowflake dataset as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 query: 'select * from MYTABLE',
 format: 'query') ~> SnowflakeSource
```

If you use inline dataset, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 format: 'query',
 query: 'select * from MYTABLE',
 store: 'snowflake') ~> SnowflakeSource
```

Sink transformation

The below table lists the properties supported by Snowflake sink. You can edit these properties in the **Settings** tab. When using inline dataset, you will see additional settings, which are the same as the properties described in dataset properties section. The connector utilizes Snowflake internal data transfer.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Update method	Specify what operations are allowed on your Snowflake destination. To update, upsert, or delete rows, an Alter row transformation is required to tag rows for those actions.	Yes	`true` or `false`	deletable insertable updateable upsertable
Key columns	For updates, upserts and deletes, a key column or columns must be set to determine which row to alter.	No	Array	keys
Table action	Determines whether to recreate or remove all rows from the destination table prior to writing. - **None**: No action will be done to the table. - **Recreate**: The table will get dropped and recreated. Required if creating a new table dynamically. - **Truncate**: All rows from the target table will get removed.	No	`true` or `false`	recreate truncate

Snowflake sink script examples

When you use Snowflake dataset as sink type, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
 validateSchema: false,
 deletable:true,
 insertable:true,
 updateable:true,
 upsertable:false,
 keys:['movieId'],
 format: 'table',
 skipDuplicateMapInputs: true,
 skipDuplicateMapOutputs: true) ~> SnowflakeSink
```

If you use inline dataset, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
 validateSchema: false,
 format: 'table',
 tableName: 'table',
 schemaName: 'schema',
 deletable: true,
 insertable: true,
 updateable: true,
 upsertable: false,
 store: 'snowflake',
 skipDuplicateMapInputs: true,
 skipDuplicateMapOutputs: true) ~> SnowflakeSink
```

Lookup activity properties

For more information about the properties, see Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by Copy activity, see supported data stores and formats.

Copy data from Spark using Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Spark. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Spark connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell

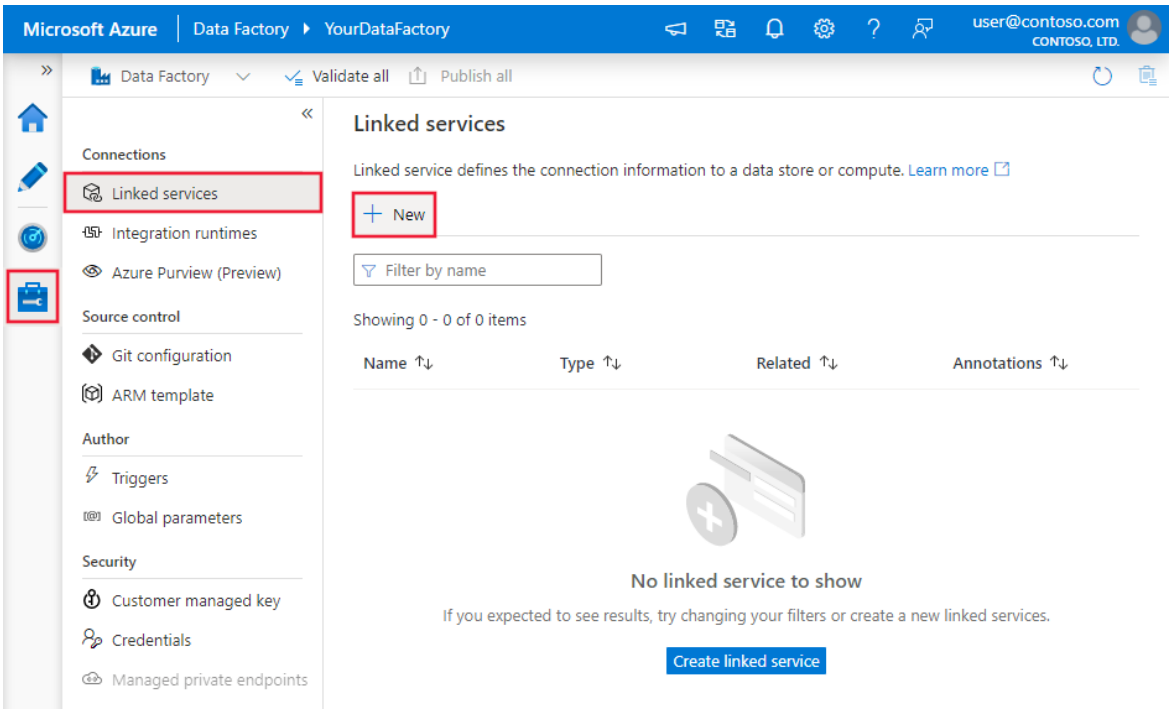
- The REST API
- The Azure Resource Manager template

Create a linked service to Spark using UI

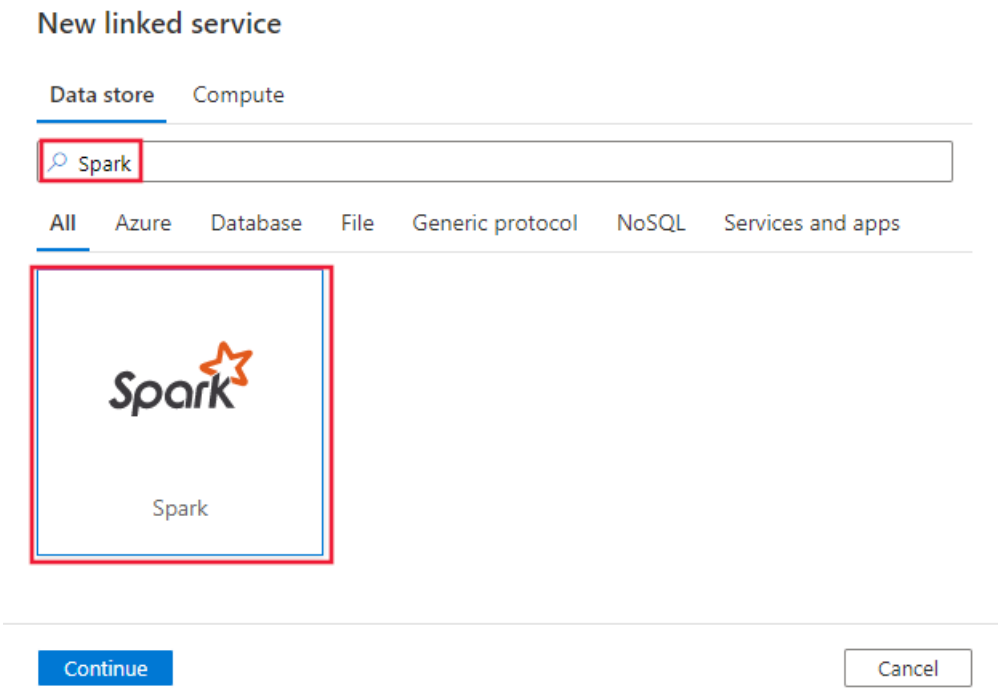
Use the following steps to create a linked service to Spark in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

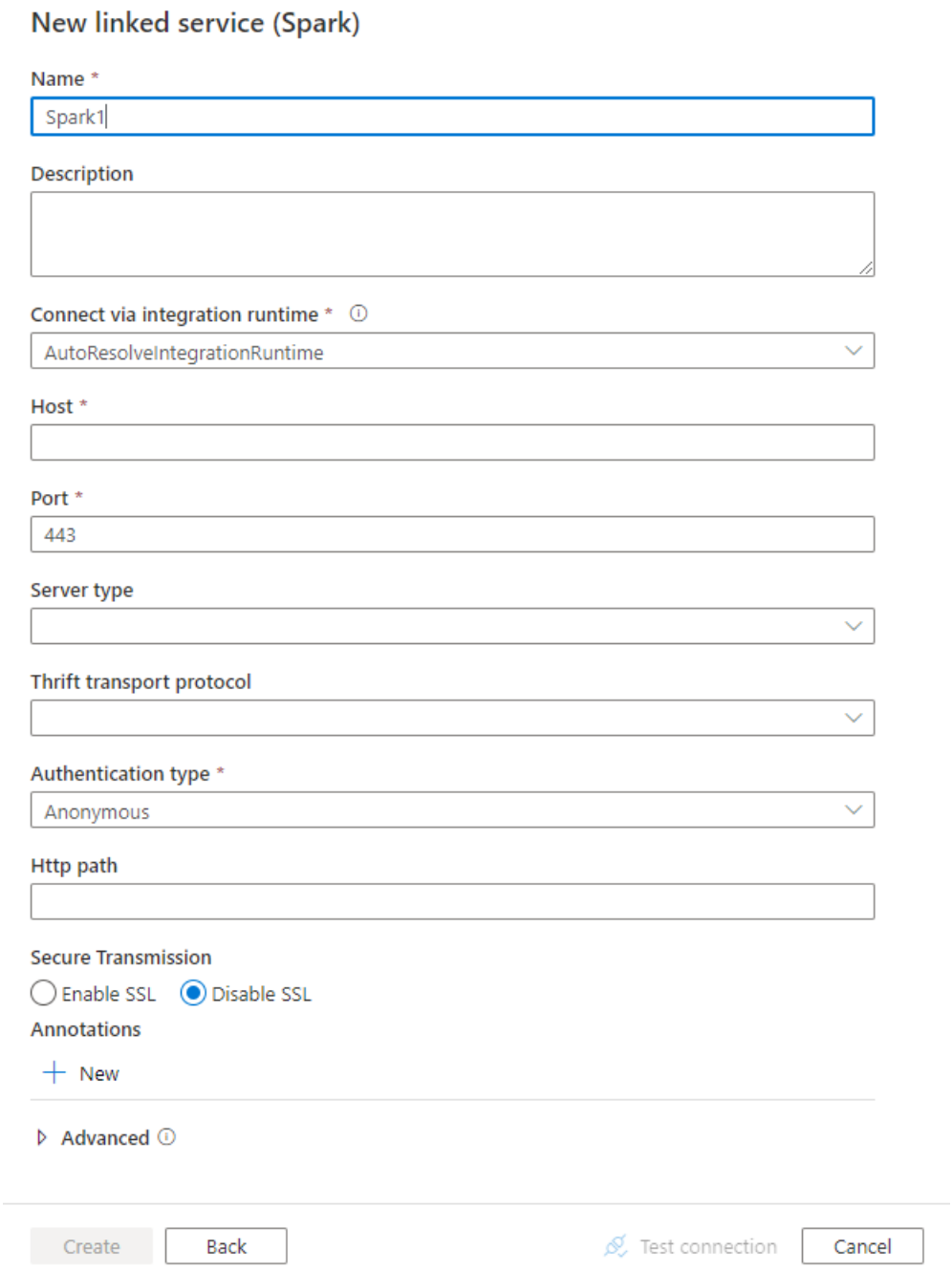
   - Azure Data Factory
   - Azure Synapse



2. Search for Spark and select the Spark connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Spark connector.

Linked service properties

The following properties are supported for Spark linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Spark**	Yes
host	IP address or host name of the Spark server	Yes

PROPERTY	DESCRIPTION	REQUIRED
port	The TCP port that the Spark server uses to listen for client connections. If you connect to Azure HDInsights, specify port as 443.	Yes
serverType	The type of Spark server. Allowed values are: **SharkServer**, **SharkServer2**, **SparkThriftServer**	No
thriftTransportProtocol	The transport protocol to use in the Thrift layer. Allowed values are: **Binary**, **SASL**, **HTTP**	No
authenticationType	The authentication method used to access the Spark server. Allowed values are: **Anonymous**, **Username**, **UsernameAndPassword**, **WindowsAzureHDInsightService**	Yes
username	The user name that you use to access Spark Server.	No
password	The password corresponding to the user. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	No
httpPath	The partial URL corresponding to the Spark server.	No
enableSsl	Specifies whether the connections to the server are encrypted using TLS. The default value is false.	No
trustedCertPath	The full path of the .pem file containing trusted CA certificates for verifying the server when connecting over TLS. This property can only be set when using TLS on self-hosted IR. The default value is the cacerts.pem file installed with the IR.	No
useSystemTrustStore	Specifies whether to use a CA certificate from the system trust store or from a specified PEM file. The default value is false.	No
allowHostNameCNMismatch	Specifies whether to require a CA-issued TLS/SSL certificate name to match the host name of the server when connecting over TLS. The default value is false.	No

PROPERTY	DESCRIPTION	REQUIRED
allowSelfSignedServerCert	Specifies whether to allow self-signed certificates from the server. The default value is false.	No
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "SparkLinkedService",
    "properties": {
        "type": "Spark",
        "typeProperties": {
            "host" : "<cluster>.azurehdinsight.net",
            "port" : "<port>",
            "authenticationType" : "WindowsAzureHDInsightService",
            "username" : "<username>",
            "password": {
                 "type": "SecureString",
                 "value": "<password>"
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Spark dataset.

To copy data from Spark, set the type property of the dataset to **SparkObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **SparkObject**	Yes
schema	Name of the schema.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table with schema. This property is supported for backward compatibility. Use `schema` and `table` for new workload.	No (if "query" in activity source is specified)

Example

```
{
    "name": "SparkDataset",
    "properties": {
        "type": "SparkObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Spark linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Spark source.

Spark as source

To copy data from Spark, set the source type in the copy activity to **SparkSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **SparkSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromSpark",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Spark input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SparkSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy and transform data to and from SQL Server by using Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 28 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the copy activity in Azure Data Factory and Azure Synapse pipelines to copy data from and to SQL Server database and use Data Flow to transform data in SQL Server database. To learn more read the introductory article for Azure Data Factory or Azure Synapse Analytics.

Supported capabilities

This SQL Server connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/sink)	① ②
Mapping data flow (source/sink)	①
Lookup activity	① ②
GetMetadata activity	① ②
Script activity	① ②
Stored procedure activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources or sinks by the copy activity, see the Supported data stores table.

Specifically, this SQL Server connector supports:

- SQL Server version 2005 and above.
- Copying data by using SQL or Windows authentication.
- As a source, retrieving data by using a SQL query or a stored procedure. You can also choose to parallel copy from SQL Server source, see the Parallel copy from SQL database section for details.
- As a sink, automatically creating destination table if not exists based on the source schema; appending data to a table or invoking a stored procedure with custom logic during copy.

SQL Server Express LocalDB is not supported.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

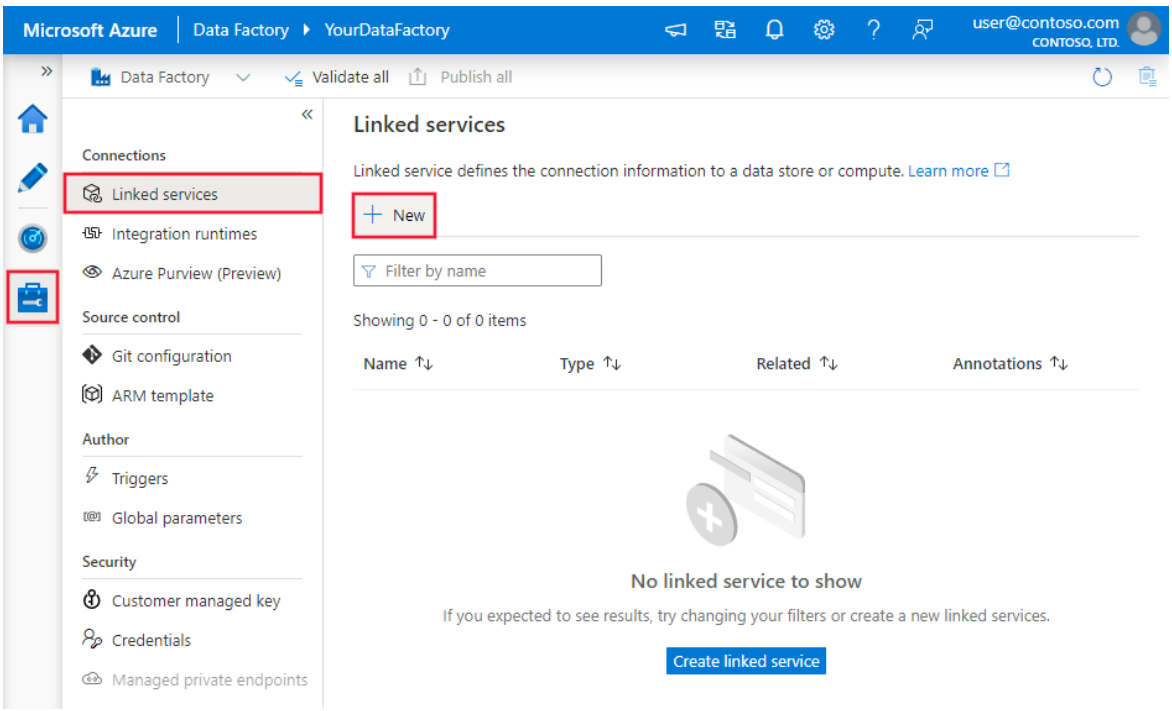
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a SQL Server linked service using UI

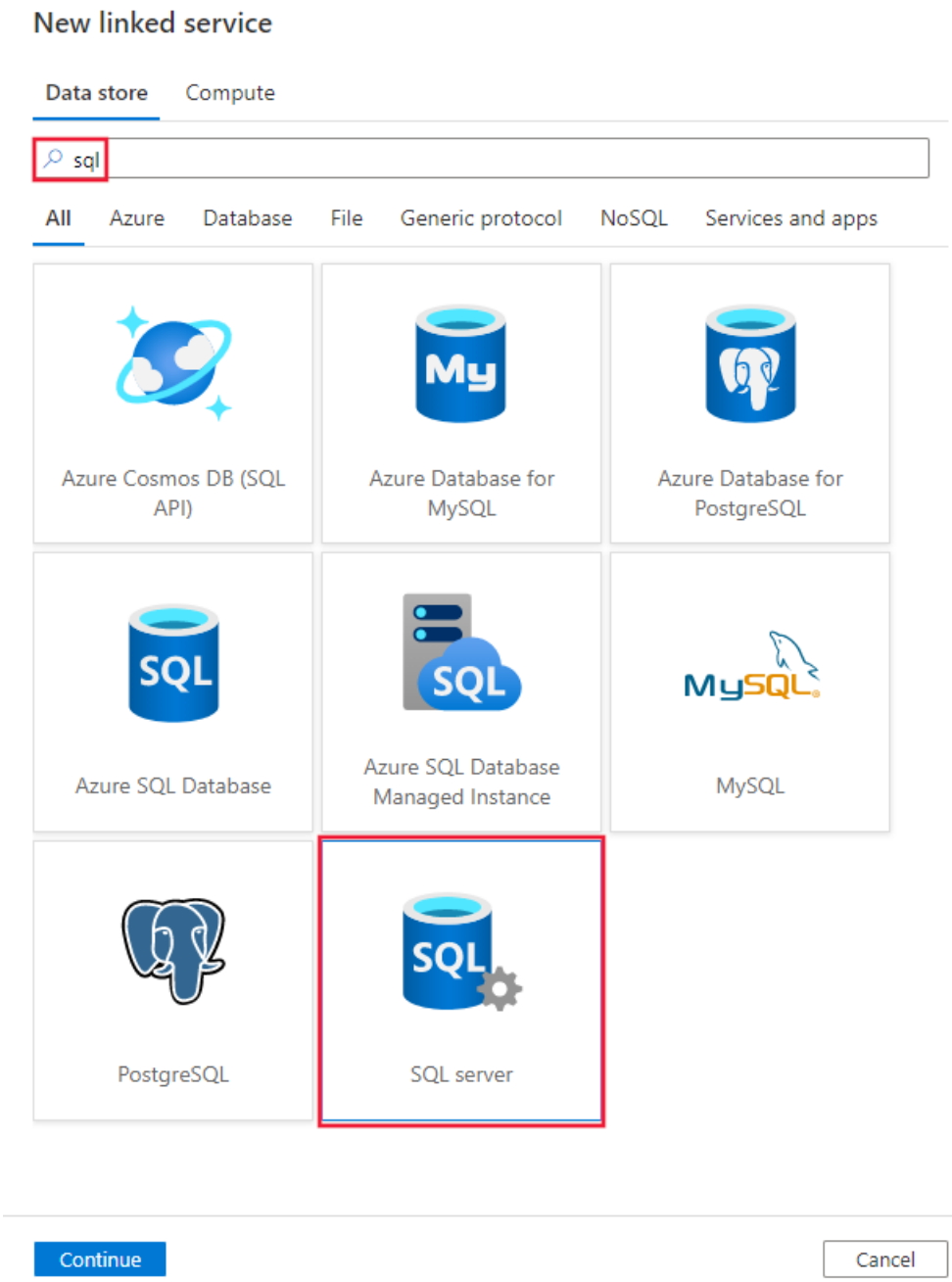
Use the following steps to create a SQL Server linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

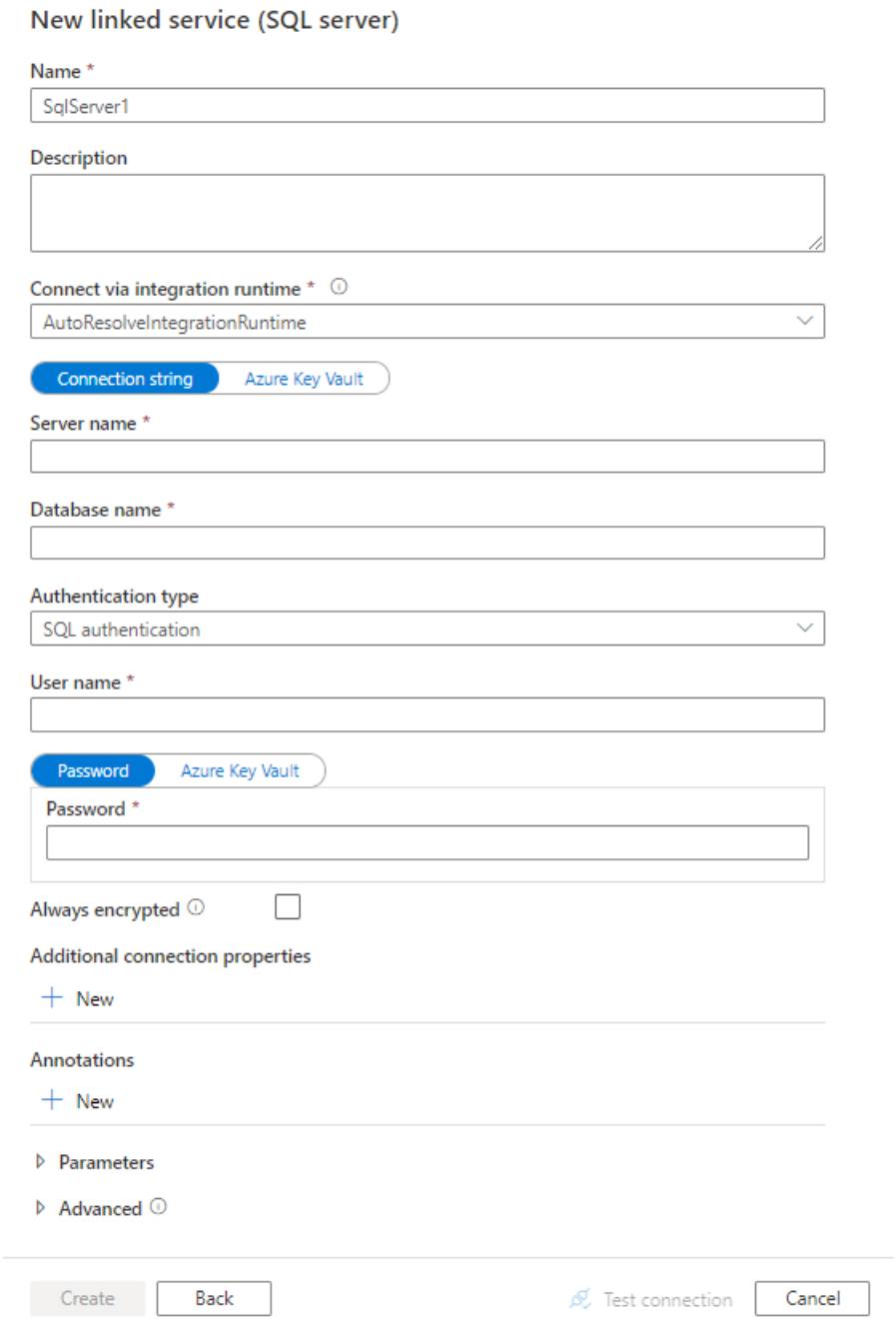
   - Azure Data Factory
   - Azure Synapse



2. Search for SQL and select the SQL Server connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory and Synapse pipeline entities specific to the SQL Server database connector.

Linked service properties

This SQL server connector supports the following authentication types. See the corresponding sections for details.

- SQL authentication
- Windows authentication

> **TIP**
>
> If you hit an error with the error code "UserErrorFailedToConnectToSqlServer" and a message like "The session limit for the database is XXX and has been reached," add `Pooling=false` to your connection string and try again.

SQL authentication

To use SQL authentication, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **SqlServer**.	Yes
connectionString	Specify **connectionString** information that's needed to connect to the SQL Server database. Specify a login name as your user name, and ensure the database that you want to connect is mapped to this login. Refer to the following samples.	Yes
password	If you want to put a password in Azure Key Vault, pull the `password` configuration out of the connection string. For more information, see the JSON example following the table and Store credentials in Azure Key Vault.	No
alwaysEncryptedSettings	Specify **alwaysencryptedsettings** information that's needed to enable Always Encrypted to protect sensitive data stored in SQL server by using either managed identity or service principal. For more information, see the JSON example following the table and Using Always Encrypted section. If not specified, the default always encrypted setting is disabled.	No
connectVia	This integration runtime is used to connect to the data store. Learn more from Prerequisites section. If not specified, the default Azure integration runtime is used.	No

Example: Use SQL authentication

```
{
    "name": "SqlServerLinkedService",
    "properties": {
        "type": "SqlServer",
        "typeProperties": {
            "connectionString": "Data Source=<servername>\\<instance name if using named instance>;Initial
Catalog=<databasename>;Integrated Security=False;User ID=<username>;Password=<password>;"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Use SQL authentication with a password in Azure Key Vault

```
{
    "name": "SqlServerLinkedService",
    "properties": {
        "type": "SqlServer",
        "typeProperties": {
            "connectionString": "Data Source=<servername>\\<instance name if using named instance>;Initial
Catalog=<databasename>;Integrated Security=False;User ID=<username>;",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Use Always Encrypted

```
{
    "name": "SqlServerLinkedService",
    "properties": {
        "type": "SqlServer",
        "typeProperties": {
            "connectionString": "Data Source=<servername>\\<instance name if using named instance>;Initial
Catalog=<databasename>;Integrated Security=False;User ID=<username>;Password=<password>;"
        },
        "alwaysEncryptedSettings": {
            "alwaysEncryptedAkvAuthType": "ServicePrincipal",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "<service principal key>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Windows authentication

To use Windows authentication, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **SqlServer**.	Yes
connectionString	Specify **connectionString** information that's needed to connect to the SQL Server database. Refer to the following samples.	

PROPERTY	DESCRIPTION	REQUIRED
userName	Specify a user name. An example is **domainname\username**.	Yes
password	Specify a password for the user account you specified for the user name. Mark this field as **SecureString** to store it securely. Or, you can reference a secret stored in Azure Key Vault.	Yes
alwaysEncryptedSettings	Specify **alwaysencryptedsettings** information that's needed to enable Always Encrypted to protect sensitive data stored in SQL server by using either managed identity or service principal. For more information, see Using Always Encrypted section. If not specified, the default always encrypted setting is disabled.	No
connectVia	This integration runtime is used to connect to the data store. Learn more from Prerequisites section. If not specified, the default Azure integration runtime is used.	No

> **NOTE**
> Windows authentication is not supported in data flow.

Example: Use Windows authentication

```
{
    "name": "SqlServerLinkedService",
    "properties": {
        "type": "SqlServer",
        "typeProperties": {
            "connectionString": "Data Source=<servername>\\<instance name if using named instance>;Initial
Catalog=<databasename>;Integrated Security=True;",
            "userName": "<domain\\username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: Use Windows authentication with a password in Azure Key Vault

```
{
    "name": "SqlServerLinkedService",
    "properties": {
        "annotations": [],
        "type": "SqlServer",
        "typeProperties": {
            "connectionString": "Data Source=<servername>\\<instance name if using named instance>;Initial
Catalog=<databasename>;Integrated Security=True;",
            "userName": "<domain\\username>",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by the SQL Server dataset.

To copy data from and to a SQL Server database, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **SqlServerTable**.	Yes
schema	Name of the schema.	No for source, Yes for sink
table	Name of the table/view.	No for source, Yes for sink
tableName	Name of the table/view with schema. This property is supported for backward compatibility. For new workload, use `schema` and `table`.	No for source, Yes for sink

Example

```
{
    "name": "SQLServerDataset",
    "properties":
    {
        "type": "SqlServerTable",
        "linkedServiceName": {
            "referenceName": "<SQL Server linked service name>",
            "type": "LinkedServiceReference"
        },
        "schema": [ < physical schema, optional, retrievable during authoring > ],
        "typeProperties": {
            "schema": "<schema_name>",
            "table": "<table_name>"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for use to define activities, see the Pipelines article. This section provides a list of properties supported by the SQL Server source and sink.

SQL Server as a source

> **TIP**
> To load data from SQL Server efficiently by using data partitioning, learn more from Parallel copy from SQL database.

To copy data from SQL Server, set the source type in the copy activity to **SqlSource**. The following properties are supported in the copy activity source section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **SqlSource**.	Yes
sqlReaderQuery	Use the custom SQL query to read data. An example is `select * from MyTable`.	No
sqlReaderStoredProcedureName	This property is the name of the stored procedure that reads data from the source table. The last SQL statement must be a SELECT statement in the stored procedure.	No
storedProcedureParameters	These parameters are for the stored procedure. Allowed values are name or value pairs. The names and casing of parameters must match the names and casing of the stored procedure parameters.	No

PROPERTY	DESCRIPTION	REQUIRED
isolationLevel	Specifies the transaction locking behavior for the SQL source. The allowed values are: **ReadCommitted**, **ReadUncommitted**, **RepeatableRead**, **Serializable**, **Snapshot**. If not specified, the database's default isolation level is used. Refer to this doc for more details.	No
partitionOptions	Specifies the data partitioning options used to load data from SQL Server. Allowed values are: **None** (default), **PhysicalPartitionsOfTable**, and **DynamicRange**. When a partition option is enabled (that is, not `None`), the degree of parallelism to concurrently load data from SQL Server is controlled by the `parallelCopies` setting on the copy activity.	No
partitionSettings	Specify the group of the settings for data partitioning. Apply when the partition option isn't `None`.	No
Under `partitionSettings`***:***		
partitionColumnName	Specify the name of the source column **in integer or date/datetime type** (`int`, `smallint`, `bigint`, `date`, `smalldatetime`, `datetime`, `datetime2`, or `datetimeoffset`) that will be used by range partitioning for parallel copy. If not specified, the index or the primary key of the table is auto-detected and used as the partition column. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfDynamicRangePartitionCondition` in the WHERE clause. For an example, see the Parallel copy from SQL database section.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionUpperBound	The maximum value of the partition column for partition range splitting. This value is used to decide the partition stride, not for filtering the rows in table. All rows in the table or query result will be partitioned and copied. If not specified, copy activity auto detect the value. Apply when the partition option is `DynamicRange`. For an example, see the Parallel copy from SQL database section.	No
partitionLowerBound	The minimum value of the partition column for partition range splitting. This value is used to decide the partition stride, not for filtering the rows in table. All rows in the table or query result will be partitioned and copied. If not specified, copy activity auto detect the value. Apply when the partition option is `DynamicRange`. For an example, see the Parallel copy from SQL database section.	No

Note the following points:

- If **sqlReaderQuery** is specified for **SqlSource**, the copy activity runs this query against the SQL Server source to get the data. You also can specify a stored procedure by specifying **sqlReaderStoredProcedureName** and **storedProcedureParameters** if the stored procedure takes parameters.
- When using stored procedure in source to retrieve data, note if your stored procedure is designed as returning different schema when different parameter value is passed in, you may encounter failure or see unexpected result when importing schema from UI or when copying data to SQL database with auto table creation.

Example: Use SQL query

```
"activities":[
    {
        "name": "CopyFromSQLServer",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SQL Server input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SqlSource",
                "sqlReaderQuery": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Example: Use a stored procedure

```
"activities":[
    {
        "name": "CopyFromSQLServer",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<SQL Server input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SqlSource",
                "sqlReaderStoredProcedureName": "CopyTestSrcStoredProcedureWithParameters",
                "storedProcedureParameters": {
                    "stringData": { "value": "str3" },
                    "identifier": { "value": "$$Text.Format('{0:yyyy}', <datetime parameter>)", "type":
"Int"}
                }
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

The stored procedure definition

```
CREATE PROCEDURE CopyTestSrcStoredProcedureWithParameters
(
    @stringData varchar(20),
    @identifier int
)
AS
SET NOCOUNT ON;
BEGIN
    select *
    from dbo.UnitTestSrcTable
    where dbo.UnitTestSrcTable.stringData != stringData
    and dbo.UnitTestSrcTable.identifier != identifier
END
GO
```

SQL Server as a sink

> **TIP**
> Learn more about the supported write behaviors, configurations, and best practices from Best practice for loading data into SQL Server.

To copy data to SQL Server, set the sink type in the copy activity to **SqlSink**. The following properties are supported in the copy activity sink section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to **SqlSink**.	Yes
preCopyScript	This property specifies a SQL query for the copy activity to run before writing data into SQL Server. It's invoked only once per copy run. You can use this property to clean up the preloaded data.	No
tableOption	Specifies whether to automatically create the sink table if not exists based on the source schema. Auto table creation is not supported when sink specifies stored procedure. Allowed values are: `none` (default), `autoCreate`.	No
sqlWriterStoredProcedureName	The name of the stored procedure that defines how to apply source data into a target table. This stored procedure is *invoked per batch*. For operations that run only once and have nothing to do with source data, for example, delete or truncate, use the `preCopyScript` property. See example from Invoke a stored procedure from a SQL sink.	No
storedProcedureTableTypeParameterName	The parameter name of the table type specified in the stored procedure.	No

PROPERTY	DESCRIPTION	REQUIRED
sqlWriterTableType	The table type name to be used in the stored procedure. The copy activity makes the data being moved available in a temp table with this table type. Stored procedure code can then merge the data that's being copied with existing data.	No
storedProcedureParameters	Parameters for the stored procedure. Allowed values are name and value pairs. Names and casing of parameters must match the names and casing of the stored procedure parameters.	No
writeBatchSize	Number of rows to insert into the SQL table *per batch*. Allowed values are integers for the number of rows. By default, the service dynamically determines the appropriate batch size based on the row size.	No
writeBatchTimeout	This property specifies the wait time for the batch insert operation to complete before it times out. Allowed values are for the timespan. An example is "00:30:00" for 30 minutes. If no value is specified, the timeout defaults to "02:00:00".	No
maxConcurrentConnections	The upper limit of concurrent connecti ons established to the data store durin g the activity run. Specify a value only when you want to limit concurrent con nections.	No
WriteBehavior	Specify the write behavior for copy activity to load data into SQL Server Database. The allowed value is **Insert** and **Upsert**. By default, the service uses insert to load data.	No
upsertSettings	Specify the group of the settings for write behavior. Apply when the WriteBehavior option is `Upert`.	No
Under `upsertSettings`***:***		
useTempDB	Specify whether to use the a global temporary table or physical table as the interim table for upsert. By default, the service uses global temporary table as the interim table. value is `true`.	No

PROPERTY	DESCRIPTION	REQUIRED
interimSchemaName	Specify the interim schema for creating interim table if physical table is used. Note: user need to have the permission for creating and deleting table. By default, interim table will share the same schema as sink table. Apply when the useTempDB option is `False`.	No
keys	Specify the column names for unique row identification. Either a single key or a series of keys can be used. If not specified, the primary key is used.	No

Example 1: Append data

```
"activities":[
    {
        "name": "CopyToSQLServer",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<SQL Server output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "SqlSink",
                "tableOption": "autoCreate",
                "writeBatchSize": 100000
            }
        }
    }
]
```

Example 2: Invoke a stored procedure during copy

Learn more details from Invoke a stored procedure from a SQL sink.

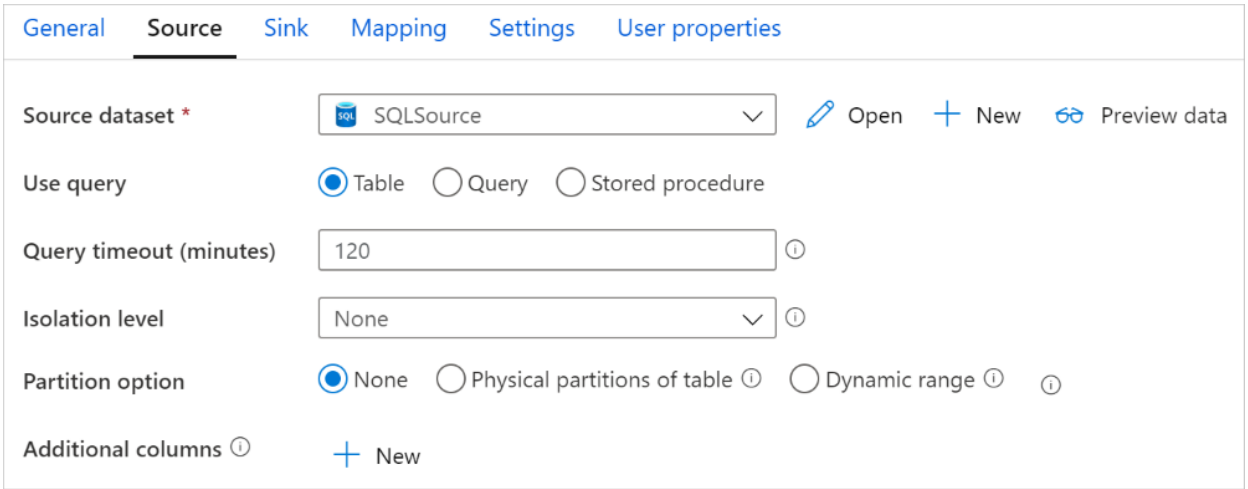
```
"activities":[
    {
        "name": "CopyToSQLServer",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<SQL Server output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "SqlSink",
                "sqlWriterStoredProcedureName": "CopyTestStoredProcedureWithParameters",
                "storedProcedureTableTypeParameterName": "MyTable",
                "sqlWriterTableType": "MyTableType",
                "storedProcedureParameters": {
                    "identifier": { "value": "1", "type": "Int" },
                    "stringData": { "value": "str1" }
                }
            }
        }
    }
]
```

Example 3: Upsert data

```
"activities":[
    {
        "name": "CopyToSQLServer",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<SQL Server output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "SqlSink",
                "tableOption": "autoCreate",
                "writeBehavior": "upsert",
                "upsertSettings": {
                    "useTempDB": true,
                    "keys": [
                        "<column name>"
                    ]
                },
            }
        }
    }
]
```

Parallel copy from SQL database

The SQL Server connector in copy activity provides built-in data partitioning to copy data in parallel. You can find data partitioning options on the **Source** tab of the copy activity.



When you enable partitioned copy, copy activity runs parallel queries against your SQL Server source to load data by partitions. The parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your SQL Server.

You are suggested to enable parallel copy with data partitioning especially when you load large amount of data from your SQL Server. The following are suggested configurations for different scenarios. When copying data

into file-based data store, it's recommended to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

SCENARIO	SUGGESTED SETTINGS
Full load from large table, with physical partitions.	**Partition option**: Physical partitions of table. During execution, the service automatically detects the physical partitions, and copies data by partitions. To check if your table has physical partition or not, you can refer to this query.
Full load from large table, without physical partitions, while with an integer or datetime column for data partitioning.	**Partition options**: Dynamic range partition. **Partition column** (optional): Specify the column used to partition data. If not specified, the primary key column is used. **Partition upper bound** and **partition lower bound** (optional): Specify if you want to determine the partition stride. This is not for filtering the rows in table, all rows in the table will be partitioned and copied. If not specified, copy activity auto detects the values and it can take long time depending on MIN and MAX values. It is recommended to provide upper bound and lower bound. For example, if your partition column "ID" has values range from 1 to 100, and you set the lower bound as 20 and the upper bound as 80, with parallel copy as 4, the service retrieves data by 4 partitions - IDs in range <=20, [21, 50], [51, 80], and >=81, respectively.

SCENARIO	SUGGESTED SETTINGS

Load a large amount of data by using a custom query, without physical partitions, while with an integer or date/datetime column for data partitioning.	**Partition options**: Dynamic range partition.
	Query:
	`SELECT * FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	.
	Partition column: Specify the column used to partition data.
	Partition upper bound and **partition lower bound** (optional): Specify if you want to determine the partition stride. This is not for filtering the rows in table, all rows in the query result will be partitioned and copied. If not specified, copy activity auto detect the value.
	
	During execution, the service replaces `?AdfRangePartitionColumnName` with the actual column name and value ranges for each partition, and sends to SQL Server.
	For example, if your partition column "ID" has values range from 1 to 100, and you set the lower bound as 20 and the upper bound as 80, with parallel copy as 4, the service retrieves data by 4 partitions- IDs in range <=20, [21, 50], [51, 80], and >=81, respectively.
	
	Here are more sample queries for different scenarios:
	1. Query the whole table:
	`SELECT * FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition`
	2. Query from a table with column selection and additional where-clause filters:
	`SELECT <column_list> FROM <TableName> WHERE ? AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	3. Query with subqueries:
	`SELECT <column_list> FROM (<your_sub_query>) AS T WHERE ?AdfDynamicRangePartitionCondition AND <your_additional_where_clause>`
	4. Query with partition in subquery:
	`SELECT <column_list> FROM (SELECT <your_sub_query_column_list> FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition) AS T`

Best practices to load data with partition option:

1. Choose distinctive column as partition column (like primary key or unique key) to avoid data skew.
2. If the table has built-in partition, use partition option "Physical partitions of table" to get better performance.
3. If you use Azure Integration Runtime to copy data, you can set larger "Data Integration Units (DIU)" (>4) to utilize more computing resource. Check the applicable scenarios there.
4. "Degree of copy parallelism" control the partition numbers, setting this number too large sometime hurts the performance, recommend setting this number as (DIU or number of Self-hosted IR nodes) * (2 to 4).

Example: full load from large table with physical partitions

```
"source": {
    "type": "SqlSource",
    "partitionOption": "PhysicalPartitionsOfTable"
}
```

Example: query with dynamic range partition

```
"source": {
    "type": "SqlSource",
    "query":"SELECT * FROM <TableName> WHERE ?AdfDynamicRangePartitionCondition AND
<your_additional_where_clause>",
    "partitionOption": "DynamicRange",
    "partitionSettings": {
        "partitionColumnName": "<partition_column_name>",
        "partitionUpperBound": "<upper_value_of_partition_column (optional) to decide the partition stride,
not as data filter>",
        "partitionLowerBound": "<lower_value_of_partition_column (optional) to decide the partition stride,
not as data filter>"
    }
}
```

Sample query to check physical partition

```
SELECT DISTINCT s.name AS SchemaName, t.name AS TableName, pf.name AS PartitionFunctionName, c.name AS
ColumnName, iif(pf.name is null, 'no', 'yes') AS HasPartition
FROM sys.tables AS t
LEFT JOIN sys.objects AS o ON t.object_id = o.object_id
LEFT JOIN sys.schemas AS s ON o.schema_id = s.schema_id
LEFT JOIN sys.indexes AS i ON t.object_id = i.object_id
LEFT JOIN sys.index_columns AS ic ON ic.partition_ordinal > 0 AND ic.index_id = i.index_id AND ic.object_id
= t.object_id
LEFT JOIN sys.columns AS c ON c.object_id = ic.object_id AND c.column_id = ic.column_id
LEFT JOIN sys.partition_schemes ps ON i.data_space_id = ps.data_space_id
LEFT JOIN sys.partition_functions pf ON pf.function_id = ps.function_id
WHERE s.name='[your schema]' AND t.name = '[your table name]'
```

If the table has physical partition, you would see "HasPartition" as "yes" like the following.

SchemaName	TableName	PartitionFunctionName	ColumnName	HasPartition
dbo	PerfWithPartition	RangePF1	Id	yes

Best practice for loading data into SQL Server

When you copy data into SQL Server, you might require different write behavior:

- Append: My source data has only new records.
- Upsert: My source data has both inserts and updates.
- Overwrite: I want to reload the entire dimension table each time.
- Write with custom logic: I need extra processing before the final insertion into the destination table.

See the respective sections for how to configure and best practices.

Append data

Appending data is the default behavior of this SQL Server sink connector. the service does a bulk insert to write to your table efficiently. You can configure the source and sink accordingly in the copy activity.

Upsert data

Copy activity now supports natively loading data into a database temporary table and then update the data in sink table if key exists and otherwise insert new data. To learn more about upsert settings in copy activities, see SQL Server as a sink.

Overwrite the entire table

You can configure the **preCopyScript** property in a copy activity sink. In this case, for each copy activity that runs, the service runs the script first. Then it runs the copy to insert the data. For example, to overwrite the entire table with the latest data, specify a script to first delete all the records before you bulk load the new data from

the source.

Write data with custom logic

The steps to write data with custom logic are similar to those described in the Upsert data section. When you need to apply extra processing before the final insertion of source data into the destination table, you can load to a staging table then invoke stored procedure activity, or invoke a stored procedure in copy activity sink to apply data.

Invoke a stored procedure from a SQL sink

When you copy data into SQL Server database, you also can configure and invoke a user-specified stored procedure with additional parameters on each batch of the source table. The stored procedure feature takes advantage of table-valued parameters. Note that the service automatically wraps the stored procedure in its own transaction, so any transaction created inside the stored procedure will become a nested transaction, and could have implications for exception handling.

You can use a stored procedure when built-in copy mechanisms don't serve the purpose. An example is when you want to apply extra processing before the final insertion of source data into the destination table. Some extra processing examples are when you want to merge columns, look up additional values, and insert into more than one table.

The following sample shows how to use a stored procedure to do an upsert into a table in the SQL Server database. Assume that the input data and the sink **Marketing** table each have three columns: **ProfileID**, **State**, and **Category**. Do the upsert based on the **ProfileID** column, and only apply it for a specific category called "ProductA".

1. In your database, define the table type with the same name as **sqlWriterTableType**. The schema of the table type is the same as the schema returned by your input data.

   ```
   CREATE TYPE [dbo].[MarketingType] AS TABLE(
       [ProfileID] [varchar](256) NOT NULL,
       [State] [varchar](256) NOT NULL,
       [Category] [varchar](256) NOT NULL
   )
   ```

2. In your database, define the stored procedure with the same name as **sqlWriterStoredProcedureName**. It handles input data from your specified source and merges into the output table. The parameter name of the table type in the stored procedure is the same as **tableName** defined in the dataset.

   ```
   CREATE PROCEDURE spOverwriteMarketing @Marketing [dbo].[MarketingType] READONLY, @category
   varchar(256)
   AS
   BEGIN
   MERGE [dbo].[Marketing] AS target
   USING @Marketing AS source
   ON (target.ProfileID = source.ProfileID and target.Category = @category)
   WHEN MATCHED THEN
       UPDATE SET State = source.State
   WHEN NOT MATCHED THEN
       INSERT (ProfileID, State, Category)
       VALUES (source.ProfileID, source.State, source.Category);
   END
   ```

3. Define the **SQL sink** section in the copy activity as follows:

```
"sink": {
    "type": "SqlSink",
    "sqlWriterStoredProcedureName": "spOverwriteMarketing",
    "storedProcedureTableTypeParameterName": "Marketing",
    "sqlWriterTableType": "MarketingType",
    "storedProcedureParameters": {
        "category": {
            "value": "ProductA"
        }
    }
}
```

Mapping data flow properties

When transforming data in mapping data flow, you can read and write to tables from SQL Server Database. For more information, see the source transformation and sink transformation in mapping data flows.

> **NOTE**
>
> To access on premise SQL Server, you need to use Azure Data Factory or Synapse workspace Managed Virtual Network using a private endpoint. Refer to this tutorial for detailed steps.

Source transformation

The below table lists the properties supported by SQL Server source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Table	If you select Table as input, data flow fetches all the data from the table specified in the dataset.	No	-	-

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Query	If you select Query as input, specify a SQL query to fetch data from source, which overrides any table you specify in dataset. Using queries is a great way to reduce rows for testing or lookups. **Order By** clause is not supported, but you can set a full SELECT FROM statement. You can also use user-defined table functions. **select * from udfGetData()** is a UDF in SQL that returns a table that you can use in data flow. Query example: `Select * from MyTable where customerId > 1000 and customerId < 2000`	No	String	query
Batch size	Specify a batch size to chunk large data into reads.	No	Integer	batchSize
Isolation Level	Choose one of the following isolation levels: - Read Committed - Read Uncommitted (default) - Repeatable Read - Serializable - None (ignore isolation level)	No	READ_COMMITTED READ_UNCOMMITTED REPEATABLE_READ SERIALIZABLE NONE	isolationLevel

TIP

The common table expression (CTE) in SQL is not supported in the mapping data flow **Query** mode, because the prerequisite of using this mode is that queries can be used in the SQL query FROM clause but CTEs cannot do this. To use CTEs, you need to create a stored procedure using the following query:

```
CREATE PROC CTESP @query nvarchar(max)
AS
BEGIN
EXECUTE sp_executesql @query;
END
```

Then use the **Stored procedure** mode in the source transformation of the mapping data flow and set the `@query` like example `with CTE as (select 'test' as a) select * from CTE`. Then you can use CTEs as expected.

SQL Server source script example

When you use SQL Server as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
    validateSchema: false,
    isolationLevel: 'READ_UNCOMMITTED',
    query: 'select * from MYTABLE',
    format: 'query') ~> SQLSource
```

Sink transformation

The below table lists the properties supported by SQL Server sink. You can edit these properties in the **Sink options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Update method	Specify what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an Alter row transformation is required to tag rows for those actions.	Yes	`true` or `false`	deletable insertable updateable upsertable
Key columns	For updates, upserts and deletes, key column(s) must be set to determine which row to alter. The column name that you pick as the key will be used as part of the subsequent update, upsert, delete. Therefore, you must pick a column that exists in the Sink mapping.	No	Array	keys

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Skip writing key columns	If you wish to not write the value to the key column, select "Skip writing key columns".	No	`true` or `false`	skipKeyWrites
Table action	Determines whether to recreate or remove all rows from the destination table prior to writing. - **None**: No action will be done to the table. - **Recreate**: The table will get dropped and recreated. Required if creating a new table dynamically. - **Truncate**: All rows from the target table will get removed.	No	`true` or `false`	recreate truncate
Batch size	Specify how many rows are being written in each batch. Larger batch sizes improve compression and memory optimization, but risk out of memory exceptions when caching data.	No	Integer	batchSize
Pre and Post SQL scripts	Specify multi-line SQL scripts that will execute before (pre-processing) and after (post-processing) data is written to your Sink database.	No	String	preSQLs postSQLs

TIP

1. It's recommended to break single batch scripts with multiple commands into multiple batches.
2. Only Data Definition Language (DDL) and Data Manipulation Language (DML) statements that return a simple update count can be run as part of a batch. Learn more from Performing batch operations

SQL Server sink script example

When you use SQL Server as sink type, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    deletable:false,
    insertable:true,
    updateable:true,
    upsertable:true,
    keys:['keyColumn'],
    format: 'table',
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> SQLSink
```

Data type mapping for SQL Server

When you copy data from and to SQL Server, the following mappings are used from SQL Server data types to Azure Data Factory interim data types. Synapse pipelines, which implement Data Factory, use the same mappings. To learn how the copy activity maps the source schema and data type to the sink, see Schema and data type mappings.

SQL SERVER DATA TYPE	DATA FACTORY INTERIM DATA TYPE
bigint	Int64
binary	Byte[]
bit	Boolean
char	String, Char[]
date	DateTime
Datetime	DateTime
datetime2	DateTime
Datetimeoffset	DateTimeOffset
Decimal	Decimal
FILESTREAM attribute (varbinary(max))	Byte[]
Float	Double
image	Byte[]
int	Int32
money	Decimal
nchar	String, Char[]
ntext	String, Char[]
numeric	Decimal

SQL SERVER DATA TYPE	DATA FACTORY INTERIM DATA TYPE
nvarchar	String, Char[]
real	Single
rowversion	Byte[]
smalldatetime	DateTime
smallint	Int16
smallmoney	Decimal
sql_variant	Object
text	String, Char[]
time	TimeSpan
timestamp	Byte[]
tinyint	Int16
uniqueidentifier	Guid
varbinary	Byte[]
varchar	String, Char[]
xml	String

> **NOTE**
>
> For data types that map to the Decimal interim type, currently Copy activity supports precision up to 28. If you have data that requires precision larger than 28, consider converting to a string in a SQL query.
>
> When copying data from SQL Server using Azure Data Factory, the bit data type is mapped to the Boolean interim data type. If you have data that need to be kept as the bit data type, use queries with T-SQL CAST or CONVERT.

Lookup activity properties

To learn details about the properties, check Lookup activity.

GetMetadata activity properties

To learn details about the properties, check GetMetadata activity

Using Always Encrypted

When you copy data from/to SQL Server with Always Encrypted, follow below steps:

1. Store the Column Master Key (CMK) in an Azure Key Vault. Learn more on how to configure Always

Encrypted by using Azure Key Vault

2. Make sure to grant access to the key vault where the Column Master Key (CMK) is stored. Refer to this article for required permissions.

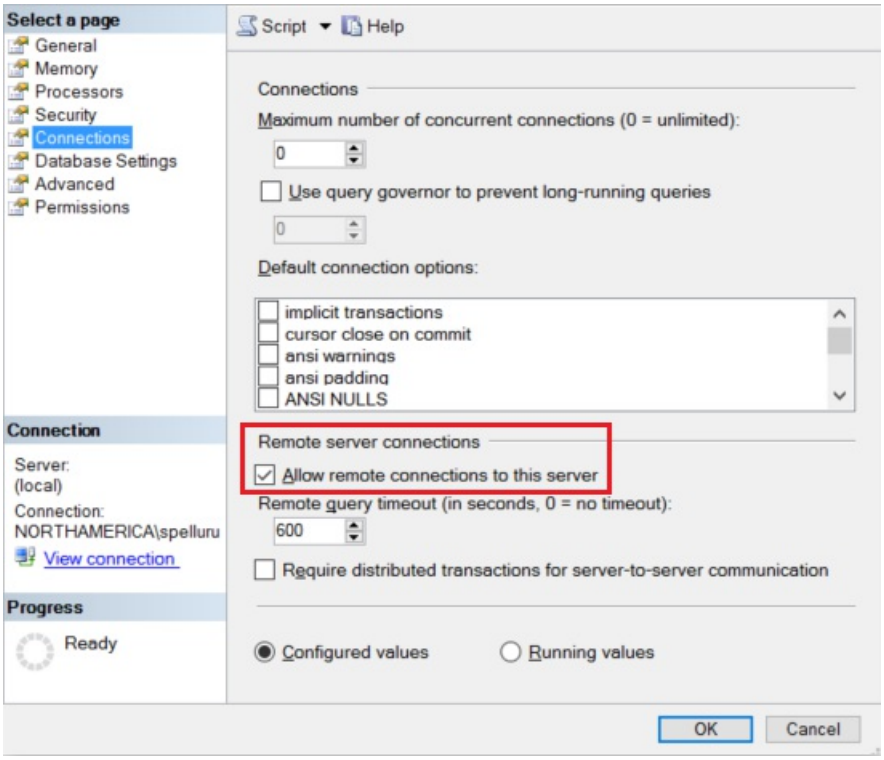
3. Create linked service to connect to your SQL database and enable 'Always Encrypted' function by using either managed identity or service principal.

> **NOTE**
>
> SQL Server Always Encrypted supports below scenarios:
>
> 1. Either source or sink data stores is using managed identity or service principal as key provider authentication type.
> 2. Both source and sink data stores are using managed identity as key provider authentication type.
> 3. Both source and sink data stores are using the same service principal as key provider authentication type.

> **NOTE**
>
> Currently, SQL Server **Always Encrypted** is only supported for source transformation in mapping data flows.

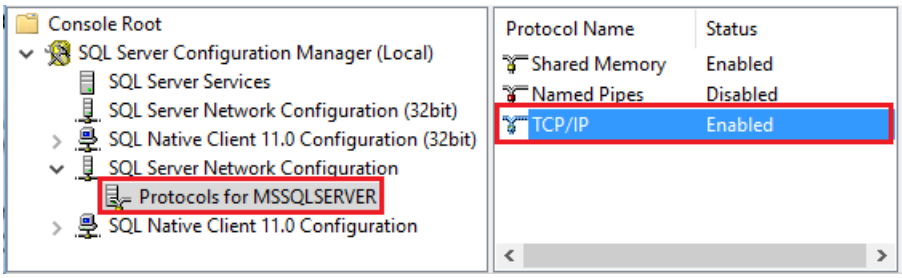
Troubleshoot connection issues

1. Configure your SQL Server instance to accept remote connections. Start **SQL Server Management Studio**, right-click **server**, and select **Properties**. Select **Connections** from the list, and select the **Allow remote connections to this server** check box.



For detailed steps, see Configure the remote access server configuration option.

2. Start **SQL Server Configuration Manager**. Expand **SQL Server Network Configuration** for the instance you want, and select **Protocols for MSSQLSERVER**. Protocols appear in the right pane. Enable TCP/IP by right-clicking **TCP/IP** and selecting **Enable**.



For more information and alternate ways of enabling TCP/IP protocol, see Enable or disable a server network protocol.

3. In the same window, double-click **TCP/IP** to launch the **TCP/IP Properties** window.
4. Switch to the **IP Addresses** tab. Scroll down to see the **IPAll** section. Write down the **TCP Port**. The default is **1433**.
5. Create a **rule for the Windows Firewall** on the machine to allow incoming traffic through this port.
6. **Verify connection**: To connect to SQL Server by using a fully qualified name, use SQL Server Management Studio from a different machine. An example is `"<machine>.<domain>.corp.<company>.com,1433"`.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from Square using Azure Data Factory or Synapse Analytics (Preview)

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Square. It builds on the copy activity overview article that presents a general overview of copy activity.

IMPORTANT

This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

NOTE

Currently, this connector doesn't support sandbox accounts.

Supported capabilities

This Square connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

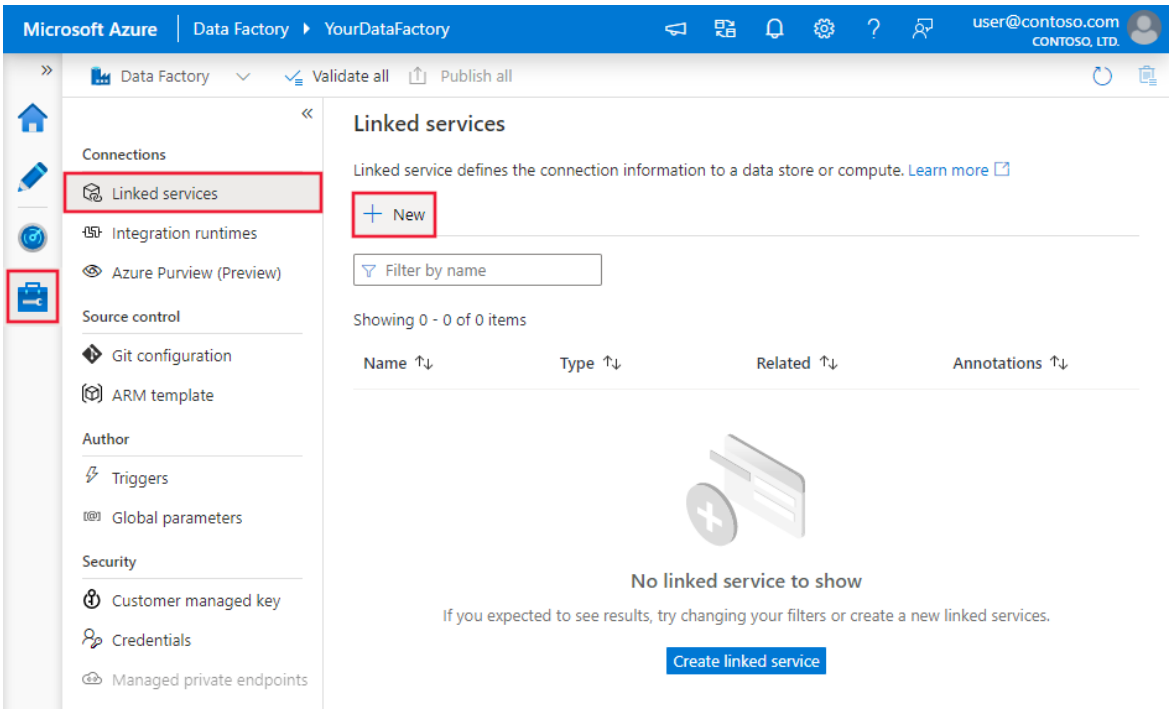
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Square using UI

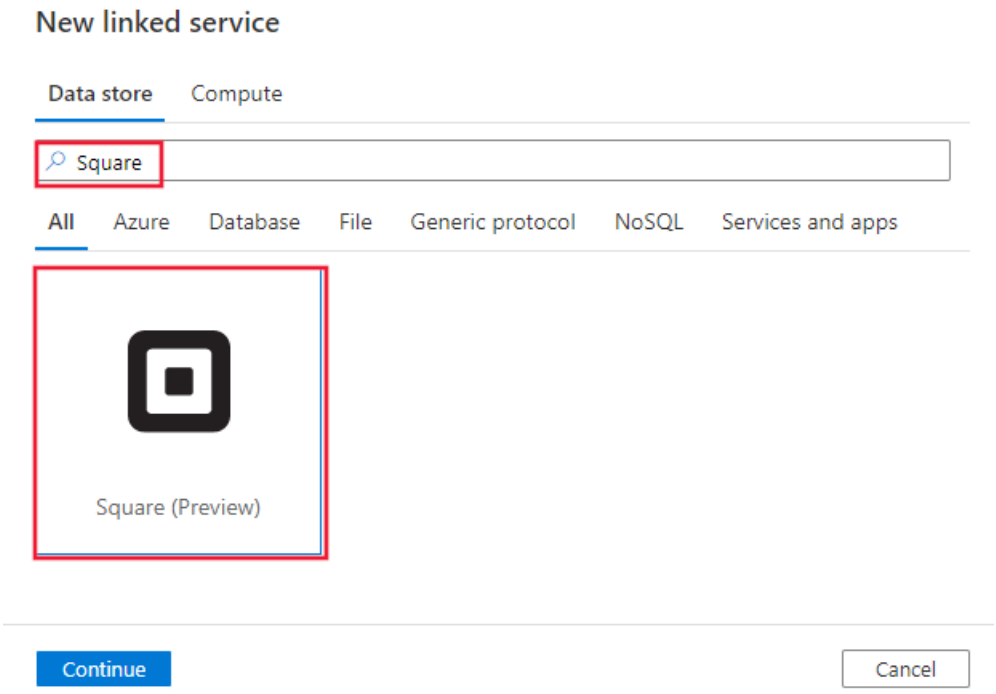
Use the following steps to create a linked service to Square in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

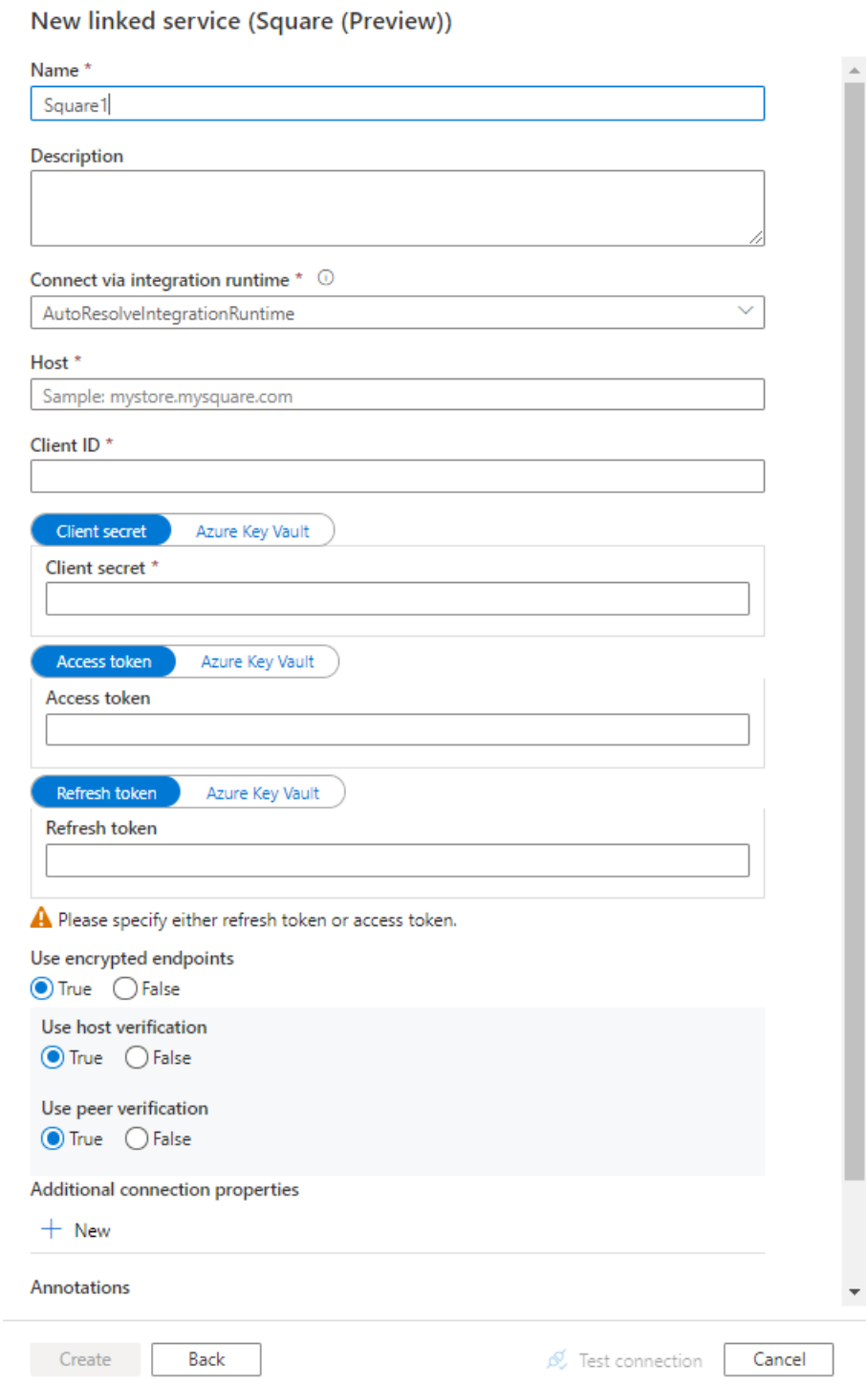
   - Azure Data Factory
   - Azure Synapse



2. Search for Square and select the Square connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Square connector.

Linked service properties

The following properties are supported for Square linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Square**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionProperties	A group of properties that defines how to connect to Square.	Yes
Under `connectionProperties`*:*		
host	The URL of the Square instance. (i.e. mystore.mysquare.com)	Yes
clientId	The client ID associated with your Square application.	Yes
clientSecret	The client secret associated with your Square application. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
accessToken	The access token obtained from Square. Grants limited access to a Square account by asking an authenticated user for explicit permissions. OAuth access tokens expires 30 days after issued, but refresh tokens do not expire. Access tokens can be refreshed by refresh token. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
refreshToken	The refresh token obtained from Square. Used to obtain new access tokens when the current one expires. Mark this field as a SecureString to store it securelyFactory, or reference a secret stored in Azure Key Vault.	No
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Square support two types of access token: **personal** and **OAuth**.

- Personal access tokens are used to get unlimited Connect API access to resources in your own Square account.
- OAuth access tokens are used to get authenticated and scoped Connect API access to any Square account.

Use them when your app accesses resources in other Square accounts on behalf of account owners. OAuth access tokens can also be used to access resources in your own Square account.

> **IMPORTANT**
> To perform **Test connection** in the linked service, `MERCHANT_PROFILE_READ` is required to get a scoped OAuth access token. For permissions to access other tables, see Square OAuth Permissions Reference.

Authentication via personal access token only needs `accessToken`, while authentication via OAuth requires `accessToken` and `refreshToken`. Learn how to retrieve access token from here.

Example:

```
{
    "name": "SquareLinkedService",
    "properties": {
        "type": "Square",
        "typeProperties": {
            "connectionProperties":{
                "host":"<e.g. mystore.mysquare.com>",
                "clientId":"<client ID>",
                "clientSecrect":{
                    "type": "SecureString",
                    "value": "<clientSecret>"
                },
                "accessToken":{
                    "type": "SecureString",
                    "value": "<access token>"
                },
                "refreshToken":{
                    "type": "SecureString",
                    "value": "<refresh token>"
                },
                "useEncryptedEndpoints":true,
                "useHostVerification":true,
                "usePeerVerification":true
            }
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Square dataset.

To copy data from Square, set the type property of the dataset to **SquareObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **SquareObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "SquareDataset",
    "properties": {
        "type": "SquareObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Square linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Square source.

Square as source

To copy data from Square, set the source type in the copy activity to **SquareSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **SquareSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM Business"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromSquare",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Square input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SquareSource",
                "query": "SELECT * FROM Business"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Sybase using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from a Sybase database. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Sybase connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	②
Lookup activity	②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

Specifically, this Sybase connector supports:

- SAP Sybase SQL Anywhere (ASA) **version 16 and above**.
- Copying data using **Basic** or **Windows** authentication.

Sybase IQ and ASE are not supported. You can use generic ODBC connector with Sybase driver instead.

Prerequisites

To use this Sybase connector, you need to:

- Set up a Self-hosted Integration Runtime. See Self-hosted Integration Runtime article for details.
- Install the data provider for Sybase iAnywhere.Data.SQLAnywhere 16 or above on the Integration Runtime machine.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API

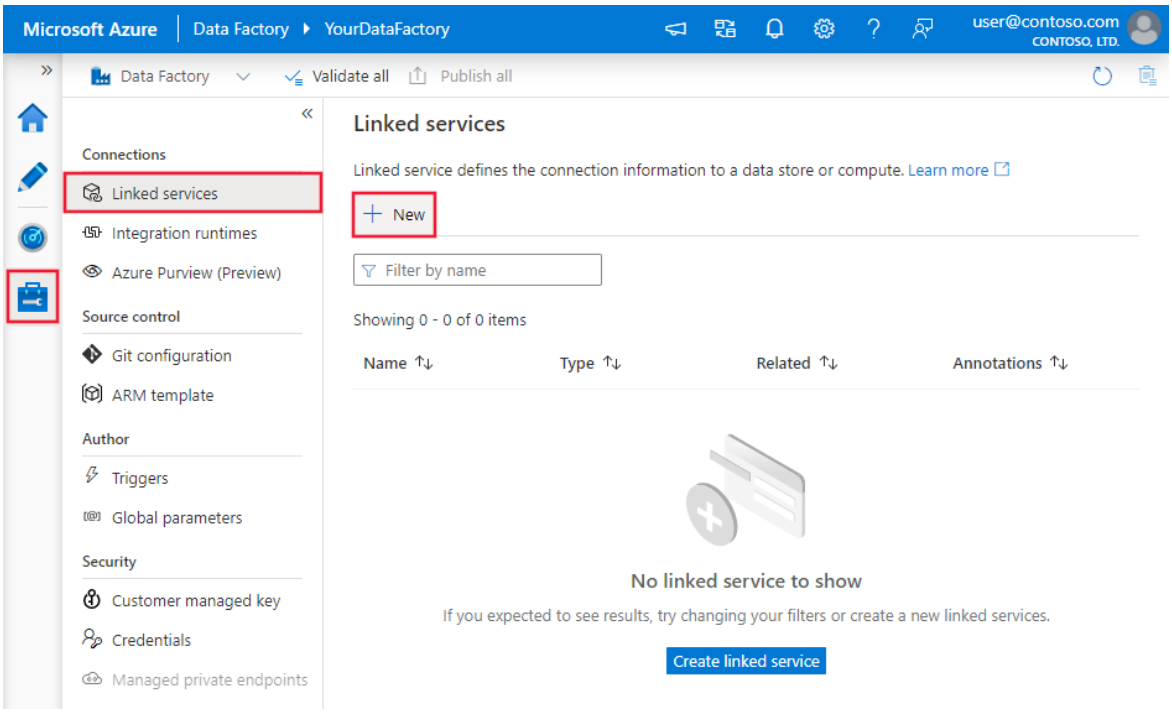
- The Azure Resource Manager template

Create a linked service to Sybase using UI

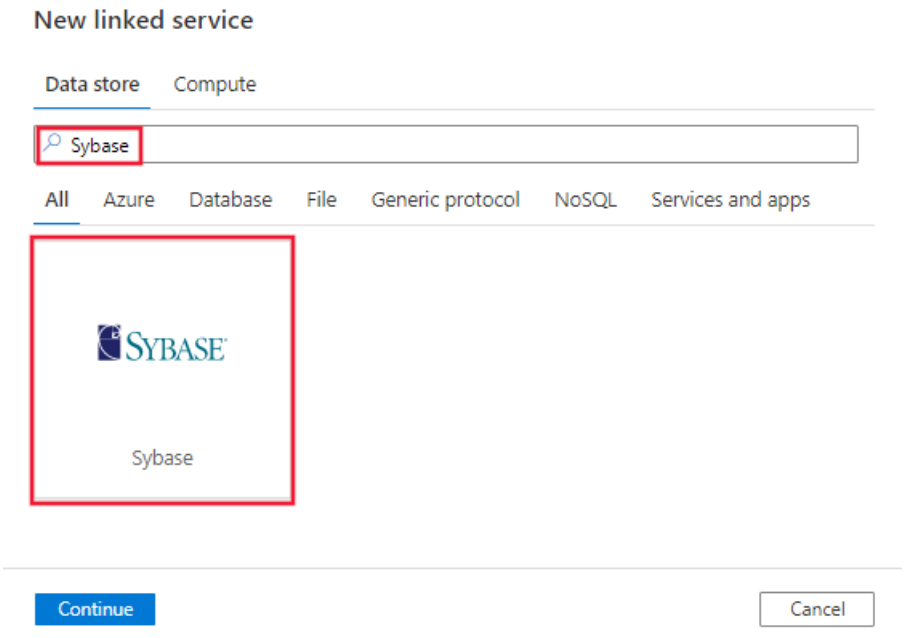
Use the following steps to create a linked service to Sybase in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

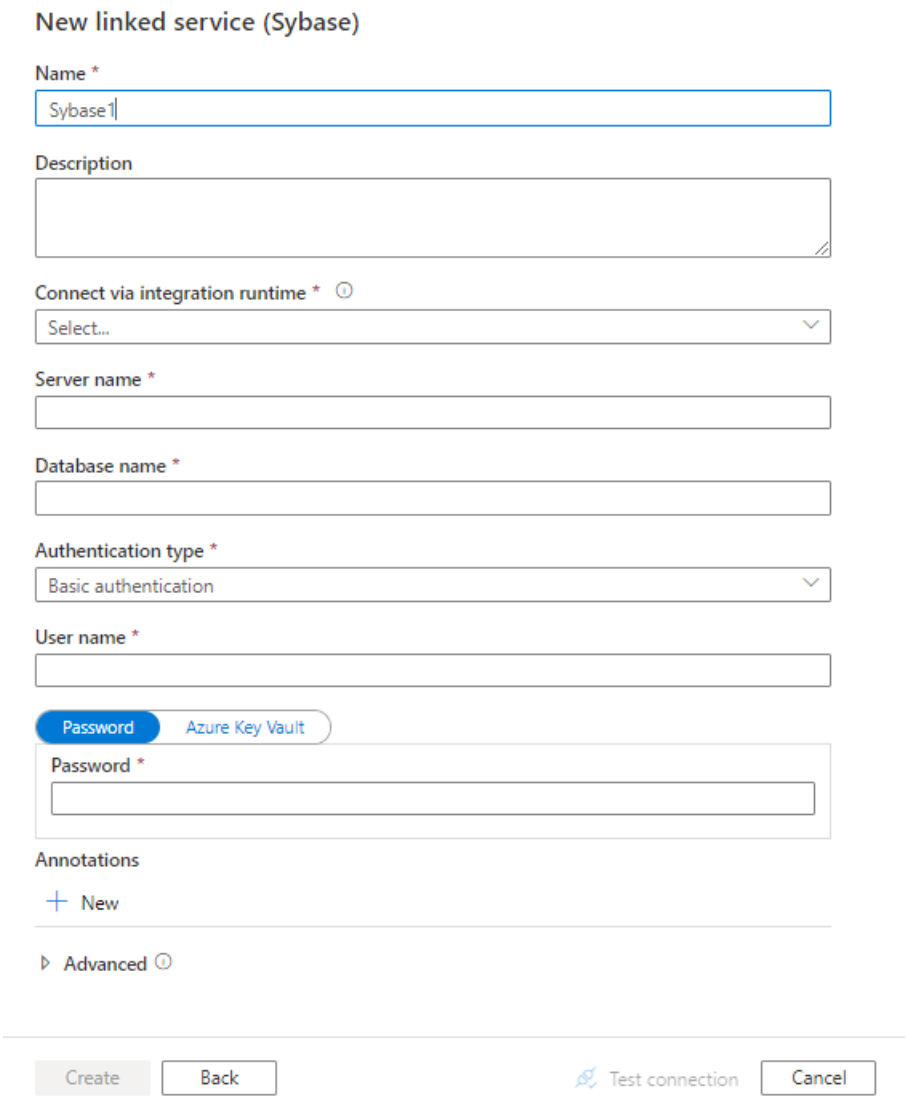
   - Azure Data Factory
   - Azure Synapse



2. Search for Sybase and select the Sybase connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Sybase connector.

Linked service properties

The following properties are supported for Sybase linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Sybase**	Yes
server	Name of the Sybase server.	Yes
database	Name of the Sybase database.	Yes
authenticationType	Type of authentication used to connect to the Sybase database. Allowed values are: **Basic**, and **Windows**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
username	Specify user name to connect to the Sybase database.	Yes
password	Specify password for the user account you specified for the username. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. A Self-hosted Integration Runtime is required as mentioned in Prerequisites.	Yes

Example:

```
{
    "name": "SybaseLinkedService",
    "properties": {
        "type": "Sybase",
        "typeProperties": {
            "server": "<server>",
            "database": "<database>",
            "authenticationType": "Basic",
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Sybase dataset.

To copy data from Sybase, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **SybaseTable**	Yes
tableName	Name of the table in the Sybase database.	No (if "query" in activity source is specified)

Example

```
{
    "name": "SybaseDataset",
    "properties": {
        "type": "SybaseTable",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Sybase linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

If you were using `RelationalTable` typed dataset, it is still supported as-is, while you are suggested to use the new one going forward.

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Sybase source.

Sybase as source

To copy data from Sybase, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **SybaseSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromSybase",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Sybase input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "SybaseSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

If you were using `RelationalSource` typed source, it is still supported as-is, while you are suggested to use the new one going forward.

Data type mapping for Sybase

When copying data from Sybase, the following mappings are used from Sybase data types to interim data types used internally within the service. See Schema and data type mappings to learn about how copy activity maps the source schema and data type to the sink.

Sybase supports T-SQL types. For a mapping table from SQL types to interim service data types, see Azure SQL Database Connector - data type mapping section.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Transform data in TeamDesk (Preview) using Azure Data Factory or Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Data Flow to transform data in TeamDesk (Preview). To learn more, read the introductory article for Azure Data Factory or Azure Synapse Analytics.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This TeamDesk connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Mapping data flow (source/-)	①

① Azure integration runtime ② Self-hosted integration runtime

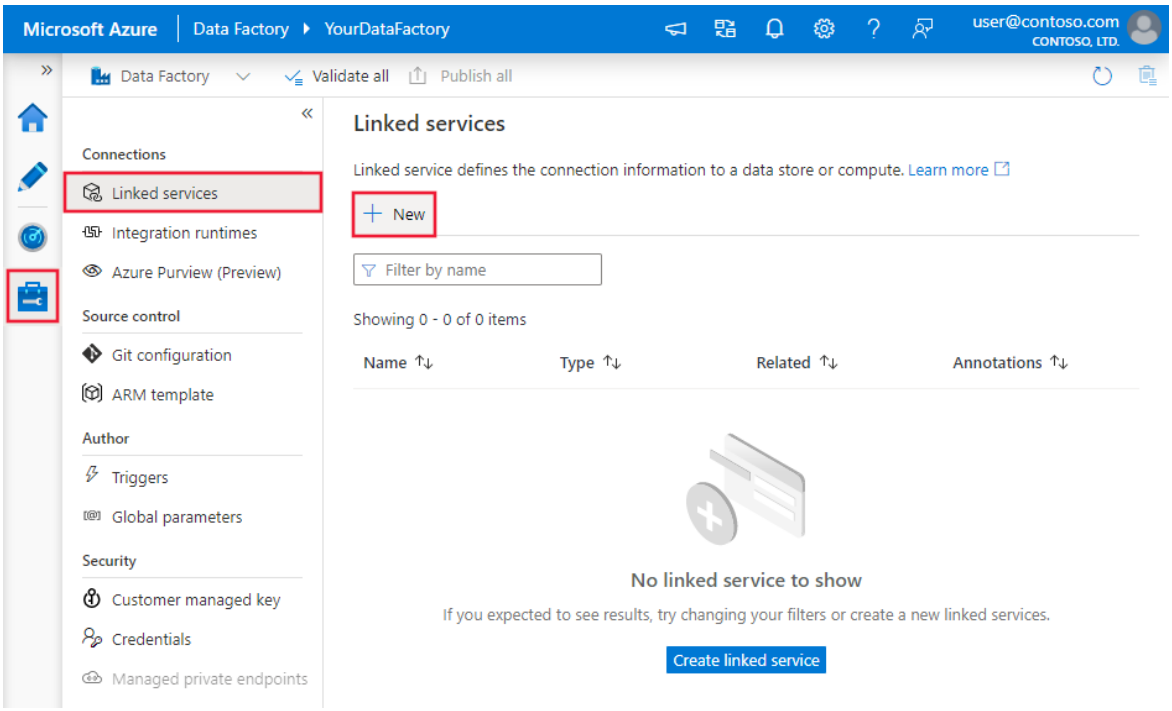
For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Create a TeamDesk linked service using UI

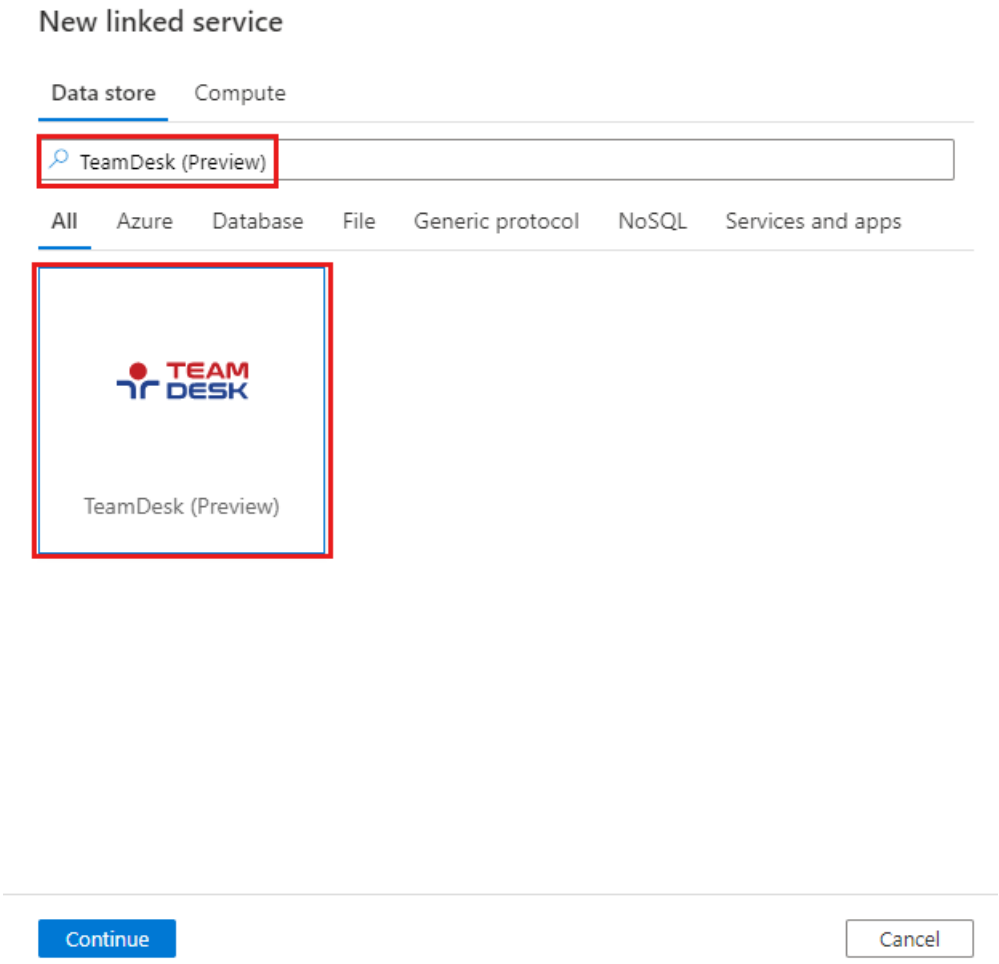
Use the following steps to create a TeamDesk linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then select New:

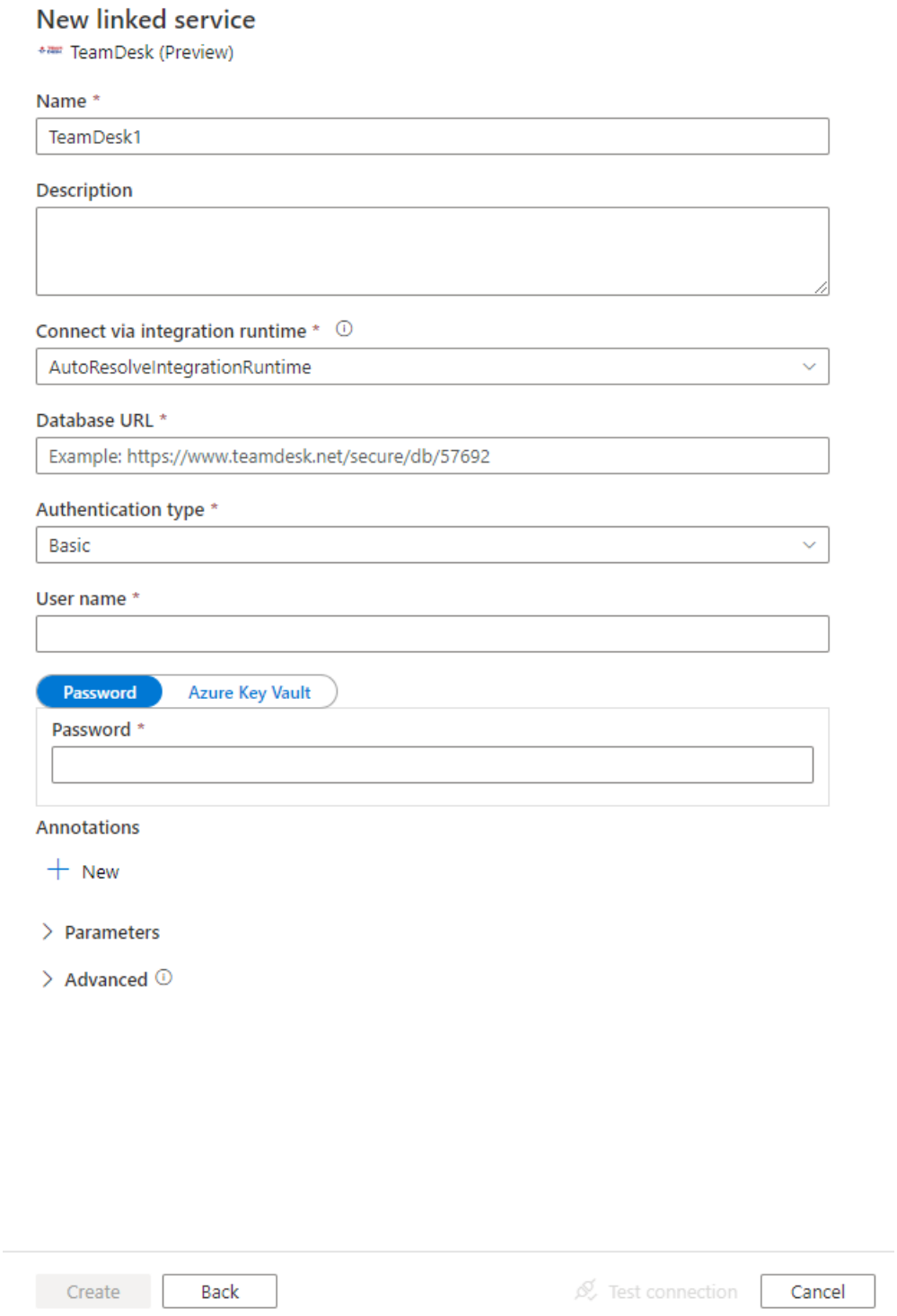
   - Azure Data Factory
   - Azure Synapse



2. Search for TeamDesk (Preview) and select the TeamDesk (Preview) connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide information about properties that are used to define Data Factory and Synapse pipeline entities specific to TeamDesk.

Linked service properties

The following properties are supported for the TeamDesk linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **TeamDesk**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
url	The URL of your TeamDesk database. An example is `https://www.teamdesk.net/secure/db/xxxxx`.	Yes
authenticationType	Type of authentication used to connect to the TeamDesk service. Allowed values are **basic** and **token**. Refer to corresponding sections below on more properties and examples respectively.	Yes

Basic authentication

Set the **authenticationType** property to **basic**. In addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
userName	The user name used to log in to TeamDesk.	Yes
password	Specify a password for the user account you specified for the user name. Mark this field as **SecureString** to store it securely. Or, you can reference a secret stored in Azure Key Vault.	Yes

Example:

```
{
    "name": "TeamDeskLinkedService",
    "properties": {
        "type": "TeamDesk",
        "typeProperties": {
            "url": "https://www.teamdesk.net/secure/db/xxxxx",
            "authenticationType": "basic",
            "userName": "<user name>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        }
    }
}
```

Token authentication

Set the **authenticationType** property to **token**. In addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
apiToken	Specify an API token for the TeamDesk. Mark this field as **SecureString** to store it securely. Or, you can reference a secret stored in Azure Key Vault.	Yes

Example:

```
{
    "name": "TeamDeskLinkedService",
    "properties": {
        "type": "TeamDesk",
        "typeProperties": {
            "url": "https://www.teamdesk.net/secure/db/xxxxx",
            "authenticationType": "token",
            "apiToken": {
                "type": "SecureString",
                "value": "<API token>"
            }
        }
    }
}
```

Mapping data flow properties

When transforming data in mapping data flow, you can read tables from TeamDesk. For more information, see the source transformation in mapping data flows. You can only use an inline dataset as source type.

Source transformation

The below table lists the properties supported by TeamDesk source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Table	Data flow will fetch all the data from the table specified in the source options.	Yes when use inline mode	-	table
View	Data flow will fetch the specified view in the table specified in the source options.	No	-	view

TeamDesk source script examples

```
source(allowSchemaDrift: true,
 validateSchema: false,
 store: 'teamdesk',
 format: 'rest',
 table: 'Table',
 view: 'View') ~> TeamDesksource
```

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from Teradata Vantage using Azure Data Factory and Synapse Analytics

8/19/2022 • 11 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the copy activity in Azure Data Factory and Synapse Analytics pipelines to copy data from Teradata Vantage. It builds on the copy activity overview.

Supported capabilities

This Teradata connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

Specifically, this Teradata connector supports:

- Teradata **version 14.10, 15.0, 15.10, 16.0, 16.10, and 16.20**.
- Copying data by using **Basic**, **Windows**, or **LDAP** authentication.
- Parallel copying from a Teradata source. See the Parallel copy from Teradata section for details.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

If you use Self-hosted Integration Runtime, note it provides a built-in Teradata driver starting from version 3.18. You don't need to manually install any driver. The driver requires "Visual C++ Redistributable 2012 Update 4" on the self-hosted integration runtime machine. If you don't yet have it installed, download it from here.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

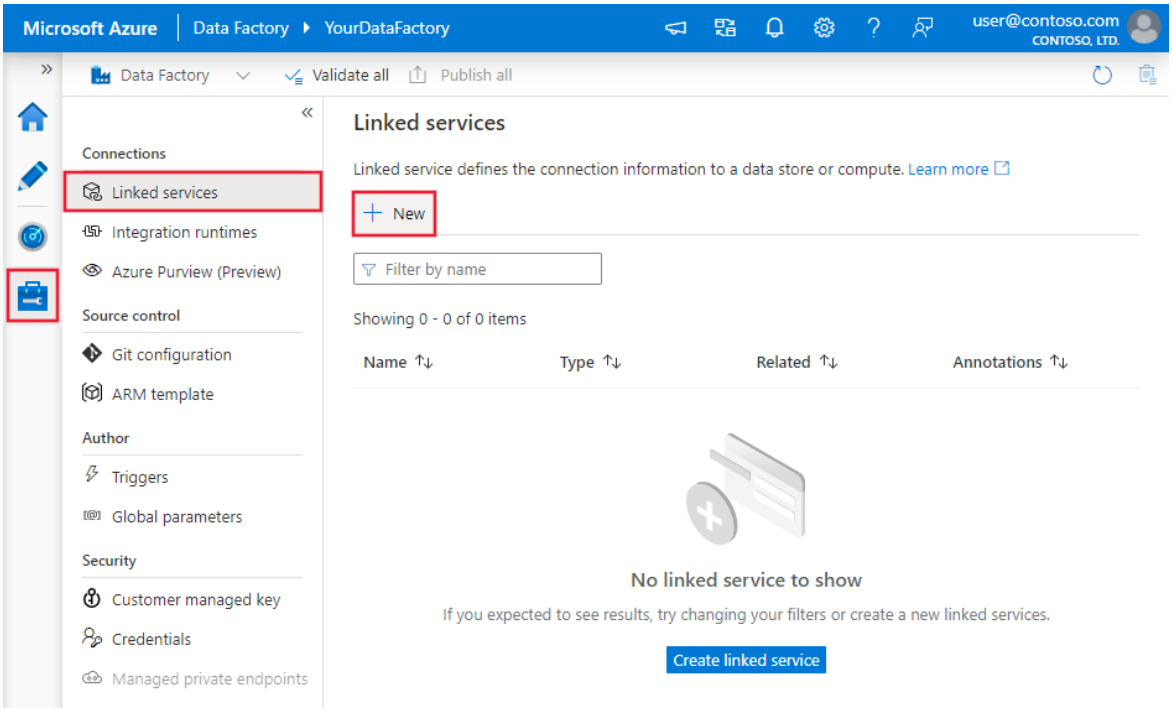
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Teradata using UI

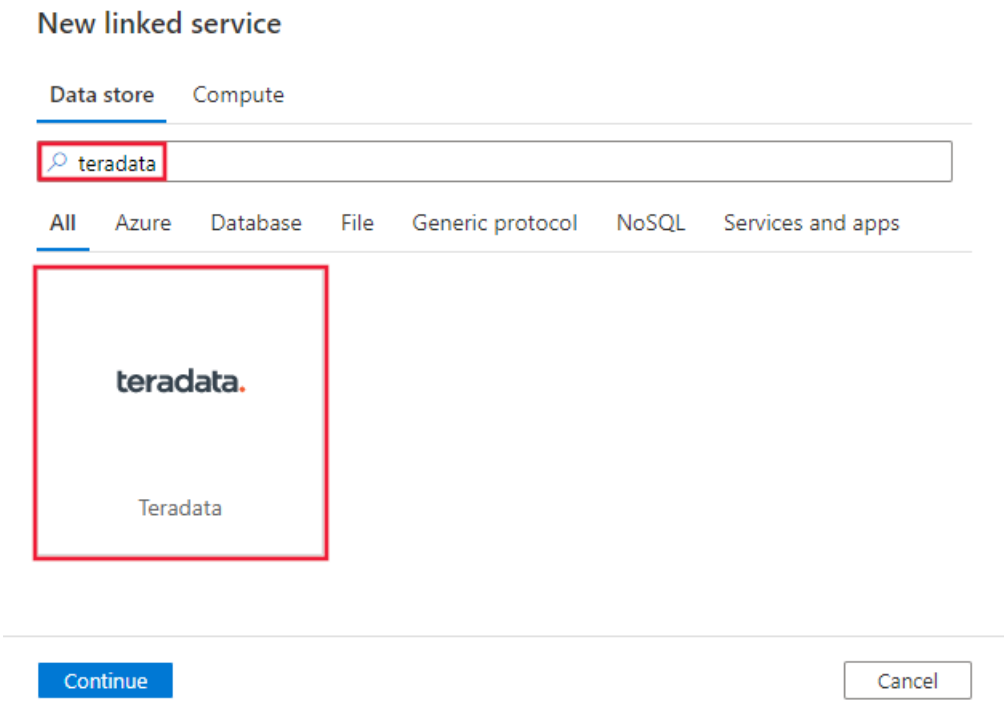
Use the following steps to create a linked service to Teradata in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

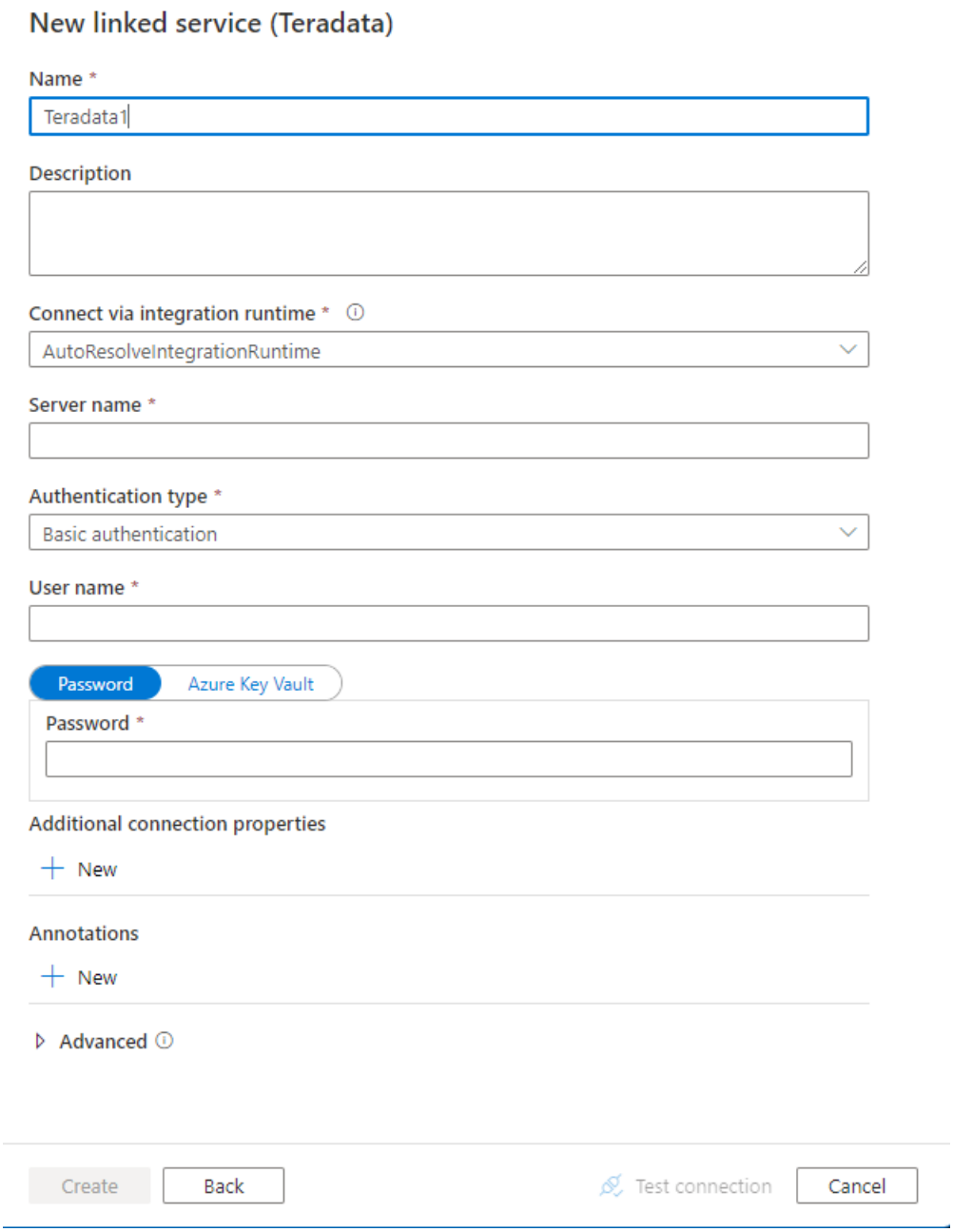
   - Azure Data Factory
   - Azure Synapse



2. Search for Teradata and select the Teradata connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to the Teradata connector.

Linked service properties

The Teradata linked service supports the following properties:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **Teradata**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	Specifies the information needed to connect to the Teradata instance. Refer to the following samples. You can also put a password in Azure Key Vault, and pull the `password` configuration out of the connection string. Refer to Store credentials in Azure Key Vault with more details.	Yes
username	Specify a user name to connect to Teradata. Applies when you are using Windows authentication.	No
password	Specify a password for the user account you specified for the user name. You can also choose to reference a secret stored in Azure Key Vault. Applies when you are using Windows authentication, or referencing a password in Key Vault for basic authentication.	No
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

More connection properties you can set in connection string per your case:

PROPERTY	DESCRIPTION	DEFAULT VALUE
TdmstPortNumber	The number of the port used to access Teradata database. Do not change this value unless instructed to do so by Technical Support.	1025
UseDataEncryption	Specifies whether to encrypt all communication with the Teradata database. Allowed values are 0 or 1. - **0 (disabled, default)**: Encrypts authentication information only. - **1 (enabled)**: Encrypts all data that is passed between the driver and the database.	`0`

PROPERTY	DESCRIPTION	DEFAULT VALUE
CharacterSet	The character set to use for the session. E.g., `CharacterSet=UTF16`. This value can be a user-defined character set, or one of the following pre-defined character sets: - ASCII - UTF8 - UTF16 - LATIN1252_0A - LATIN9_0A - LATIN1_0A - Shift-JIS (Windows, DOS compatible, KANJISJIS_0S) - EUC (Unix compatible, KANJIEC_0U) - IBM Mainframe (KANJIEBCDIC5035_0I) - KANJI932_1S0 - BIG5 (TCHBIG5_1R0) - GB (SCHGB2312_1T0) - SCHINESE936_6R0 - TCHINESE950_8R0 - NetworkKorean (HANGULKSC5601_2R4) - HANGUL949_7R0 - ARABIC1256_6A0 - CYRILLIC1251_2A0 - HEBREW1255_5A0 - LATIN1250_1A0 - LATIN1254_7A0 - LATIN1258_8A0 - THAI874_4A0	`ASCII`
MaxRespSize	The maximum size of the response buffer for SQL requests, in kilobytes (KBs). E.g., `MaxRespSize=10485760`. For Teradata Database version 16.00 or later, the maximum value is 7361536. For connections that use earlier versions, the maximum value is 1048576.	`65536`
MechanismName	To use the LDAP protocol to authenticate the connection, specify `MechanismName=LDAP`.	N/A

Example using basic authentication

```
{
    "name": "TeradataLinkedService",
    "properties": {
        "type": "Teradata",
        "typeProperties": {
            "connectionString": "DBCName=<server>;Uid=<username>;Pwd=<password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example using Windows authentication

```
{
    "name": "TeradataLinkedService",
    "properties": {
        "type": "Teradata",
        "typeProperties": {
            "connectionString": "DBCName=<server>",
            "username": "<username>",
            "password": "<password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example using LDAP authentication

```
{
    "name": "TeradataLinkedService",
    "properties": {
        "type": "Teradata",
        "typeProperties": {
            "connectionString": "DBCName=<server>;MechanismName=LDAP;Uid=<username>;Pwd=<password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

> **NOTE**
>
> The following payload is still supported. Going forward, however, you should use the new one.

Previous payload:

```
{
    "name": "TeradataLinkedService",
    "properties": {
        "type": "Teradata",
        "typeProperties": {
            "server": "<server>",
            "authenticationType": "<Basic/Windows>",
            "username": "<username>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

This section provides a list of properties supported by the Teradata dataset. For a full list of sections and properties available for defining datasets, see Datasets.

To copy data from Teradata, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to `TeradataTable`.	Yes
database	The name of the Teradata instance.	No (if "query" in activity source is specified)
table	The name of the table in the Teradata instance.	No (if "query" in activity source is specified)

Example:

```
{
    "name": "TeradataDataset",
    "properties": {
        "type": "TeradataTable",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Teradata linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

NOTE

`RelationalTable` type dataset is still supported. However, we recommend that you use the new dataset.

Previous payload:

```
{
    "name": "TeradataDataset",
    "properties": {
        "type": "RelationalTable",
        "linkedServiceName": {
            "referenceName": "<Teradata linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {}
    }
}
```

Copy activity properties

This section provides a list of properties supported by Teradata source. For a full list of sections and properties available for defining activities, see Pipelines.

Teradata as source

> **TIP**
> To load data from Teradata efficiently by using data partitioning, learn more from Parallel copy from Teradata section.

To copy data from Teradata, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to `TeradataSource`.	Yes
query	Use the custom SQL query to read data. An example is `"SELECT * FROM MyTable"`. When you enable partitioned load, you need to hook any corresponding built-in partition parameters in your query. For examples, see the Parallel copy from Teradata section.	No (if table in dataset is specified)
partitionOptions	Specifies the data partitioning options used to load data from Teradata. Allow values are: **None** (default), **Hash** and **DynamicRange**. When a partition option is enabled (that is, not `None`), the degree of parallelism to concurrently load data from Teradata is controlled by the `parallelCopies` setting on the copy activity.	No
partitionSettings	Specify the group of the settings for data partitioning. Apply when partition option isn't `None`.	No

PROPERTY	DESCRIPTION	REQUIRED
partitionColumnName	Specify the name of the source column that will be used by range partition or Hash partition for parallel copy. If not specified, the primary index of the table is autodetected and used as the partition column. Apply when the partition option is `Hash` or `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfHashPartitionCondition` or `?AdfRangePartitionColumnName` in WHERE clause. See example in Parallel copy from Teradata section.	No
partitionUpperBound	The maximum value of the partition column to copy data out. Apply when partition option is `DynamicRange`. If you use query to retrieve source data, hook `?AdfRangePartitionUpbound` in the WHERE clause. For an example, see the Parallel copy from Teradata section.	No
partitionLowerBound	The minimum value of the partition column to copy data out. Apply when the partition option is `DynamicRange`. If you use a query to retrieve the source data, hook `?AdfRangePartitionLowbound` in the WHERE clause. For an example, see the Parallel copy from Teradata section.	No

NOTE

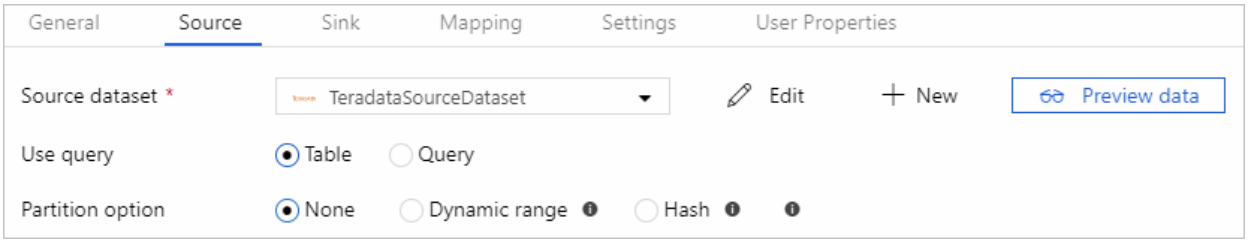
`RelationalSource` type copy source is still supported, but it doesn't support the new built-in parallel load from Teradata (partition options). However, we recommend that you use the new dataset.

Example: copy data by using a basic query without partition

```
"activities":[
    {
        "name": "CopyFromTeradata",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Teradata input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "TeradataSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Parallel copy from Teradata

The Teradata connector provides built-in data partitioning to copy data from Teradata in parallel. You can find data partitioning options on the **Source** table of the copy activity.



When you enable partitioned copy, the service runs parallel queries against your Teradata source to load data by partitions. The parallel degree is controlled by the `parallelCopies` setting on the copy activity. For example, if you set `parallelCopies` to four, the service concurrently generates and runs four queries based on your specified partition option and settings, and each query retrieves a portion of data from your Teradata.

You are suggested to enable parallel copy with data partitioning especially when you load large amount of data from your Teradata. The following are suggested configurations for different scenarios. When copying data into file-based data store, it's recommended to write to a folder as multiple files (only specify folder name), in which case the performance is better than writing to a single file.

SCENARIO	SUGGESTED SETTINGS
Full load from large table.	**Partition option**: Hash. During execution, the service automatically detects the primary index column, applies a hash against it, and copies data by partitions.

SCENARIO	SUGGESTED SETTINGS
Load large amount of data by using a custom query.	**Partition option**: Hash. **Query**: `SELECT * FROM <TABLENAME> WHERE ? AdfHashPartitionCondition AND <your_additional_where_clause>` . **Partition column**: Specify the column used for apply hash partition. If not specified, the service automatically detects the PK column of the table you specified in the Teradata dataset. During execution, the service replaces `?AdfHashPartitionCondition` with the hash partition logic, and sends to Teradata.
Load large amount of data by using a custom query, having an integer column with evenly distributed value for range partitioning.	**Partition options**: Dynamic range partition. **Query**: `SELECT * FROM <TABLENAME> WHERE ? AdfRangePartitionColumnName <= ? AdfRangePartitionUpbound AND ? AdfRangePartitionColumnName >= ? AdfRangePartitionLowbound AND <your_additional_where_clause>` . **Partition column**: Specify the column used to partition data. You can partition against the column with integer data type. **Partition upper bound** and **partition lower bound**: Specify if you want to filter against the partition column to retrieve data only between the lower and upper range. During execution, the service replaces `?AdfRangePartitionColumnName`, `?AdfRangePartitionUpbound`, and `?AdfRangePartitionLowbound` with the actual column name and value ranges for each partition, and sends to Teradata. For example, if your partition column "ID" set with the lower bound as 1 and the upper bound as 80, with parallel copy set as 4, the service retrieves data by 4 partitions. Their IDs are between [1,20], [21, 40], [41, 60], and [61, 80], respectively.

Example: query with hash partition

```
"source": {
    "type": "TeradataSource",
    "query":"SELECT * FROM <TABLENAME> WHERE ?AdfHashPartitionCondition AND <your_additional_where_clause>",
    "partitionOption": "Hash",
    "partitionSettings": {
        "partitionColumnName": "<hash_partition_column_name>"
    }
}
```

Example: query with dynamic range partition

```
"source": {
    "type": "TeradataSource",
    "query":"SELECT * FROM <TABLENAME> WHERE ?AdfRangePartitionColumnName <= ?AdfRangePartitionUpbound AND ?
AdfRangePartitionColumnName >= ?AdfRangePartitionLowbound AND <your_additional_where_clause>",
    "partitionOption": "DynamicRange",
    "partitionSettings": {
        "partitionColumnName": "<dynamic_range_partition_column_name>",
        "partitionUpperBound": "<upper_value_of_partition_column>",
        "partitionLowerBound": "<lower_value_of_partition_column>"
    }
}
```

Data type mapping for Teradata

When you copy data from Teradata, the following mappings apply from Teradata's data types to the internal data types used by the service. To learn about how the copy activity maps the source schema and data type to the sink, see Schema and data type mappings.

TERADATA DATA TYPE	INTERIM SERVICE DATA TYPE
BigInt	Int64
Blob	Byte[]
Byte	Byte[]
ByteInt	Int16
Char	String
Clob	String
Date	DateTime
Decimal	Decimal
Double	Double
Graphic	Not supported. Apply explicit cast in source query.
Integer	Int32
Interval Day	Not supported. Apply explicit cast in source query.
Interval Day To Hour	Not supported. Apply explicit cast in source query.
Interval Day To Minute	Not supported. Apply explicit cast in source query.
Interval Day To Second	Not supported. Apply explicit cast in source query.
Interval Hour	Not supported. Apply explicit cast in source query.
Interval Hour To Minute	Not supported. Apply explicit cast in source query.

TERADATA DATA TYPE	INTERIM SERVICE DATA TYPE
Interval Hour To Second	Not supported. Apply explicit cast in source query.
Interval Minute	Not supported. Apply explicit cast in source query.
Interval Minute To Second	Not supported. Apply explicit cast in source query.
Interval Month	Not supported. Apply explicit cast in source query.
Interval Second	Not supported. Apply explicit cast in source query.
Interval Year	Not supported. Apply explicit cast in source query.
Interval Year To Month	Not supported. Apply explicit cast in source query.
Number	Double
Period (Date)	Not supported. Apply explicit cast in source query.
Period (Time)	Not supported. Apply explicit cast in source query.
Period (Time With Time Zone)	Not supported. Apply explicit cast in source query.
Period (Timestamp)	Not supported. Apply explicit cast in source query.
Period (Timestamp With Time Zone)	Not supported. Apply explicit cast in source query.
SmallInt	Int16
Time	TimeSpan
Time With Time Zone	TimeSpan
Timestamp	DateTime
Timestamp With Time Zone	DateTime
VarByte	Byte[]
VarChar	String
VarGraphic	Not supported. Apply explicit cast in source query.
Xml	Not supported. Apply explicit cast in source query.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Transform data in Twilio (Preview) using Azure Data Factory or Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Data Flow to transform data in Twilio (Preview). To learn more, read the introductory article for Azure Data Factory or Azure Synapse Analytics.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Twilio connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Mapping data flow (source/-)	①

① Azure integration runtime ② Self-hosted integration runtime

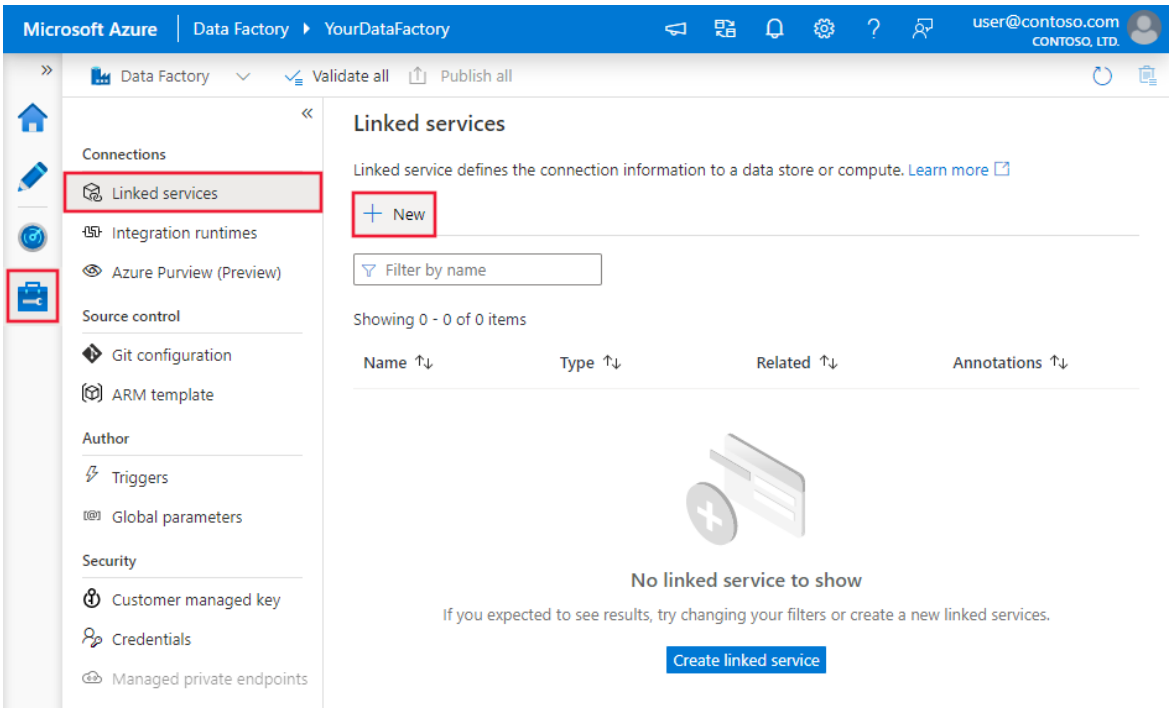
For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Create a Twilio linked service using UI

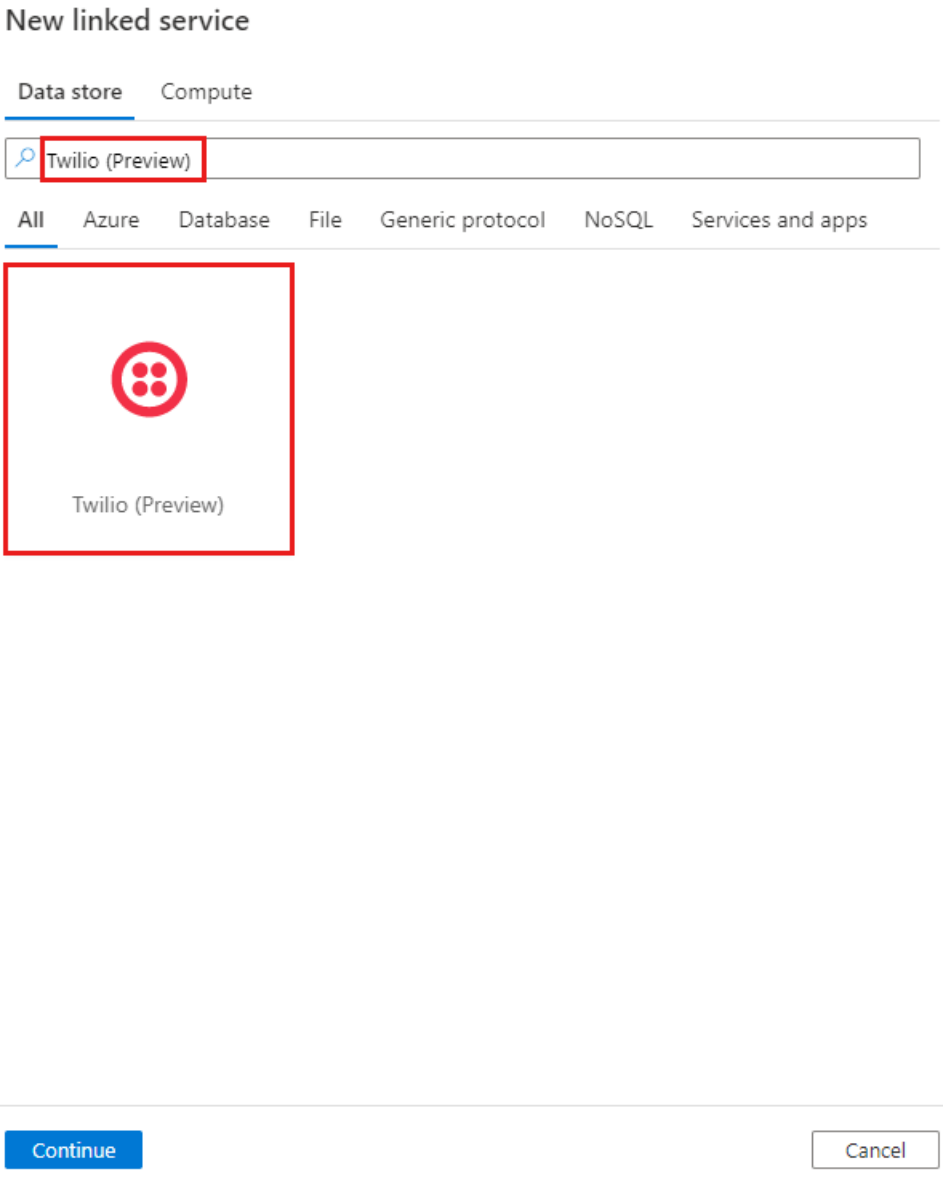
Use the following steps to create a Twilio linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then select New:

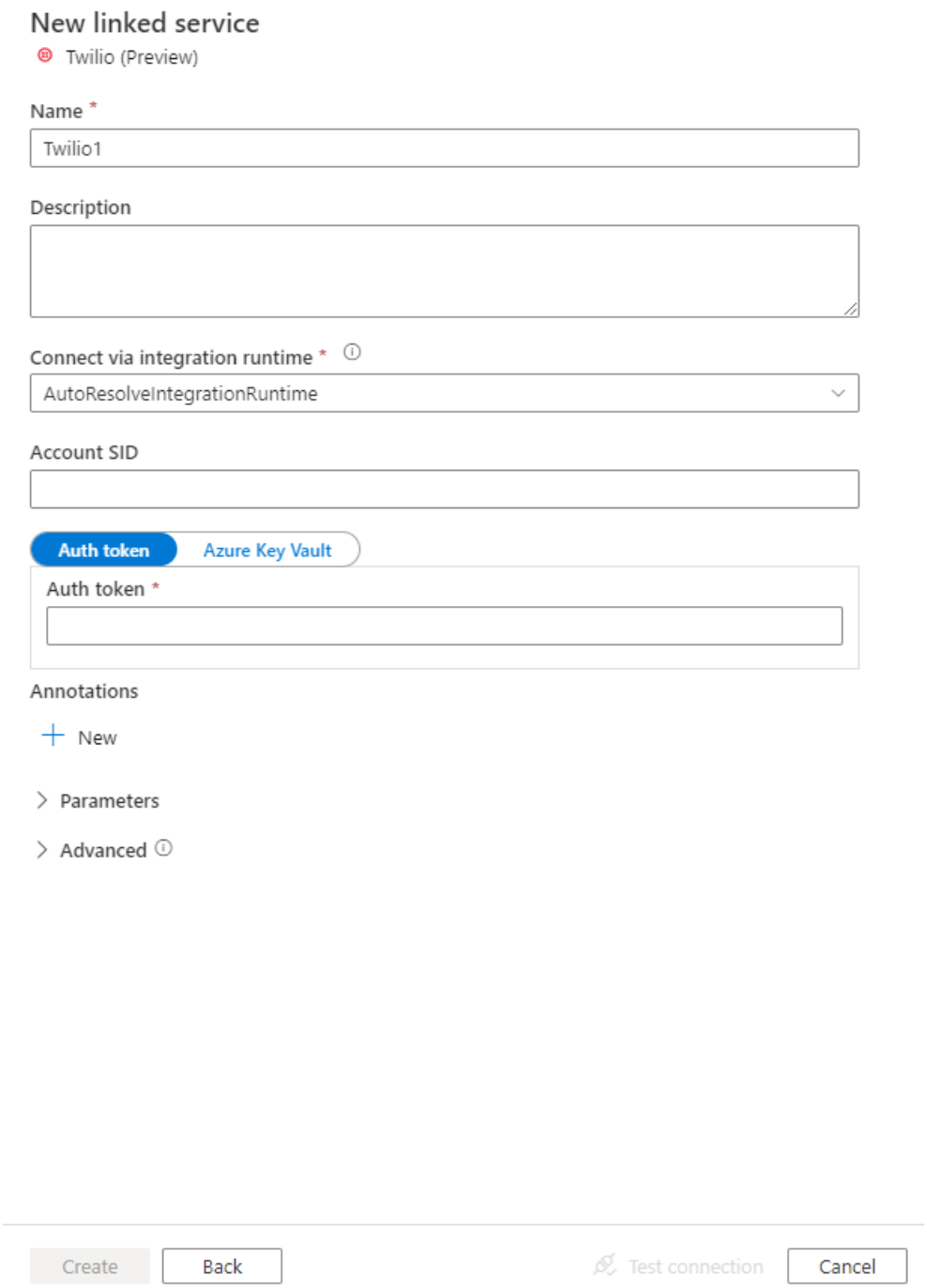
   - Azure Data Factory
   - Azure Synapse



2. Search for Twilio (Preview) and select the Twilio (Preview) connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide information about properties that are used to define Data Factory and Synapse pipeline entities specific to Twilio.

Linked service properties

The following properties are supported for the Twilio linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **Twilio**.	Yes
userName	The account SID of your Twilio account.	No

PROPERTY	DESCRIPTION	REQUIRED
password	The auth token of your Twilio account. Mark this field as **SecureString** to store it securely. Or, you can reference a secret stored in Azure Key Vault.	Yes

Example:

```
{
    "name": "TwilioLinkedService",
    "properties": {
        "type": "Twilio",
        "typeProperties": {
            "userName": "<account SID>",
            "password": {
                "type": "SecureString",
                "value": "<auth token>"
            }
        }
    }
}
```

Mapping data flow properties

When transforming data in mapping data flow, you can read resources from Twilio. For more information, see the source transformation in mapping data flows. You can only use an inline dataset as source type.

Source transformation

The below table lists the properties supported by Twilio source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Resource	The type of resources that data flow fetch from Twilio.	Yes	`Messages` `Calls`	resource
From	The phone number with country code, for example `+17755425856`.	No	String	from
To	The phone number with country code, for example `+17755425856`.	No	String	to

Twilio source script example

When you use Twilio as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
 validateSchema: false,
 store: 'twilio',
 format: 'rest',
 resource: 'Messages',
 from: '+17755425856') ~> TwilioSource
```

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from Vertica using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Vertica. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Vertica connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks by the copy activity, see the Supported data stores table.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Prerequisites

If your data store is located inside an on-premises network, an Azure virtual network, or Amazon Virtual Private Cloud, you need to configure a self-hosted integration runtime to connect to it.

If your data store is a managed cloud data service, you can use the Azure Integration Runtime. If the access is restricted to IPs that are approved in the firewall rules, you can add Azure Integration Runtime IPs to the allow list.

You can also use the managed virtual network integration runtime feature in Azure Data Factory to access the on-premises network without installing and configuring a self-hosted integration runtime.

For more information about the network security mechanisms and options supported by Data Factory, see Data access strategies.

Getting started

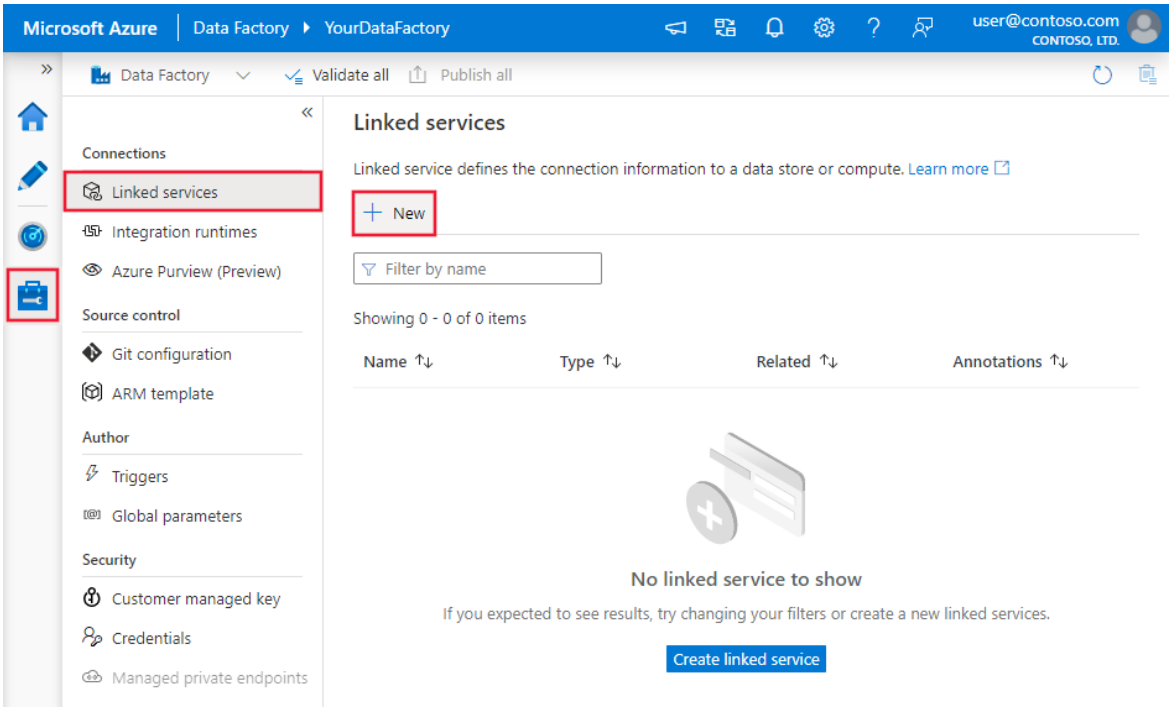
You can create a pipeline with copy activity using .NET SDK, Python SDK, Azure PowerShell, REST API, or Azure Resource Manager template. See Copy activity tutorial for step-by-step instructions to create a pipeline with a copy activity.

Create a linked service to Vertica using UI

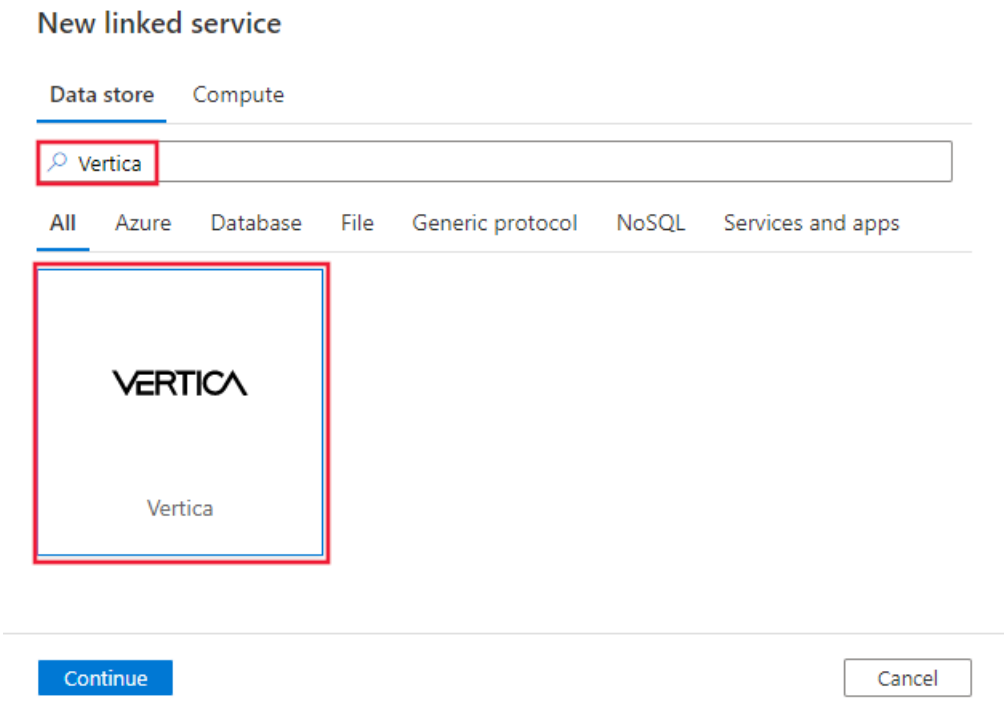
Use the following steps to create a linked service to Vertica in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

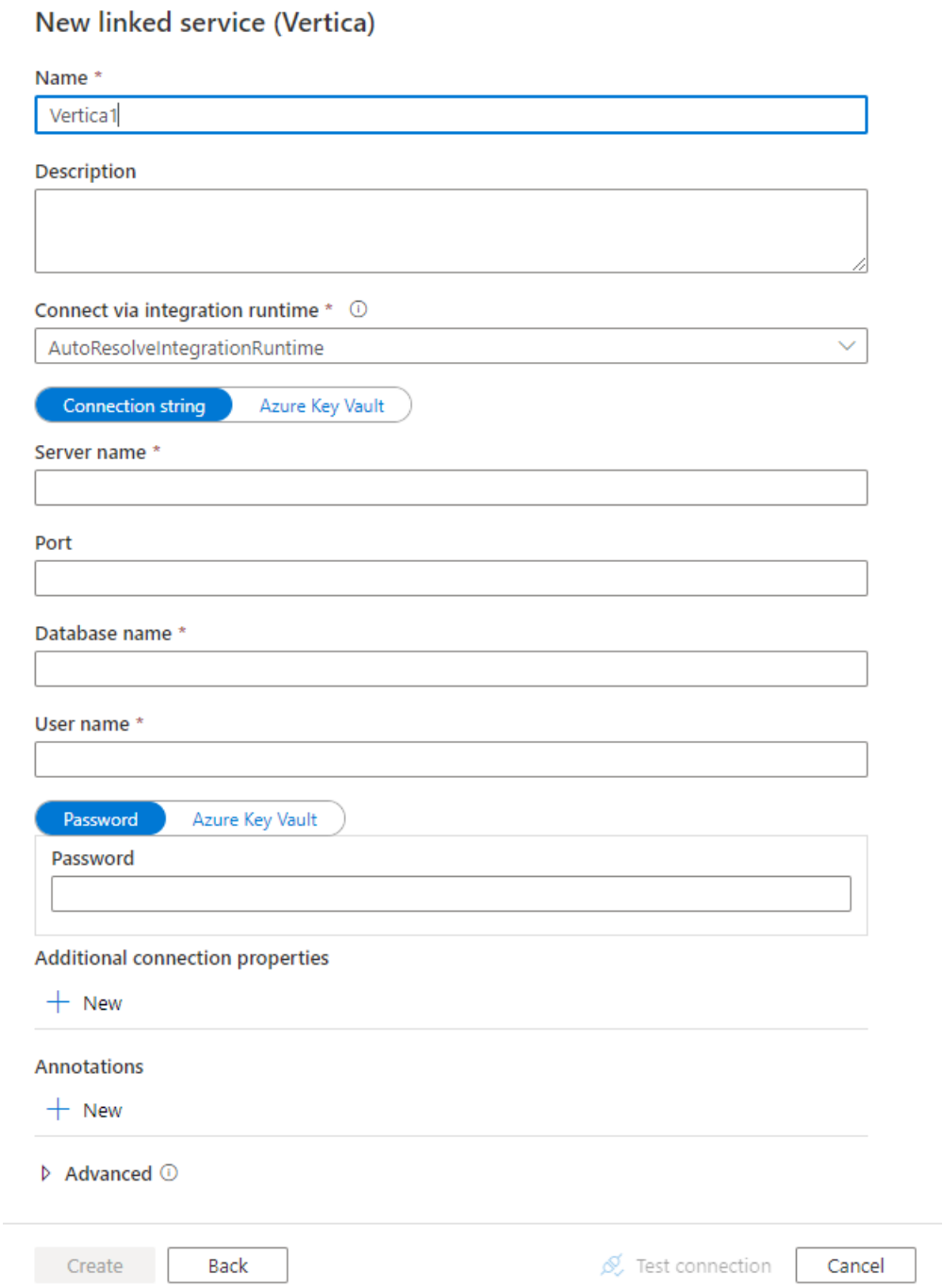
   - Azure Data Factory
   - Azure Synapse



2. Search for Vertica and select the Vertica connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Vertica connector.

Linked service properties

The following properties are supported for Vertica linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Vertica**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionString	An ODBC connection string to connect to Vertica. You can also put password in Azure Key Vault and pull the `pwd` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. Learn more from Prerequisites section. If not specified, it uses the default Azure Integration Runtime.	No

Example:

```
{
    "name": "VerticaLinkedService",
    "properties": {
        "type": "Vertica",
        "typeProperties": {
            "connectionString": "Server=<server>;Port=<port>;Database=<database>;UID=<user name>;PWD=
<password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

```
{
    "name": "VerticaLinkedService",
    "properties": {
        "type": "Vertica",
        "typeProperties": {
            "connectionString": "Server=<server>;Port=<port>;Database=<database>;UID=<user name>;",
            "pwd": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section

provides a list of properties supported by Vertica dataset.

To copy data from Vertica, set the type property of the dataset to **VerticaTable**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **VerticaTable**	Yes
schema	Name of the schema.	No (if "query" in activity source is specified)
table	Name of the table.	No (if "query" in activity source is specified)
tableName	Name of the table with schema. This property is supported for backward compatibility. Use `schema` and `table` for new workload.	No (if "query" in activity source is specified)

Example

```
{
    "name": "VerticaDataset",
    "properties": {
        "type": "VerticaTable",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Vertica linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Vertica source.

Vertica as source

To copy data from Vertica, set the source type in the copy activity to **VerticaSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **VerticaSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromVertica",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Vertica input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "VerticaSource",
                "query": "SELECT * FROM MyTable"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Web table by using Azure Data Factory or Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from a Web table database. It builds on the copy activity overview article that presents a general overview of copy activity.

The difference among this Web table connector, the REST connector and the HTTP connector are:

- **Web table connector** extracts table content from an HTML webpage.
- **REST connector** specifically support copying data from RESTful APIs.
- **HTTP connector** is generic to retrieve data from any HTTP endpoint, e.g. to download file.

Supported capabilities

This Web table connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	②
Lookup activity	②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Specifically, this Web table connector supports **extracting table content from an HTML page**.

Prerequisites

To use this Web table connector, you need to set up a Self-hosted Integration Runtime. See Self-hosted Integration Runtime article for details.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

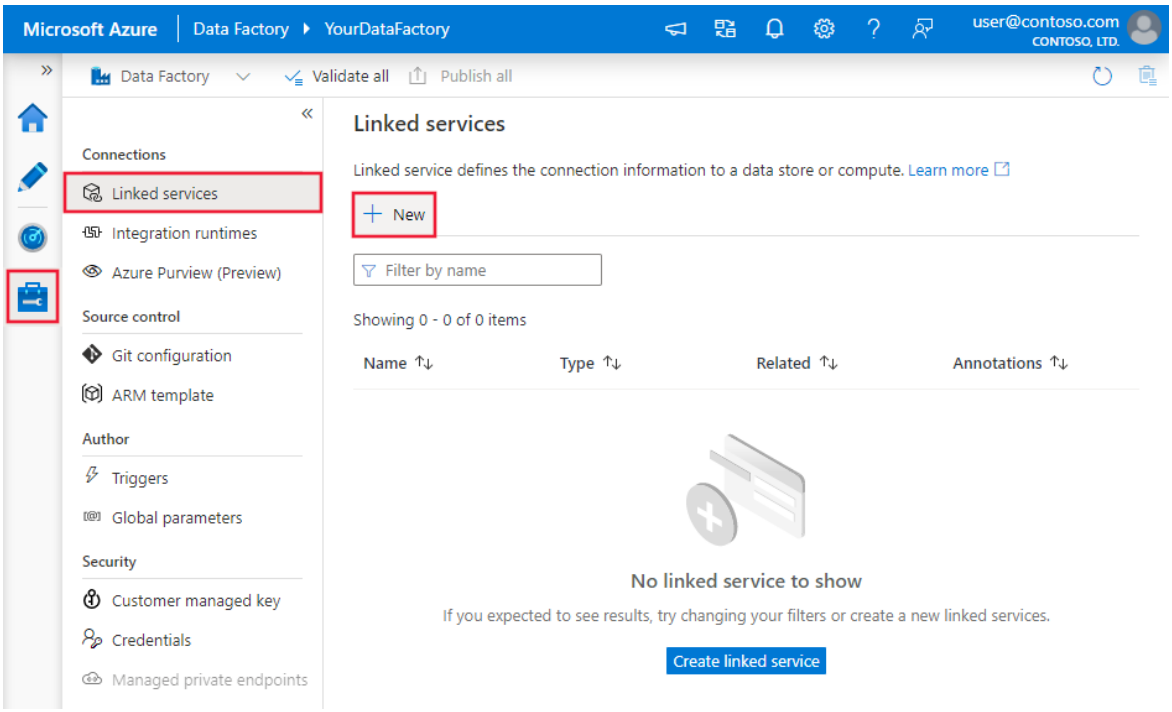
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Web Table using UI

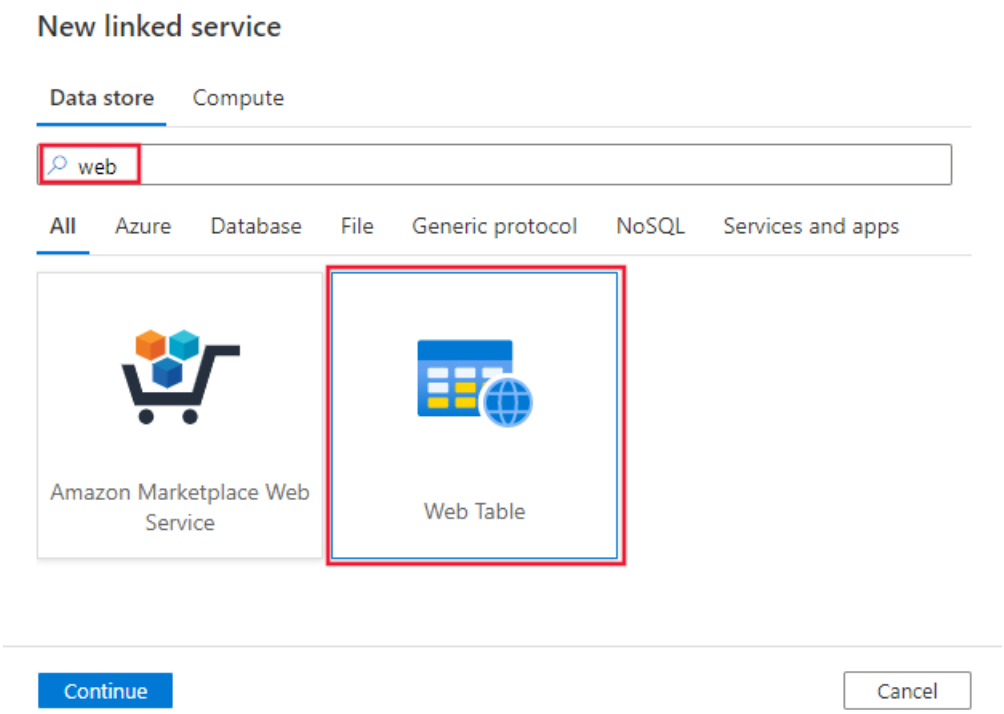
Use the following steps to create a linked service to Web Table in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

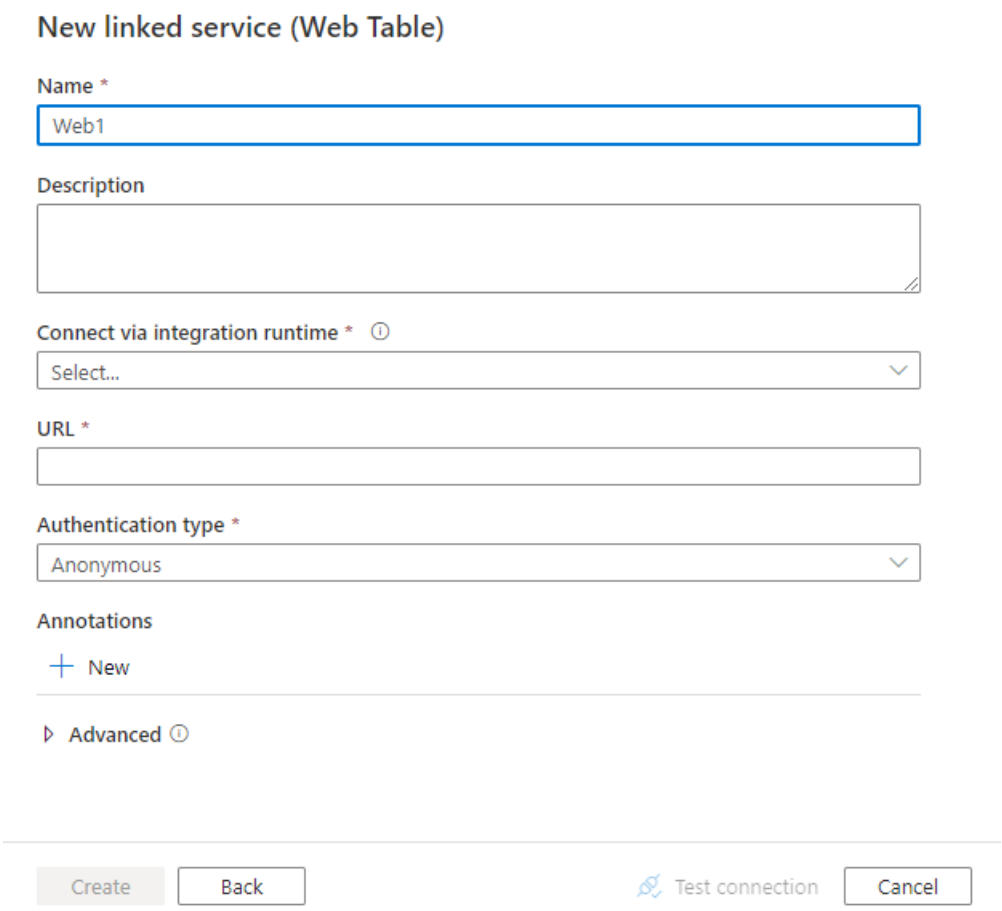
   - Azure Data Factory
   - Azure Synapse



2. Search for Web and select the Web Table connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Web table connector.

Linked service properties

The following properties are supported for Web table linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Web**	Yes
url	URL to the Web source	Yes
authenticationType	Allowed value is: **Anonymous**.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. A Self-hosted Integration Runtime is required as mentioned in Prerequisites.	Yes

Example:

```
{
    "name": "WebLinkedService",
    "properties": {
        "type": "Web",
        "typeProperties": {
            "url" : "https://en.wikipedia.org/wiki/",
            "authenticationType": "Anonymous"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Web table dataset.

To copy data from Web table, set the type property of the dataset to **WebTable**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **WebTable**	Yes
path	A relative URL to the resource that contains the table.	No. When path is not specified, only the URL specified in the linked service definition is used.
index	The index of the table in the resource. See Get index of a table in an HTML page section for steps to getting index of a table in an HTML page.	Yes

Example:

```
{
    "name": "WebTableInput",
    "properties": {
        "type": "WebTable",
        "typeProperties": {
            "index": 1,
            "path": "AFI's_100_Years...100_Movies"
        },
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Web linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section

provides a list of properties supported by Web table source.

Web table as source

To copy data from Web table, set the source type in the copy activity to **WebSource**, no additional properties are supported.

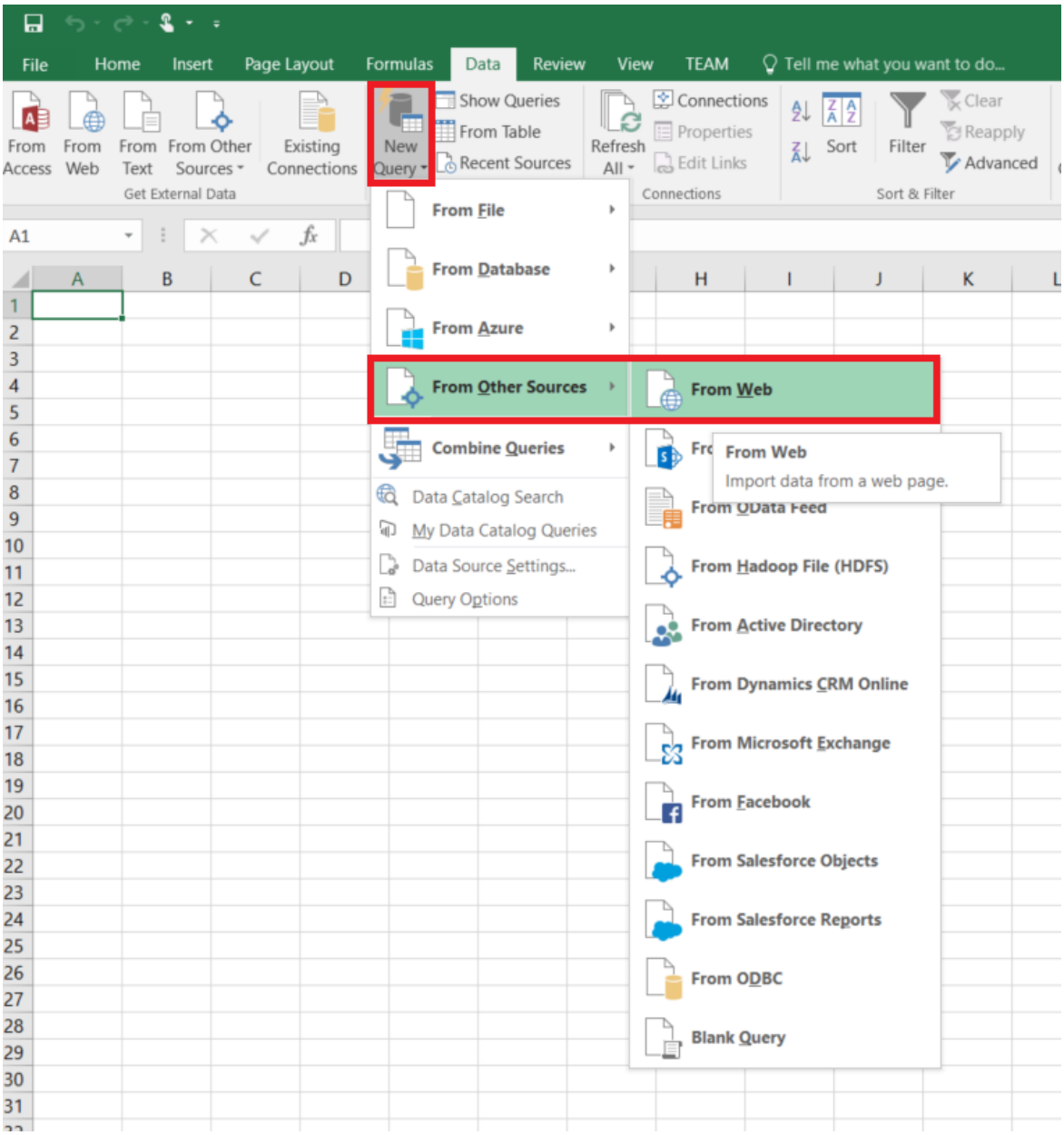
Example:

```
"activities":[
    {
        "name": "CopyFromWebTable",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Web table input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "WebSource"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

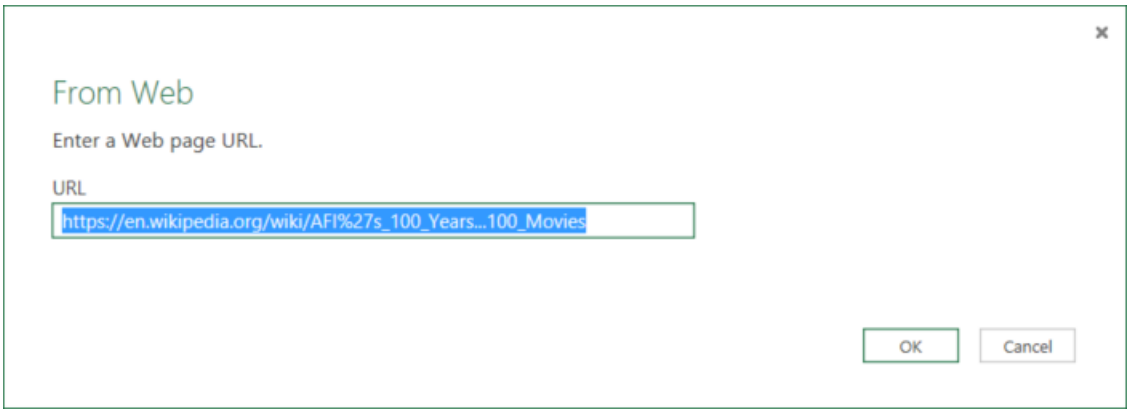
Get index of a table in an HTML page

To get the index of a table which you need to configure in dataset properties, you can use e.g. Excel 2016 as the tool as follows:

1. Launch **Excel 2016** and switch to the **Data** tab.
2. Click **New Query** on the toolbar, point to **From Other Sources** and click **From Web**.

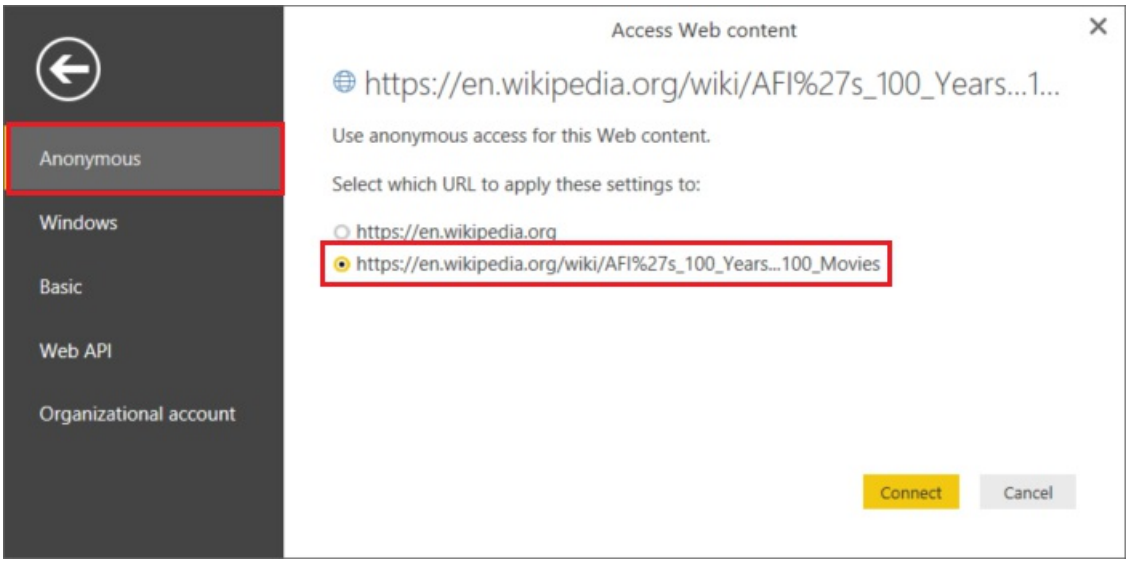


3. In the **From Web** dialog box, enter **URL** that you would use in linked service JSON (for example: https://en.wikipedia.org/wiki/) along with path you would specify for the dataset (for example: AFI%27s_100_Years...100_Movies), and click **OK**.

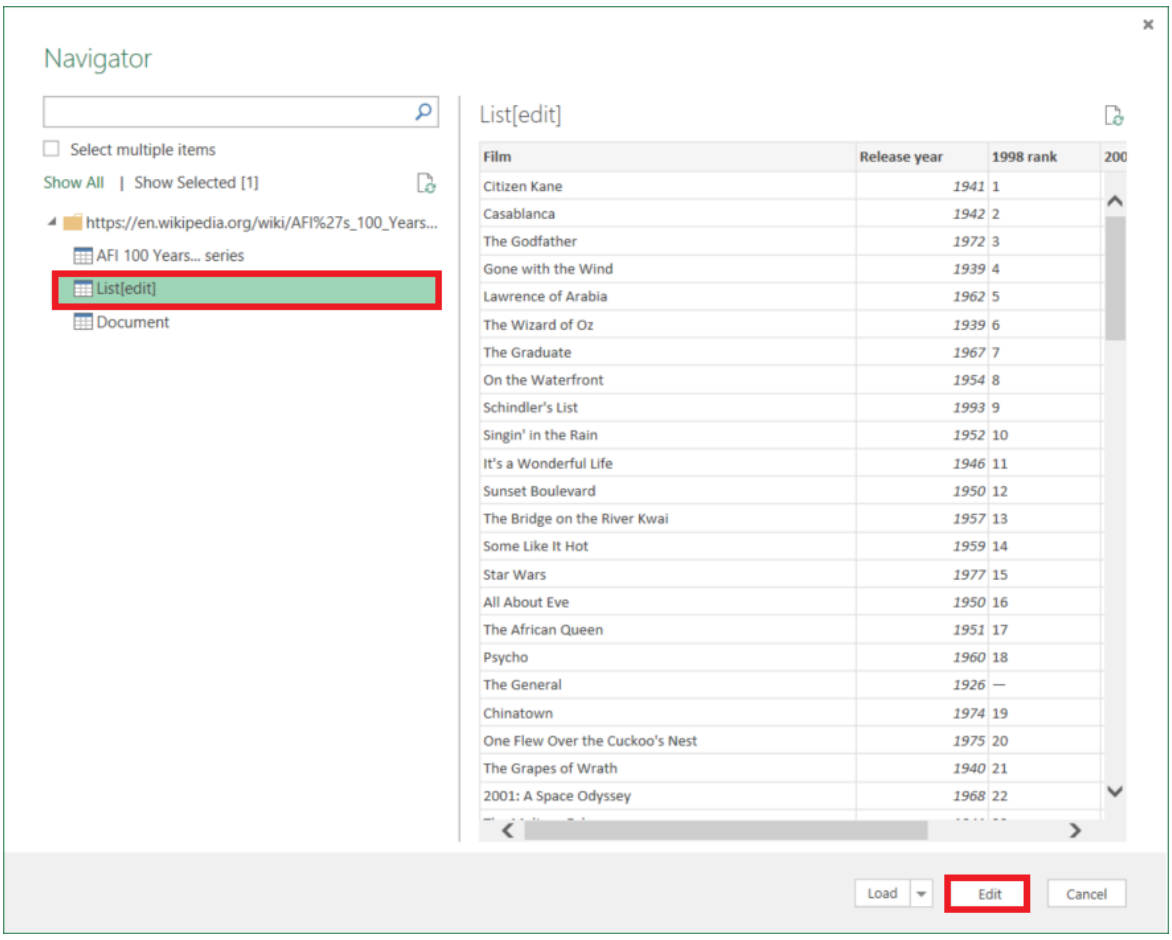


URL used in this example: https://en.wikipedia.org/wiki/AFI%27s_100_Years...100_Movies

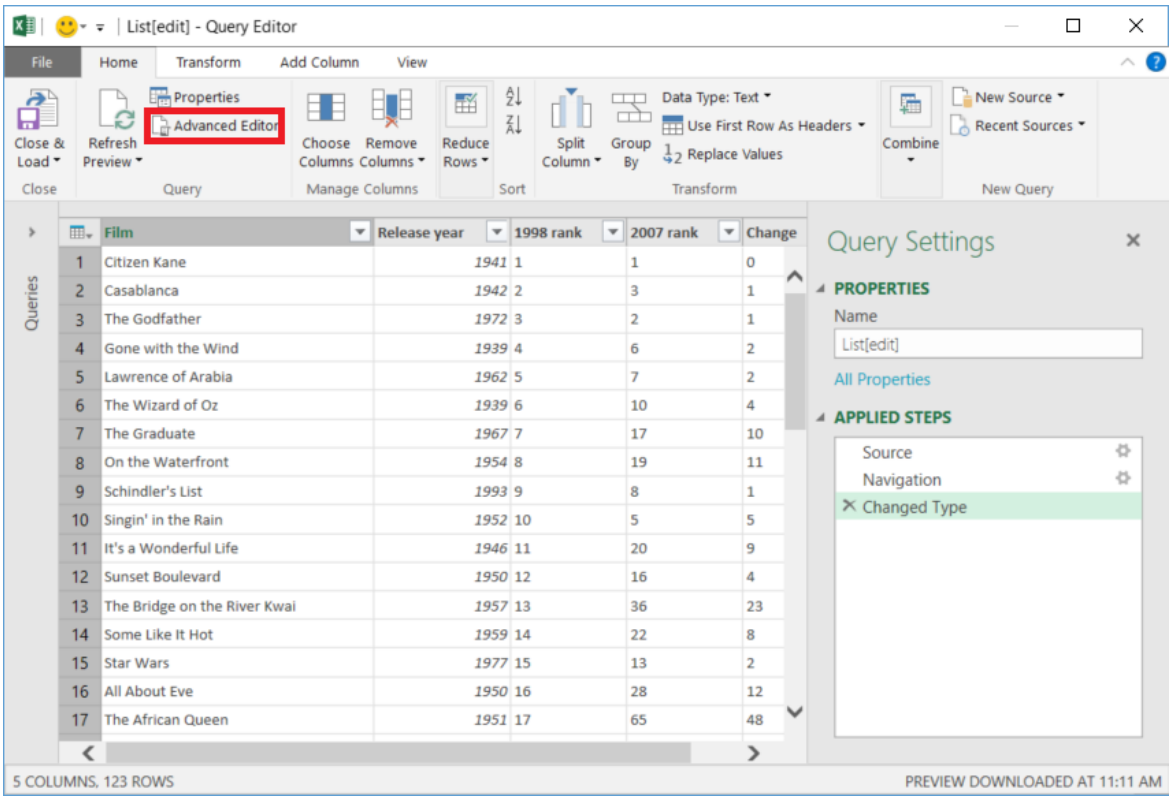
4. If you see **Access Web content** dialog box, select the right **URL**, **authentication**, and click **Connect**.



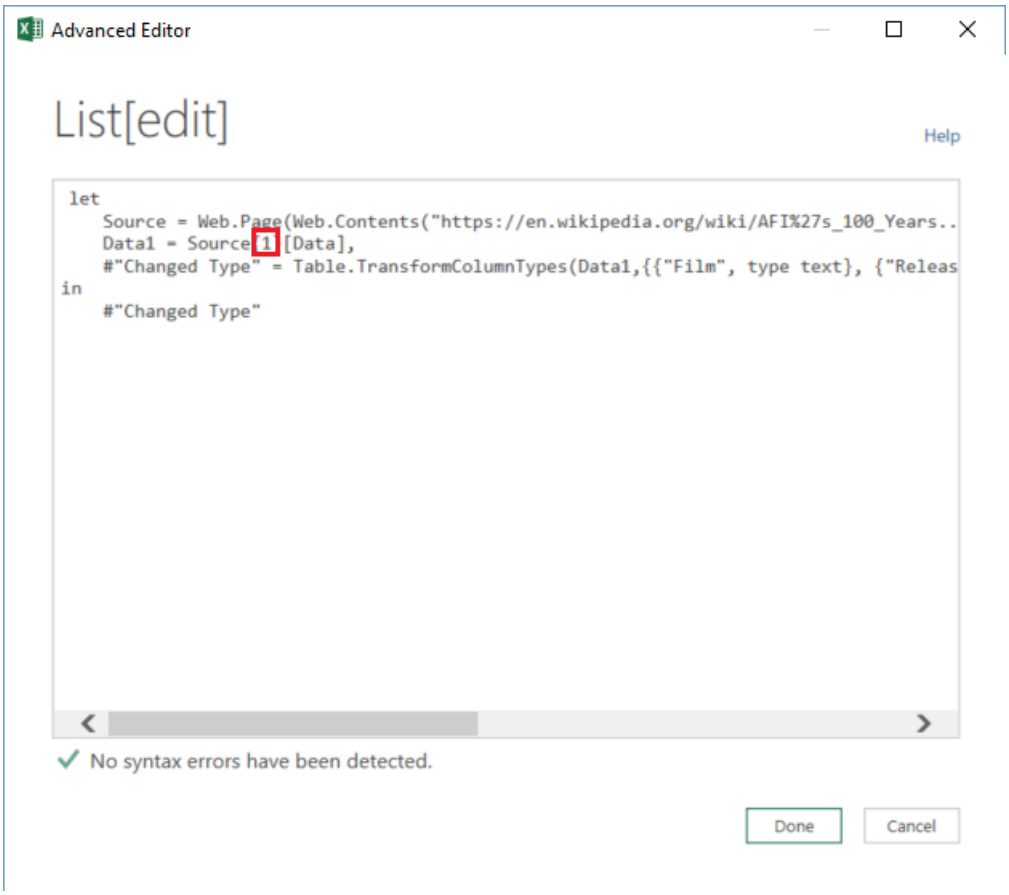
5. Click a **table** item in the tree view to see content from the table and then click **Edit** button at the bottom.



6. In the **Query Editor** window, click **Advanced Editor** button on the toolbar.



7. In the Advanced Editor dialog box, the number next to "Source" is the index.



If you are using Excel 2013, use Microsoft Power Query for Excel to get the index. See Connect to a web page article for details. The steps are similar if you are using Microsoft Power BI for Desktop.

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy data from Xero using Azure Data Factory or Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Xero. It builds on the copy activity overview article that presents a general overview of copy activity.

Supported capabilities

This Xero connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Specifically, this Xero connector supports:

- OAuth 2.0 and OAuth 1.0 authentication. For OAuth 1.0, the connector supports Xero private application but not public application.
- All Xero tables (API endpoints) except "Reports".

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

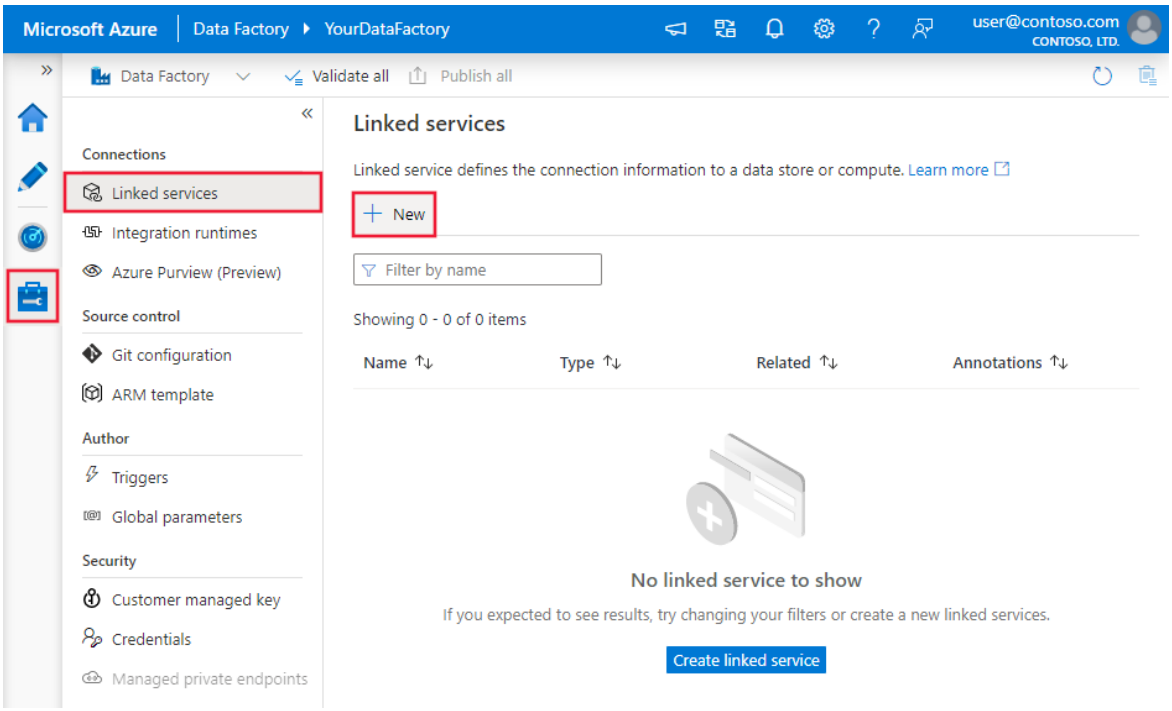
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Xero using UI

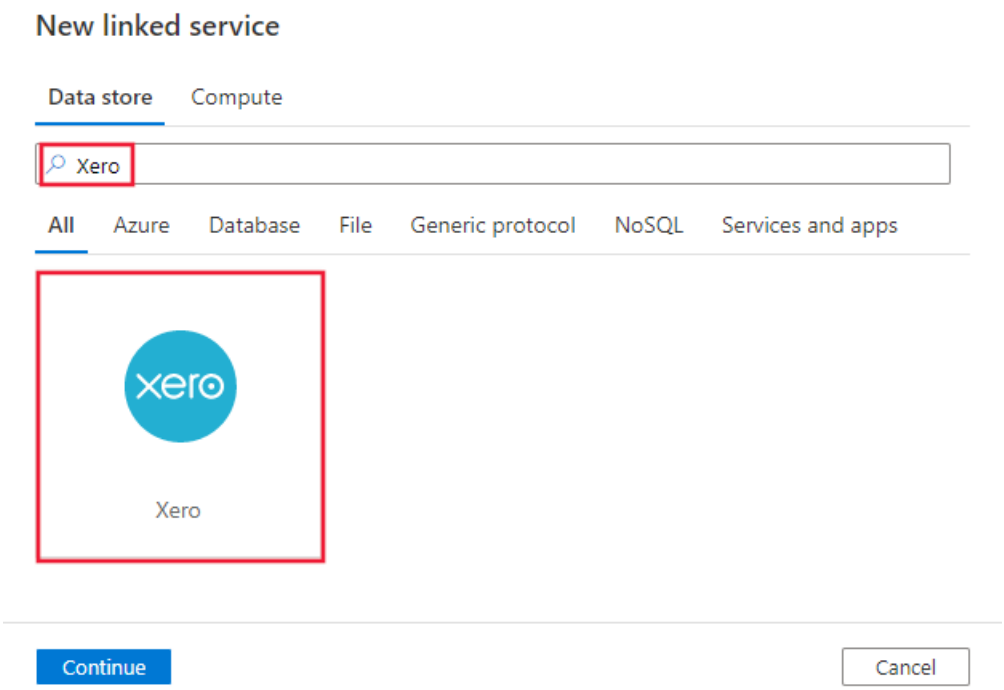
Use the following steps to create a linked service to Xero in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

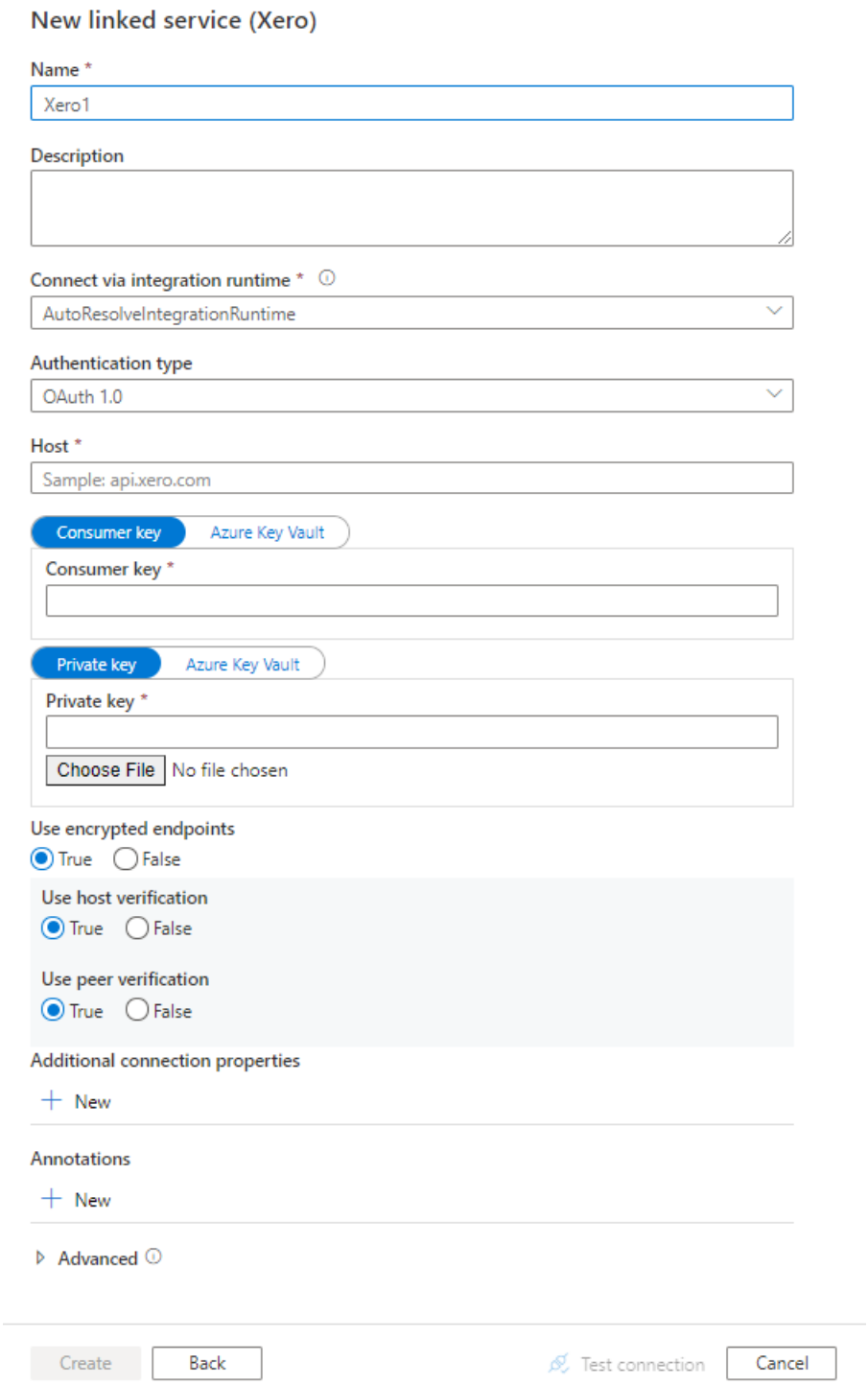
   - Azure Data Factory
   - Azure Synapse



2. Search for Xero and select the Xero connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Xero connector.

Linked service properties

The following properties are supported for Xero linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Xero**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionProperties	A group of properties that defines how to connect to Xero.	Yes
Under `connectionProperties`*:*		
host	The endpoint of the Xero server (`api.xero.com`).	Yes
authenticationType	Allowed values are `OAuth_2.0` and `OAuth_1.0` .	Yes
consumerKey	For OAuth 2.0, specify the **client ID** for your Xero application. For OAuth 1.0, specify the consumer key associated with the Xero application. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
privateKey	For OAuth 2.0, specify the **client secret** for your Xero application. For OAuth 1.0, specify the private key from the .pem file that was generated for your Xero private application. Note to **generate the privatekey.pem with numbits of 512** using `openssl genrsa -out privatekey.pem 512` , 1024 is not supported. Include all the text from the .pem file including the Unix line endings(\n), see sample below. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
tenantId	The tenant ID associated with your Xero application. Applicable for OAuth 2.0 authentication. Learn how to get the tenant ID from Check the tenants you're authorized to access section.	Yes for OAuth 2.0 authentication

PROPERTY	DESCRIPTION	REQUIRED
refreshToken	Applicable for OAuth 2.0 authentication. TheOAuth 2.0 refreshtoken is associated withtheXero application and used to refresh the accesstoken; the access token expires after 30 minutes. Learn about how the Xero authorization flow works and how to get the refresh token from this article. To get a refresh token, you must request the offline_access scope. **Know limitation**: Note Xero resets the refresh token after it's used for access token refresh. For operationalized workload, before each copy activity run, you need to set a valid refresh token for the service to use. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes for OAuth 2.0 authentication
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether the host name is required in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example: OAuth 2.0 authentication

```
{
    "name": "XeroLinkedService",
    "properties": {
        "type": "Xero",
        "typeProperties": {
            "connectionProperties": {
                "host":"api.xero.com",
                "authenticationType":"OAuth_2.0",
                "consumerKey": {
                    "type": "SecureString",
                    "value": "<client ID>"
                },
                "privateKey": {
                    "type": "SecureString",
                    "value": "<client secret>"
                },
                "tenantId":"<tenant ID>",
                "refreshToken":{
                    "type": "SecureString",
                    "value": "<refresh token>"
                },
                "useEncryptedEndpoints":true,
                "useHostVerification":true,
                "usePeerVerification":true
            }
        }
    }
}
```

Example: OAuth 1.0 authentication

```
{
    "name": "XeroLinkedService",
    "properties": {
        "type": "Xero",
        "typeProperties": {
            "connectionProperties": {
                "host":"api.xero.com",
                "authenticationType":"OAuth_1.0",
                "consumerKey": {
                    "type": "SecureString",
                    "value": "<consumer key>"
                },
                "privateKey": {
                    "type": "SecureString",
                    "value": "<private key>"
                },
                "useEncryptedEndpoints":true,
                "useHostVerification":true,
                "usePeerVerification":true
}
        }
    }
}
```

Sample private key value:

Include all the text from the .pem file including the Unix line endings(\n).

```
"-----BEGIN RSA PRIVATE KEY-----
\nMII***********************************************P\nbu***********************************************
*****s\nU/***********************************************B\nA***********************************************
**********W\njH***********************************************e\nsx***********************************
*****************l\nq***********************************************X\nh***************************
************************i\nd***********************************************s\nA**********************
*****************************dsfb\nN***********************************************M\np*************
************************************Ly\nK***********************************************Y=\n-----
END RSA PRIVATE KEY-----"
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Xero dataset.

To copy data from Xero, set the type property of the dataset to **XeroObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **XeroObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "XeroDataset",
    "properties": {
        "type": "XeroObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Xero linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Xero source.

Xero as source

To copy data from Xero, set the source type in the copy activity to **XeroSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **XeroSource**	Yes

PROPERTY	DESCRIPTION	REQUIRED
query	Use the custom SQL query to read data. For example: `"SELECT * FROM Contacts"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromXero",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Xero input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "XeroSource",
                "query": "SELECT * FROM Contacts"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Note the following when specifying the Xero query:

- Tables with complex items will be split to multiple tables. For example, Bank transactions has a complex data structure "LineItems", so data of bank transaction is mapped to table `Bank_Transaction` and `Bank_Transaction_Line_Items`, with `Bank_Transaction_ID` as foreign key to link them together.
- Xero data is available through two schemas: `Minimal` (default) and `Complete`. The Complete schema contains prerequisite call tables which require additional data (e.g. ID column) before making the desired query.

The following tables have the same information in the Minimal and Complete schema. To reduce the number of API calls, use Minimal schema (default).

- Bank_Transactions
- Contact_Groups
- Contacts
- Contacts_Sales_Tracking_Categories
- Contacts_Phones
- Contacts_Addresses
- Contacts_Purchases_Tracking_Categories
- Credit_Notes
- Credit_Notes_Allocations

- Expense_Claims
- Expense_Claim_Validation_Errors
- Invoices
- Invoices_Credit_Notes
- Invoices_ Prepayments
- Invoices_Overpayments
- Manual_Journals
- Overpayments
- Overpayments_Allocations
- Prepayments
- Prepayments_Allocations
- Receipts
- Receipt_Validation_Errors
- Tracking_Categories

The following tables can only be queried with complete schema:

- Complete.Bank_Transaction_Line_Items
- Complete.Bank_Transaction_Line_Item_Tracking
- Complete.Contact_Group_Contacts
- Complete.Contacts_Contact_ Persons
- Complete.Credit_Note_Line_Items
- Complete.Credit_Notes_Line_Items_Tracking
- Complete.Expense_Claim_ Payments
- Complete.Expense_Claim_Receipts
- Complete.Invoice_Line_Items
- Complete.Invoices_Line_Items_Tracking
- Complete.Manual_Journal_Lines
- Complete.Manual_Journal_Line_Tracking
- Complete.Overpayment_Line_Items
- Complete.Overpayment_Line_Items_Tracking
- Complete.Prepayment_Line_Items
- Complete.Prepayment_Line_Item_Tracking
- Complete.Receipt_Line_Items
- Complete.Receipt_Line_Item_Tracking
- Complete.Tracking_Category_Options

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

For a list of supported data stores by the copy activity, see supported data stores.

XML format in Azure Data Factory and Synapse Analytics pipelines

8/19/2022 • 8 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Follow this article when you want to **parse the XML files**.

XML format is supported for the following connectors: Amazon S3, Amazon S3 Compatible Storage, Azure Blob, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure Files, File System, FTP, Google Cloud Storage, HDFS, HTTP, Oracle Cloud Storage and SFTP. It is supported as source but not sink.

Dataset properties

For a full list of sections and properties available for defining datasets, see the Datasets article. This section provides a list of properties supported by the XML dataset.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to **Xml**.	Yes
location	Location settings of the file(s). Each file-based connector has its own location type and supported properties under `location`. **See details in connector article -> Dataset properties section**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
encodingName	The encoding type used to read/write test files. Allowed values are as follows: "UTF-8", "UTF-16", "UTF-16BE", "UTF-32", "UTF-32BE", "US-ASCII", "UTF-7", "BIG5", "EUC-JP", "EUC-KR", "GB2312", "GB18030", "JOHAB", "SHIFT-JIS", "CP875", "CP866", "IBM00858", "IBM037", "IBM273", "IBM437", "IBM500", "IBM737", "IBM775", "IBM850", "IBM852", "IBM855", "IBM857", "IBM860", "IBM861", "IBM863", "IBM864", "IBM865", "IBM869", "IBM870", "IBM01140", "IBM01141", "IBM01142", "IBM01143", "IBM01144", "IBM01145", "IBM01146", "IBM01147", "IBM01148", "IBM01149", "ISO-2022-JP", "ISO-2022-KR", "ISO-8859-1", "ISO-8859-2", "ISO-8859-3", "ISO-8859-4", "ISO-8859-5", "ISO-8859-6", "ISO-8859-7", "ISO-8859-8", "ISO-8859-9", "ISO-8859-13", "ISO-8859-15", "WINDOWS-874", "WINDOWS-1250", "WINDOWS-1251", "WINDOWS-1252", "WINDOWS-1253", "WINDOWS-1254", "WINDOWS-1255", "WINDOWS-1256", "WINDOWS-1257", "WINDOWS-1258".	No
nullValue	Specifies the string representation of null value. The default value is **empty string**.	No
compression	Group of properties to configure file compression. Configure this section when you want to do compression/decompression during activity execution.	No

PROPERTY	DESCRIPTION	REQUIRED
type (*under* `compression`)	The compression codec used to read/write XML files. Allowed values are **bzip2**, **gzip**, **deflate**, **ZipDeflate**, **TarGzip**, **Tar**, **snappy**, or **lz4**. Default is not compressed. **Note** currently Copy activity doesn't support "snappy" & "lz4", and mapping data flow doesn't support "ZipDeflate", "TarGzip" and "Tar". **Note** when using copy activity to decompress **ZipDeflate**/**TarGzip**/**Tar** file(s) and write to file-based sink data store, by default files are extracted to the folder: `<path specified in dataset>/<folder named as source compressed file>/`, use `preserveZipFileNameAsFolder` / `preserveCompressionFileNameAsFolder` on copy activity source to control whether to preserve the name of the compressed file(s) as folder structure.	No.
level (*under* `compression`)	The compression ratio. Allowed values are **Optimal** or **Fastest**. - **Fastest:** The compression operation should complete as quickly as possible, even if the resulting file is not optimally compressed. - **Optimal**: The compression operation should be optimally compressed, even if the operation takes a longer time to complete. For more information, see Compression Level topic.	No

Below is an example of XML dataset on Azure Blob Storage:

```
{
    "name": "XMLDataset",
    "properties": {
        "type": "Xml",
        "linkedServiceName": {
            "referenceName": "<Azure Blob Storage linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "container": "containername",
                "folderPath": "folder/subfolder",
            },
            "compression": {
                "type": "ZipDeflate"
            }
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by the XML source.

Learn about how to map XML data and sink data store/format from schema mapping. When previewing XML files, data is shown with JSON hierarchy, and you use JSON path to point to the fields.

XML as source

The following properties are supported in the copy activity ***source*** section. Learn more from XML connector behavior.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to **XmlSource**.	Yes
formatSettings	A group of properties. Refer to **XML read settings** table below.	No
storeSettings	A group of properties on how to read data from a data store. Each file-based connector has its own supported read settings under `storeSettings`. **See details in connector article -> Copy activity properties section**.	No

Supported **XML read settings** under `formatSettings`:

PROPERTY	DESCRIPTION	REQUIRED
type	The type of formatSettings must be set to **XmlReadSettings**.	Yes
validationMode	Specifies whether to validate the XML schema. Allowed values are **none** (default, no validation), **xsd** (validate using XSD), **dtd** (validate using DTD).	No
namespaces	Whether to enable namespace when parsing the XML files. Allowed values are: **true** (default), **false**.	No
namespacePrefixes	Namespace URI to prefix mapping, which is used to name fields when parsing the xml file. If an XML file has namespace and namespace is enabled, by default, the field name is the same as it is in the XML document. If there is an item defined for the namespace URI in this map, the field name is `prefix:fieldName`.	No
detectDataType	Whether to detect integer, double, and Boolean data types. Allowed values are: **true** (default), **false**.	No

PROPERTY	DESCRIPTION	REQUIRED
compressionProperties	A group of properties on how to decompress data for a given compression codec.	No
preserveZipFileNameAsFolder (*under* `compressionProperties` -> `type` *as* `ZipDeflateReadSettings`)	Applies when input dataset is configured with **ZipDeflate** compression. Indicates whether to preserve the source zip file name as folder structure during copy. - When set to **true (default)**, the service writes unzipped files to `<path specified in dataset>/<folder named as source zip file>/`. - When set to **false**, the service writes unzipped files directly to `<path specified in dataset>`. Make sure you don't have duplicated file names in different source zip files to avoid racing or unexpected behavior.	No
preserveCompressionFileNameAsFolder (*under* `compressionProperties` -> `type` *as* `TarGZipReadSettings` *or* `TarReadSettings`)	Applies when input dataset is configured with **TarGzip/Tar** compression. Indicates whether to preserve the source compressed file name as folder structure during copy. - When set to **true (default)**, the service writes decompressed files to `<path specified in dataset>/<folder named as source compressed file>/`. - When set to **false**, the service writes decompressed files directly to `<path specified in dataset>`. Make sure you don't have duplicated file names in different source files to avoid racing or unexpected behavior.	No

Mapping data flow properties

In mapping data flows, you can read XML format in the following data stores: Azure Blob Storage, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Amazon S3 and SFTP. You can point to XML files either using XML dataset or using an inline dataset.

Source properties

The below table lists the properties supported by an XML source. You can edit these properties in the **Source options** tab. Learn more from XML connector behavior. When using inline dataset, you will see additional file settings, which are the same as the properties described in dataset properties section.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Wild card paths	All files matching the wildcard path will be processed. Overrides the folder and file path set in the dataset.	No	String[]	wildcardPaths
Partition root path	For file data that is partitioned, you can enter a partition root path in order to read partitioned folders as columns	No	String	partitionRootPath
List of files	Whether your source is pointing to a text file that lists files to process	No	`true` or `false`	fileList
Column to store file name	Create a new column with the source file name and path	No	String	rowUrlColumn
After completion	Delete or move the files after processing. File path starts from the container root	No	Delete: `true` or `false` Move: `['<from>', '<to>']`	purgeFiles moveFiles
Filter by last modified	Choose to filter files based upon when they were last altered	No	Timestamp	modifiedAfter modifiedBefore
Validation mode	Specifies whether to validate the XML schema.	No	`None` (default, no validation) `xsd` (validate using XSD) `dtd` (validate using DTD).	validationMode
Namespaces	Whether to enable namespace when parsing the XML files.	No	`true` (default) or `false`	namespaces

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY
Namespace prefix pairs	Namespace URI to prefix mapping, which is used to name fields when parsing the xml file. If an XML file has namespace and namespace is enabled, by default, the field name is the same as it is in the XML document. If there is an item defined for the namespace URI in this map, the field name is `prefix:fieldName`.	No	Array with pattern `['URI1'->'prefix1','URI2'->'prefix2']`	namespacePrefixes
Allow no files found	If true, an error is not thrown if no files are found	no	`true` or `false`	ignoreNoFilesFound

XML source script example

The below script is an example of an XML source configuration in mapping data flows using dataset mode.

```
source(allowSchemaDrift: true,
    validateSchema: false,
    validationMode: 'xsd',
    namespaces: true) ~> XMLSource
```

The below script is an example of an XML source configuration using inline dataset mode.

```
source(allowSchemaDrift: true,
    validateSchema: false,
    format: 'xml',
    fileSystem: 'filesystem',
    folderPath: 'folder',
    validationMode: 'xsd',
    namespaces: true) ~> XMLSource
```

XML connector behavior

Note the following when using XML as source.

- XML attributes:
  - Attributes of an element are parsed as the subfields of the element in the hierarchy.
  - The name of the attribute field follows the pattern `@attributeName`.
- XML schema validation:
  - You can choose to not validate schema, or validate schema using XSD or DTD.
  - When using XSD or DTD to validate XML files, the XSD/DTD must be referred inside the XML files through relative path.

- Namespace handling:
  - Namespace can be disabled when using data flow, in which case the attributes that defines the namespace will be parsed as normal attributes.
  - When namespace is enabled, the names of the element and attributes follow the pattern `namespaceUri,elementName` and `namespaceUri,@attributeName` by default. You can define namespace prefix for each namespace URI in source, in which case the names of the element and attributes follow the pattern `definedPrefix:elementName` or `definedPrefix:@attributeName` instead.
- Value column:
  - If an XML element has both simple text value and attributes/child elements, the simple text value is parsed as the value of a "value column" with built-in field name `_value_`. And it inherits the namespace of the element as well if applies.

Next steps

- Copy activity overview
- Mapping data flow
- Lookup activity
- GetMetadata activity

Transform data in Zendesk (Preview) using Azure Data Factory or Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use Data Flow to transform data in Zendesk (Preview). To learn more, read the introductory article for Azure Data Factory or Azure Synapse Analytics.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Zendesk connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Mapping data flow (source/-)	①

① Azure integration runtime ② Self-hosted integration runtime

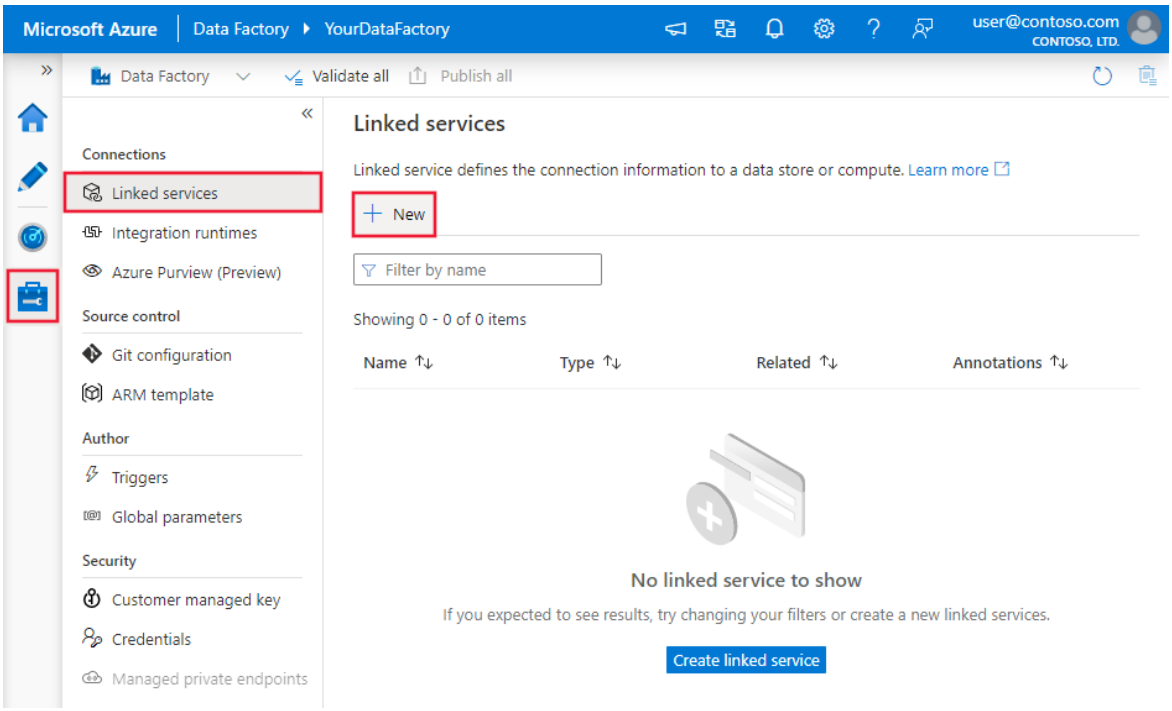
For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

Create a Zendesk linked service using UI

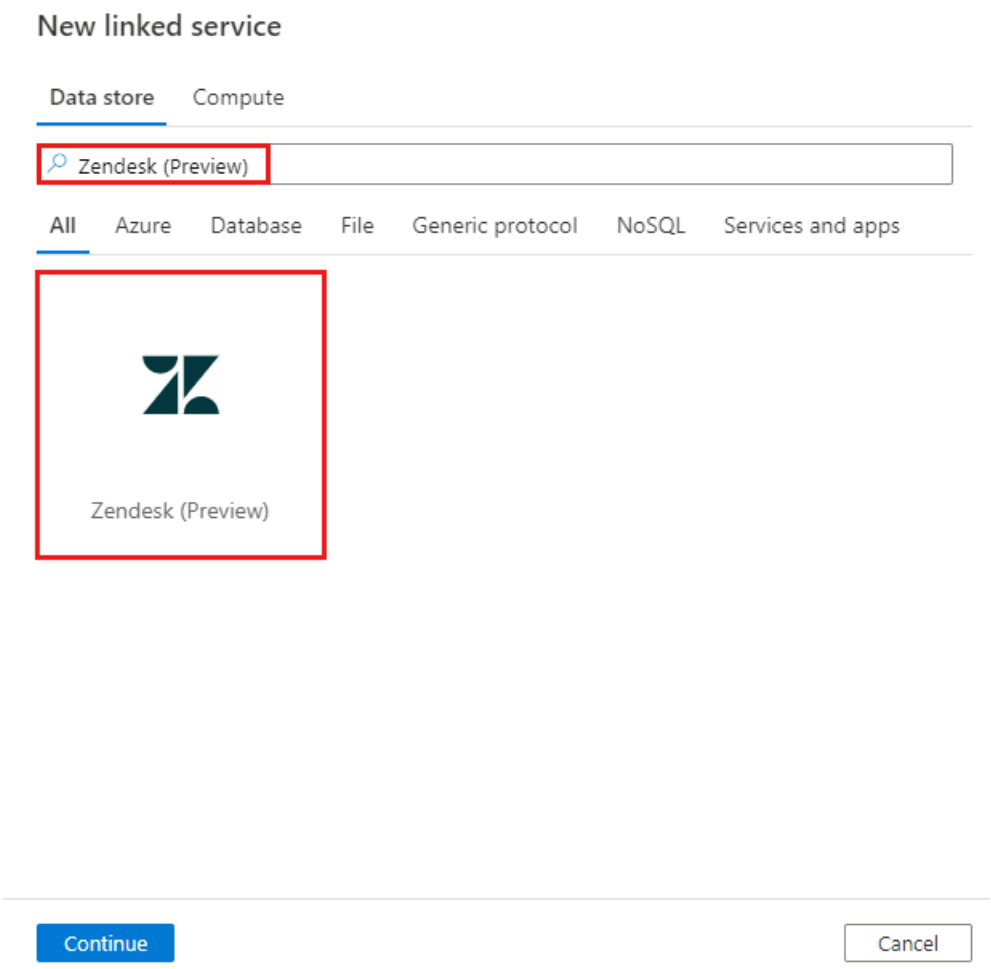
Use the following steps to create a Zendesk linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then select New:

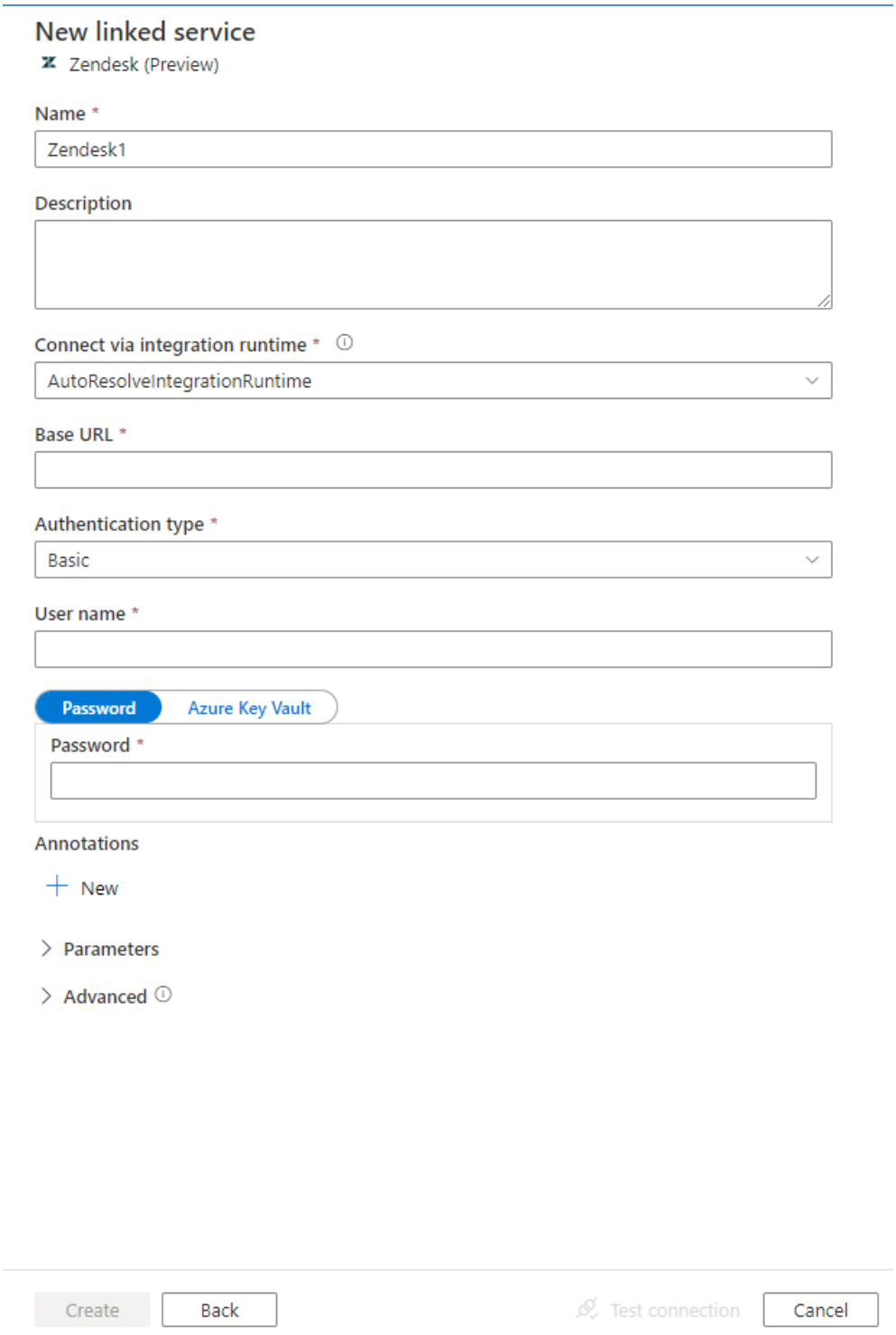
   - Azure Data Factory
   - Azure Synapse



2. Search for Zendesk (Preview) and select the Zendesk (Preview) connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide information about properties that are used to define Data Factory and Synapse pipeline entities specific to Zendesk.

Linked service properties

The following properties are supported for the Zendesk linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to **Zendesk**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
url	The base URL of your Zendesk service.	Yes
authenticationType	Type of authentication used to connect to the Zendesk service. Allowed values are **basic** and **token**. Refer to corresponding sections below on more properties and examples respectively.	Yes

Basic authentication

Set the **authenticationType** property to **basic**. In addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
userName	The user name used to log in to Zendesk.	Yes
password	Specify a password for the user account you specified for the user name. Mark this field as **SecureString** to store it securely. Or, you can reference a secret stored in Azure Key Vault.	Yes

Example:

```
{
    "name": "ZendeskLinkedService",
    "properties": {
        "type": "Zendesk",
        "typeProperties": {
            "url": "<base url>",
            "authenticationType": "basic",
            "userName": "<user name>",
            "password": {
                "type": "SecureString",
                "value": "<password>"
            }
        }
    }
}
```

Token authentication

Set the **authenticationType** property to **token**. In addition to the generic properties that are described in the preceding section, specify the following properties:

PROPERTY	DESCRIPTION	REQUIRED
apiToken	Specify an API token for the Zendesk. Mark this field as **SecureString** to store it securely. Or, you can reference a secret stored in Azure Key Vault.	Yes

Example:

```
{
    "name": "ZendeskLinkedService",
    "properties": {
        "type": "Zendesk",
        "typeProperties": {
            "url": "<base url>",
            "authenticationType": "token",
            "apiToken": {
                "type": "SecureString",
                "value": "<API token>"
            }
        }
    }
}
```

Mapping data flow properties

When transforming data in mapping data flow, you can read tables from Zendesk. For more information, see the source transformation in mapping data flows. You can only use an inline dataset as source type.

Source transformation

The below table lists the properties supported by Zendesk source. You can edit these properties in the **Source options** tab.

NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES	DATA FLOW SCRIPT PROPERTY

Entity	The logical name of the entity in Zendesk.	Yes when use inline mode	`activities`
			`group_memberships`
			`groups`
			`organizations`
			`requests`
			`satisfaction_ratings`
			`sessions`
			`tags`
			`targets`
			`ticket_audits`
			`ticket_fields`
			`ticket_metrics`
			`tickets`
			`triggers`
			`users`
			`views`
				entity

Zendesk source script examples

```
source(allowSchemaDrift: true,
 validateSchema: false,
 store: 'zendesk',
 format: 'rest',
 entity: 'tickets') ~> ZendeskSource
```

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see Supported data stores.

Copy data from Zoho using Azure Data Factory or Synapse Analytics (Preview)

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy Activity in an Azure Data Factory or Synapse Analytics pipeline to copy data from Zoho. It builds on the copy activity overview article that presents a general overview of copy activity.

> **IMPORTANT**
> This connector is currently in preview. You can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Supported capabilities

This Zoho connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR
Copy activity (source/-)	① ②
Lookup activity	① ②

① Azure integration runtime ② Self-hosted integration runtime

For a list of data stores that are supported as sources/sinks, see the Supported data stores table.

This connector supports Xero access token authentication and OAuth 2.0 authentication.

The service provides a built-in driver to enable connectivity, therefore you don't need to manually install any driver using this connector.

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

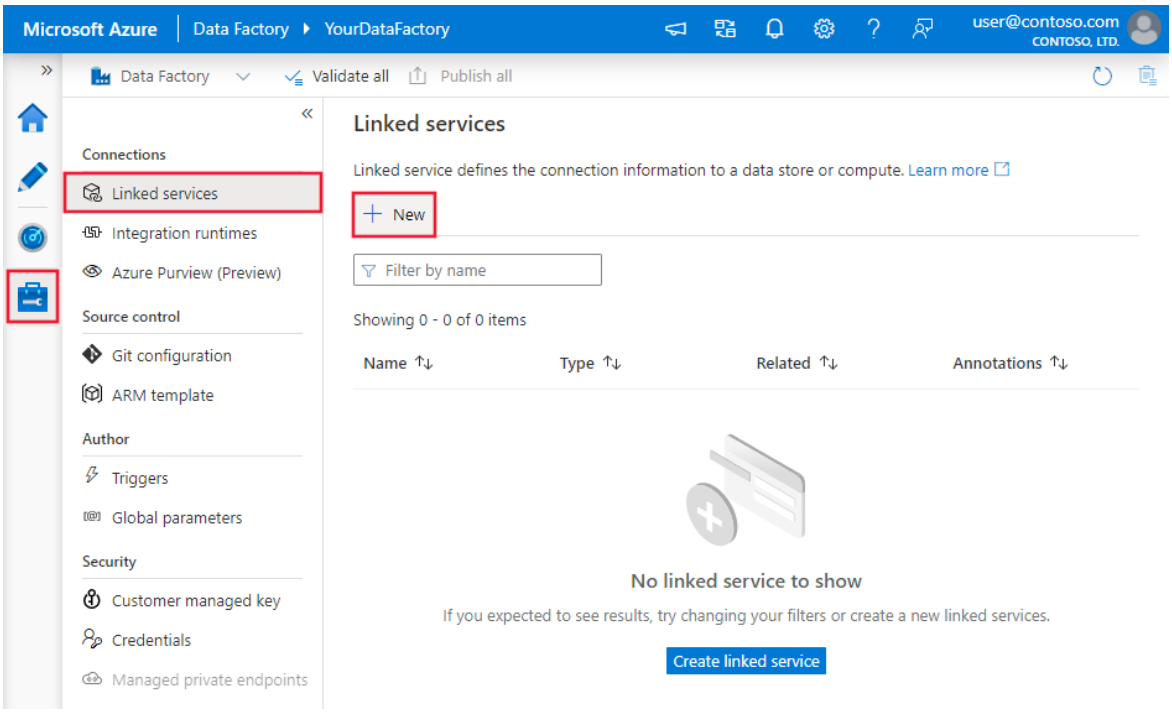
Create a linked service to Zoho using UI

Use the following steps to create a linked service to Zoho in the Azure portal UI.

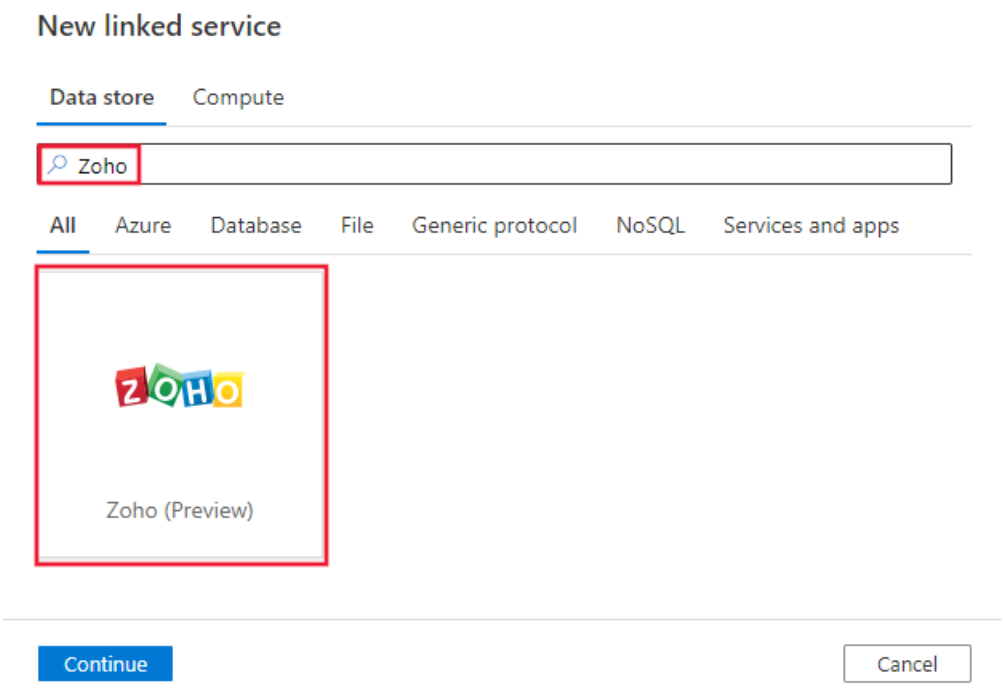
1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services,

then click New:

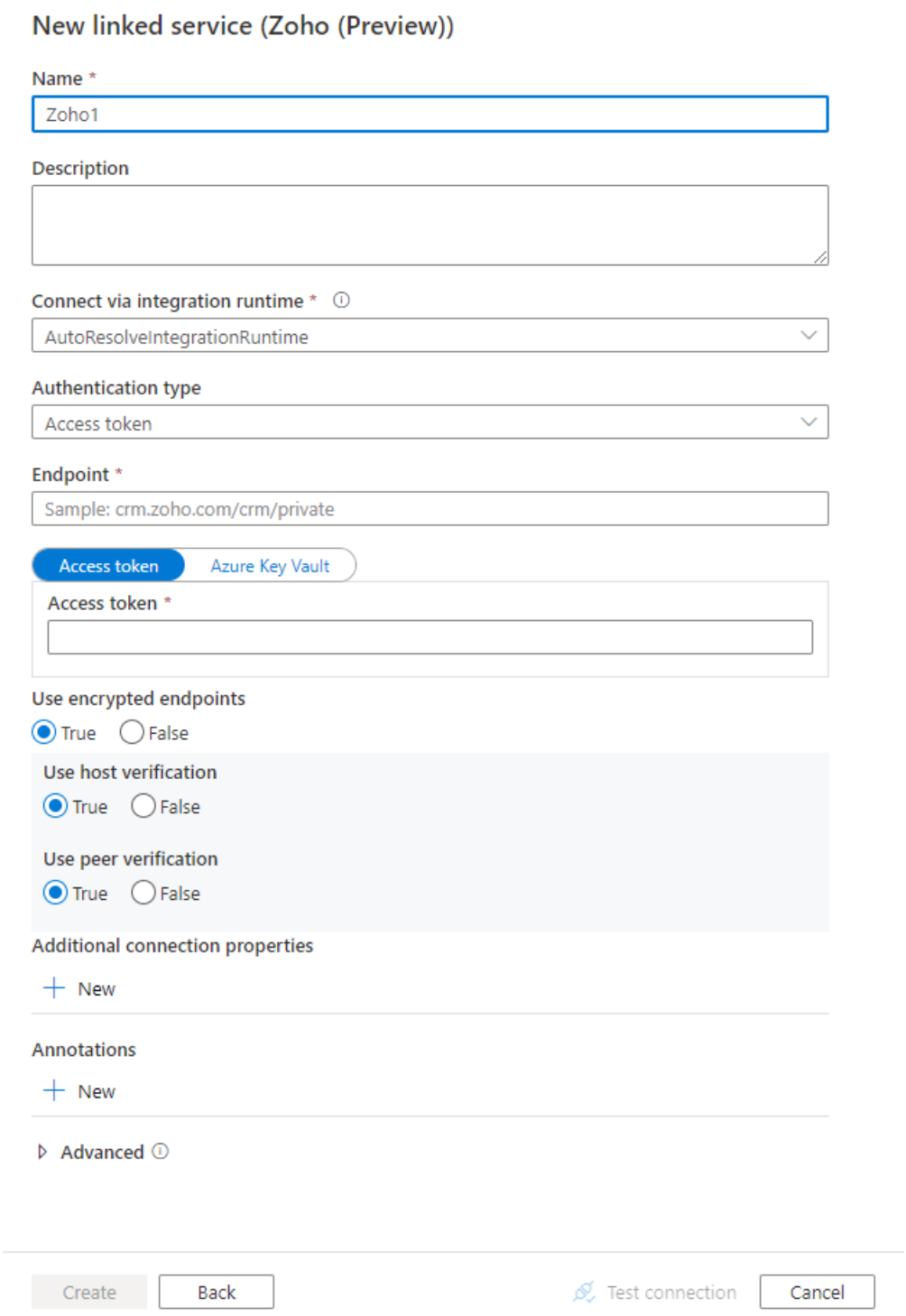
- Azure Data Factory
- Azure Synapse



2. Search for Zoho and select the Zoho connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Zoho connector.

Linked service properties

The following properties are supported for Zoho linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **Zoho**	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectionProperties	A group of properties that defines how to connect to Zoho.	Yes
Under `connectionProperties`*:*		
endpoint	The endpoint of the Zoho server (`crm.zoho.com/crm/private`).	Yes
authenticationType	Allowed values are `OAuth_2.0` and `Access Token` .	Yes
clientId	The client ID associated with your Zoho application.	Yes for OAuth 2.0 authentication
clientSecrect	The clientsecret associated with your Zoho application. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes for OAuth 2.0 authentication
refreshToken	The OAuth 2.0 refresh token associated with your Zoho application, used to refresh the access token when it expires. Refresh token will never expire. To get a refresh token, you must request the `offline` access_type, learn more from this article. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes for OAuth 2.0 authentication
accessToken	The access token for Zoho authentication. Mark this field as a SecureString to store it securely, or reference a secret stored in Azure Key Vault.	Yes
useEncryptedEndpoints	Specifies whether the data source endpoints are encrypted using HTTPS. The default value is true.	No
useHostVerification	Specifies whether to require the host name in the server's certificate to match the host name of the server when connecting over TLS. The default value is true.	No
usePeerVerification	Specifies whether to verify the identity of the server when connecting over TLS. The default value is true.	No

Example: OAuth 2.0 authentication

```
{
    "name": "ZohoLinkedService",
    "properties": {
        "type": "Zoho",
        "typeProperties": {
            "connectionProperties": {
                "authenticationType":"OAuth_2.0",
                "endpoint":"crm.zoho.com/crm/private",
                "clientId":"<client ID>",
                "clientSecrect":{
                    "type": "SecureString",
                    "value": "<client secret>"
                },
                "accessToken":{
                    "type": "SecureString",
                    "value": "<access token>"
                },
                "refreshToken":{
                    "type": "SecureString",
                    "value": "<refresh token>"
                },
                "useEncryptedEndpoints":true,
                "useHostVerification":true,
                "usePeerVerification":true
}
        }
    }
}
```

Example: access token authentication

```
{
    "name": "ZohoLinkedService",
    "properties": {
        "type": "Zoho",
        "typeProperties": {
            "connectionProperties": {
                "authenticationType":"Access Token",
                "endpoint":"crm.zoho.com/crm/private",
                "accessToken":{
                    "type": "SecureString",
                    "value": "<access token>"
                },
                "useEncryptedEndpoints":true,
                "useHostVerification":true,
                "usePeerVerification":true
}
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Zoho dataset.

To copy data from Zoho, set the type property of the dataset to **ZohoObject**. The following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to: **ZohoObject**	Yes
tableName	Name of the table.	No (if "query" in activity source is specified)

Example

```
{
    "name": "ZohoDataset",
    "properties": {
        "type": "ZohoObject",
        "typeProperties": {},
        "schema": [],
        "linkedServiceName": {
            "referenceName": "<Zoho linked service name>",
            "type": "LinkedServiceReference"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Zoho source.

Zoho as source

To copy data from Zoho, set the source type in the copy activity to **ZohoSource**. The following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to: **ZohoSource**	Yes
query	Use the custom SQL query to read data. For example: `"SELECT * FROM Accounts"`.	No (if "tableName" in dataset is specified)

Example:

```
"activities":[
    {
        "name": "CopyFromZoho",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Zoho input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "ZohoSource",
                "query": "SELECT * FROM Accounts"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Next steps

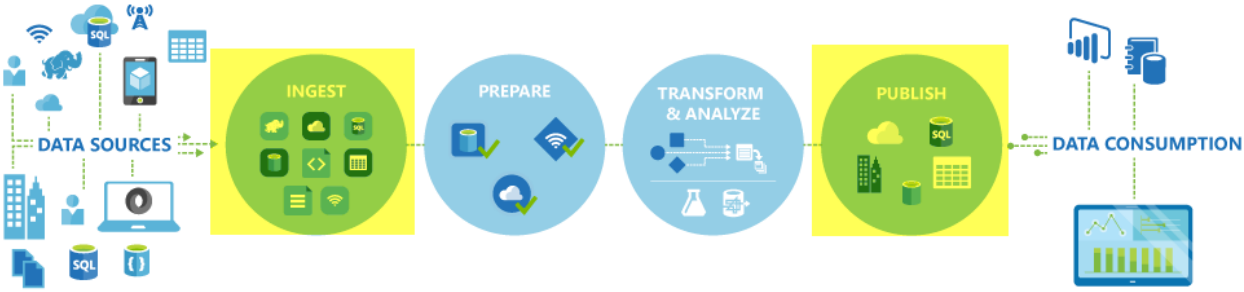
For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Copy activity in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 14 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

In Azure Data Factory and Synapse pipelines, you can use the Copy activity to copy data among data stores located on-premises and in the cloud. After you copy the data, you can use other activities to further transform and analyze it. You can also use the Copy activity to publish transformation and analysis results for business intelligence (BI) and application consumption.



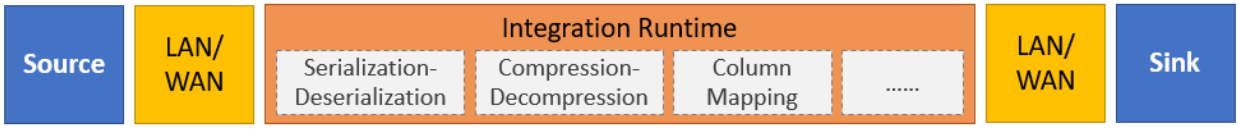
The Copy activity is executed on an integration runtime. You can use different types of integration runtimes for different data copy scenarios:

- When you're copying data between two data stores that are publicly accessible through the internet from any IP, you can use the Azure integration runtime for the copy activity. This integration runtime is secure, reliable, scalable, and globally available.
- When you're copying data to and from data stores that are located on-premises or in a network with access control (for example, an Azure virtual network), you need to set up a self-hosted integration runtime.

An integration runtime needs to be associated with each source and sink data store. For information about how the Copy activity determines which integration runtime to use, see Determining which IR to use.

To copy data from a source to a sink, the service that runs the Copy activity performs these steps:

1. Reads data from a source data store.
2. Performs serialization/deserialization, compression/decompression, column mapping, and so on. It performs these operations based on the configuration of the input dataset, output dataset, and Copy activity.
3. Writes data to the sink/destination data store.



Supported data stores and formats

CATEGORY	DATA STORE	SUPPORTED AS A SOURCE	SUPPORTED AS A SINK	SUPPORTED BY AZURE IR	SUPPORTED BY SELF-HOSTED IR
Azure	Azure Blob storage	✓	✓	✓	✓

CATEGORY	DATA STORE	SUPPORTED AS A SOURCE	SUPPORTED AS A SINK	SUPPORTED BY AZURE IR	SUPPORTED BY SELF-HOSTED IR
	Azure Cognitive Search index		✓	✓	✓
	Azure Cosmos DB (SQL API)	✓	✓	✓	✓
	Azure Cosmos DB's API for MongoDB	✓	✓	✓	✓
	Azure Data Explorer	✓	✓	✓	✓
	Azure Data Lake Storage Gen1	✓	✓	✓	✓
	Azure Data Lake Storage Gen2	✓	✓	✓	✓
	Azure Database for MariaDB	✓		✓	✓
	Azure Database for MySQL	✓	✓	✓	✓
	Azure Database for PostgreSQL	✓	✓	✓	✓
	Azure Databricks Delta Lake	✓	✓	✓	✓
	Azure Files	✓	✓	✓	✓
	Azure SQL Database	✓	✓	✓	✓
	Azure SQL Managed Instance	✓	✓	✓	✓
	Azure Synapse Analytics	✓	✓	✓	✓
	Azure Table storage	✓	✓	✓	✓
Database	Amazon RDS for Oracle	✓		✓	✓
	Amazon RDS for SQL Server	✓		✓	✓
	Amazon Redshift	✓		✓	✓

CATEGORY	DATA STORE	SUPPORTED AS A SOURCE	SUPPORTED AS A SINK	SUPPORTED BY AZURE IR	SUPPORTED BY SELF-HOSTED IR
	DB2	✓		✓	✓
	Drill	✓		✓	✓
	Google BigQuery	✓		✓	✓
	Greenplum	✓		✓	✓
	HBase	✓		✓	✓
	Hive	✓		✓	✓
	Apache Impala	✓		✓	✓
	Informix	✓	✓		✓
	MariaDB	✓		✓	✓
	Microsoft Access	✓	✓		✓
	MySQL	✓		✓	✓
	Netezza	✓		✓	✓
	Oracle	✓	✓	✓	✓
	Phoenix	✓		✓	✓
	PostgreSQL	✓		✓	✓
	Presto	✓		✓	✓
	SAP Business Warehouse via Open Hub	✓			✓
	SAP Business Warehouse via MDX	✓			✓
	SAP HANA	✓	Sink supported only with the ODBC Connector and the SAP HANA ODBC driver		✓
	SAP table	✓			✓
	Snowflake	✓	✓	✓	✓

CATEGORY	DATA STORE	SUPPORTED AS A SOURCE	SUPPORTED AS A SINK	SUPPORTED BY AZURE IR	SUPPORTED BY SELF-HOSTED IR
	Spark	✓		✓	✓
	SQL Server	✓	✓	✓	✓
	Sybase	✓			✓
	Teradata	✓		✓	✓
	Vertica	✓		✓	✓
NoSQL	Cassandra	✓		✓	✓
	Couchbase (Preview)	✓		✓	✓
	MongoDB	✓	✓	✓	✓
	MongoDB Atlas	✓	✓	✓	✓
File	Amazon S3	✓		✓	✓
	Amazon S3 Compatible Storage	✓		✓	✓
	File system	✓	✓	✓	✓
	FTP	✓		✓	✓
	Google Cloud Storage	✓		✓	✓
	HDFS	✓		✓	✓
	Oracle Cloud Storage	✓		✓	✓
	SFTP	✓	✓	✓	✓
Generic protocol	Generic HTTP	✓		✓	✓
	Generic OData	✓		✓	✓
	Generic ODBC	✓	✓		✓
	Generic REST	✓	✓	✓	✓
Services and apps	Amazon Marketplace Web Service	✓		✓	✓

CATEGORY	DATA STORE	SUPPORTED AS A SOURCE	SUPPORTED AS A SINK	SUPPORTED BY AZURE IR	SUPPORTED BY SELF-HOSTED IR
	Concur (Preview)	✓		✓	✓
	Dataverse	✓	✓	✓	✓
	Dynamics 365	✓	✓	✓	✓
	Dynamics AX	✓		✓	✓
	Dynamics CRM	✓	✓	✓	✓
	Google AdWords	✓		✓	✓
	HubSpot	✓		✓	✓
	Jira	✓		✓	✓
	Magento (Preview)	✓		✓	✓
	Marketo (Preview)	✓		✓	✓
	Microsoft 365	✓		✓	✓
	Oracle Eloqua (Preview)	✓		✓	✓
	Oracle Responsys (Preview)	✓		✓	✓
	Oracle Service Cloud (Preview)	✓		✓	✓
	PayPal (Preview)	✓		✓	✓
	QuickBooks (Preview)	✓		✓	✓
	Salesforce	✓	✓	✓	✓
	Salesforce Service Cloud	✓	✓	✓	✓
	Salesforce Marketing Cloud	✓		✓	✓
	SAP Cloud for Customer (C4C)	✓	✓	✓	✓
	SAP ECC	✓		✓	✓

CATEGORY	DATA STORE	SUPPORTED AS A SOURCE	SUPPORTED AS A SINK	SUPPORTED BY AZURE IR	SUPPORTED BY SELF-HOSTED IR
	ServiceNow	✓		✓	✓
	SharePoint Online List	✓		✓	✓
	Shopify (Preview)	✓		✓	✓
	Square (Preview)	✓		✓	✓
	Web table (HTML table)	✓			✓
	Xero	✓		✓	✓
	Zoho (Preview)	✓		✓	✓

NOTE

If a connector is marked *Preview*, you can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, contact Azure support.

Supported file formats

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

You can use the Copy activity to copy files as-is between two file-based data stores, in which case the data is copied efficiently without any serialization or deserialization. In addition, you can also parse or generate files of a given format, for example, you can perform the following:

- Copy data from a SQL Server database and write to Azure Data Lake Storage Gen2 in Parquet format.
- Copy files in text (CSV) format from an on-premises file system and write to Azure Blob storage in Avro format.
- Copy zipped files from an on-premises file system, decompress them on-the-fly, and write extracted files to Azure Data Lake Storage Gen2.
- Copy data in Gzip compressed-text (CSV) format from Azure Blob storage and write it to Azure SQL Database.
- Many more activities that require serialization/deserialization or compression/decompression.

Supported regions

The service that enables the Copy activity is available globally in the regions and geographies listed in Azure

integration runtime locations. The globally available topology ensures efficient data movement that usually avoids cross-region hops. See Products by region to check the availability of Data Factory, Synapse Workspaces and data movement in a specific region.

Configuration

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

In general, to use the Copy activity in Azure Data Factory or Synapse pipelines, you need to:

1. **Create linked services for the source data store and the sink data store.** You can find the list of supported connectors in the Supported data stores and formats section of this article. Refer to the connector article's "Linked service properties" section for configuration information and supported properties.
2. **Create datasets for the source and sink.** Refer to the "Dataset properties" sections of the source and sink connector articles for configuration information and supported properties.
3. **Create a pipeline with the Copy activity.** The next section provides an example.

Syntax

The following template of a Copy activity contains a complete list of supported properties. Specify the ones that fit your scenario.

```
"activities":[
    {
        "name": "CopyActivityTemplate",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<source dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<sink dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>",
                <properties>
            },
            "sink": {
                "type": "<sink type>"
                <properties>
            },
            "translator":
            {
                "type": "TabularTranslator",
                "columnMappings": "<column mapping>"
            },
            "dataIntegrationUnits": <number>,
            "parallelCopies": <number>,
            "enableStaging": true/false,
            "stagingSettings": {
                <properties>
            },
            "enableSkipIncompatibleRow": true/false,
            "redirectIncompatibleRowSettings": {
                <properties>
            }
        }
    }
]
```

Syntax details

PROPERTY	DESCRIPTION	REQUIRED?
type	For a Copy activity, set to `Copy`	Yes
inputs	Specify the dataset that you created that points to the source data. The Copy activity supports only a single input.	Yes
outputs	Specify the dataset that you created that points to the sink data. The Copy activity supports only a single output.	Yes
typeProperties	Specify properties to configure the Copy activity.	Yes

PROPERTY	DESCRIPTION	REQUIRED?
source	Specify the copy source type and the corresponding properties for retrieving data. For more information, see the "Copy activity properties" section in the connector article listed in Supported data stores and formats.	Yes
sink	Specify the copy sink type and the corresponding properties for writing data. For more information, see the "Copy activity properties" section in the connector article listed in Supported data stores and formats.	Yes
translator	Specify explicit column mappings from source to sink. This property applies when the default copy behavior doesn't meet your needs. For more information, see Schema mapping in copy activity.	No
dataIntegrationUnits	Specify a measure that represents the amount of power that the Azure integration runtime uses for data copy. These units were formerly known as cloud Data Movement Units (DMU). For more information, see Data Integration Units.	No
parallelCopies	Specify the parallelism that you want the Copy activity to use when reading data from the source and writing data to the sink. For more information, see Parallel copy.	No
preserve	Specify whether to preserve metadata/ACLs during data copy. For more information, see Preserve metadata.	No
enableStaging stagingSettings	Specify whether to stage the interim data in Blob storage instead of directly copying data from source to sink. For information about useful scenarios and configuration details, see Staged copy.	No
enableSkipIncompatibleRow redirectIncompatibleRowSettings	Choose how to handle incompatible rows when you copy data from source to sink. For more information, see Fault tolerance.	No

Monitoring

You can monitor the Copy activity run in the Azure Data Factory and Synapse pipelines both visually and programmatically. For details, see Monitor copy activity.

Incremental copy

Data Factory and Synapse pipelines enable you to incrementally copy delta data from a source data store to a sink data store. For details, see Tutorial: Incrementally copy data.

Performance and tuning

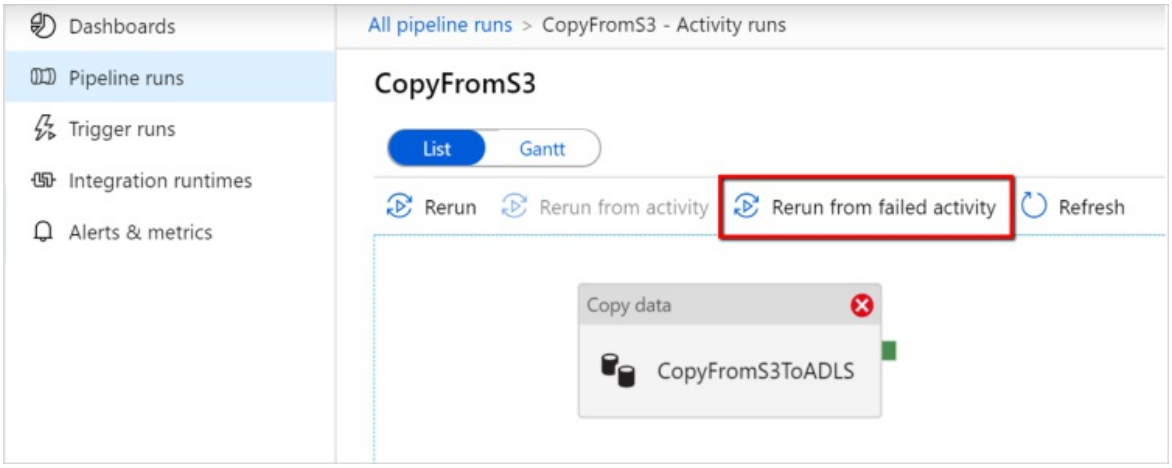
The copy activity monitoring experience shows you the copy performance statistics for each of your activity run. The Copy activity performance and scalability guide describes key factors that affect the performance of data movement via the Copy activity. It also lists the performance values observed during testing and discusses how to optimize the performance of the Copy activity.

Resume from last failed run

Copy activity supports resume from last failed run when you copy large size of files as-is with binary format between file-based stores and choose to preserve the folder/file hierarchy from source to sink, e.g. to migrate data from Amazon S3 to Azure Data Lake Storage Gen2. It applies to the following file-based connectors: Amazon S3, Amazon S3 Compatible Storage Azure Blob, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure Files, File System, FTP, Google Cloud Storage, HDFS, Oracle Cloud Storage and SFTP.

You can leverage the copy activity resume in the following two ways:

- **Activity level retry:** You can set retry count on copy activity. During the pipeline execution, if this copy activity run fails, the next automatic retry will start from last trial's failure point.
- **Rerun from failed activity:** After pipeline execution completion, you can also trigger a rerun from the failed activity in the ADF UI monitoring view or programmatically. If the failed activity is a copy activity, the pipeline will not only rerun from this activity, but also resume from the previous run's failure point.



Few points to note:

- Resume happens at file level. If copy activity fails when copying a file, in next run, this specific file will be re-copied.
- For resume to work properly, do not change the copy activity settings between the reruns.
- When you copy data from Amazon S3, Azure Blob, Azure Data Lake Storage Gen2 and Google Cloud Storage, copy activity can resume from arbitrary number of copied files. While for the rest of file-based connectors as source, currently copy activity supports resume from a limited number of files, usually at the range of tens of thousands and varies depending on the length of the file paths; files beyond this number will be re-copied during reruns.

For other scenarios than binary file copy, copy activity rerun starts from the beginning.

Preserve metadata along with data

While copying data from source to sink, in scenarios like data lake migration, you can also choose to preserve the metadata and ACLs along with data using copy activity. See Preserve metadata for details.

Schema and data type mapping

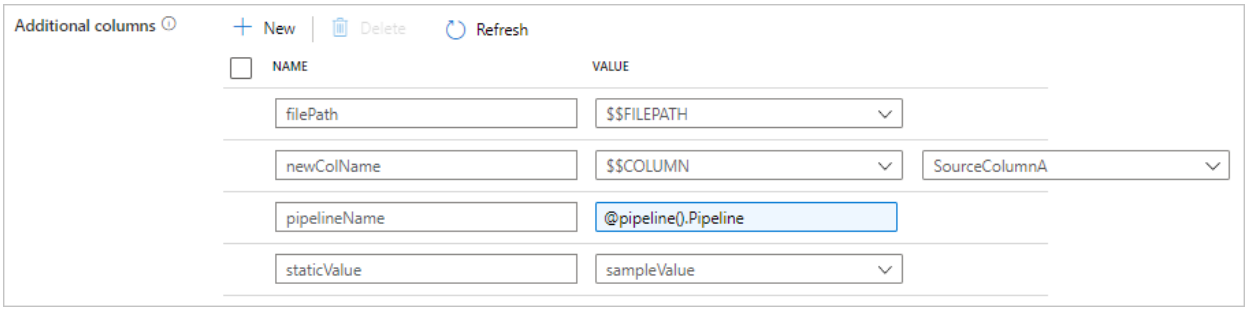
See Schema and data type mapping for information about how the Copy activity maps your source data to your sink.

Add additional columns during copy

In addition to copying data from source data store to sink, you can also configure to add additional data columns to copy along to sink. For example:

- When copy from file-based source, store the relative file path as an additional column to trace from which file the data comes from.
- Duplicate the specified source column as another column.
- Add a column with ADF expression, to attach ADF system variables like pipeline name/pipeline ID, or store other dynamic value from upstream activity's output.
- Add a column with static value to meet your downstream consumption need.

You can find the following configuration on copy activity source tab. You can also map those additional columns in copy activity schema mapping as usual by using your defined column names.



> **TIP**
> This feature works with the latest dataset model. If you don't see this option from the UI, try creating a new dataset.

To configure it programmatically, add the `additionalColumns` property in your copy activity source:

PROPERTY	DESCRIPTION	REQUIRED

PROPERTY	DESCRIPTION	REQUIRED
additionalColumns	Add additional data columns to copy to sink. Each object under the `additionalColumns` array represents an extra column. The `name` defines the column name, and the `value` indicates the data value of that column. Allowed data values are: - `$$FILEPATH` - a reserved variable indicates to store the source files' relative path to the folder path specified in dataset. Apply to file-based source. - `$$COLUMN:<source_column_name>` - a reserved variable pattern indicates to duplicate the specified source column as another column - **Expression** - **Static value**	No

Example:

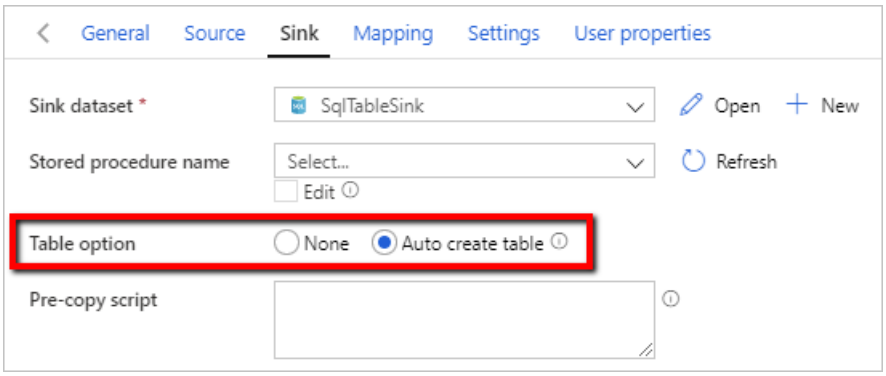
```
"activities":[
    {
        "name": "CopyWithAdditionalColumns",
        "type": "Copy",
        "inputs": [...],
        "outputs": [...],
        "typeProperties": {
            "source": {
                "type": "<source type>",
                "additionalColumns": [
                    {
                        "name": "filePath",
                        "value": "$$FILEPATH"
                    },
                    {
                        "name": "newColName",
                        "value": "$$COLUMN:SourceColumnA"
                    },
                    {
                        "name": "pipelineName",
                        "value": {
                            "value": "@pipeline().Pipeline",
                            "type": "Expression"
                        }
                    },
                    {
                        "name": "staticValue",
                        "value": "sampleValue"
                    }
                ],
                ...
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Auto create sink tables

When copying data into SQL database/Azure Synapse Analytics, if the destination table does not exist, copy activity supports automatically creating it based on the source data. It aims to help you quickly get started to load the data and evaluate SQL database/Azure Synapse Analytics. After the data ingestion, you can review and adjust the sink table schema according to your needs.

This feature is supported when copying data from any source into the following sink data stores. You can find the option on *ADF authoring UI –> Copy activity sink –> Table option –> Auto create table*, or via `tableOption` property in copy activity sink payload.

- Azure SQL Database
- Azure SQL Database Managed Instance
- Azure Synapse Analytics
- SQL Server



Fault tolerance

By default, the Copy activity stops copying data and returns a failure when source data rows are incompatible with sink data rows. To make the copy succeed, you can configure the Copy activity to skip and log the incompatible rows and copy only the compatible data. See Copy activity fault tolerance for details.

Data consistency verification

When you move data from source to destination store, copy activity provides an option for you to do additional data consistency verification to ensure the data is not only successfully copied from source to destination store, but also verified to be consistent between source and destination store. Once inconsistent files have been found during the data movement, you can either abort the copy activity or continue to copy the rest by enabling fault tolerance setting to skip inconsistent files. You can get the skipped file names by enabling session log setting in copy activity. See Data consistency verification in copy activity for details.

Session log

You can log your copied file names, which can help you to further ensure the data is not only successfully copied from source to destination store, but also consistent between source and destination store by reviewing the copy activity session logs. See Session log in copy activity for details.

Next steps

See the following quickstarts, tutorials, and samples:

- Copy data from one location to another location in the same Azure Blob storage account
- Copy data from Azure Blob storage to Azure SQL Database
- Copy data from a SQL Server database to Azure

Monitor copy activity

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

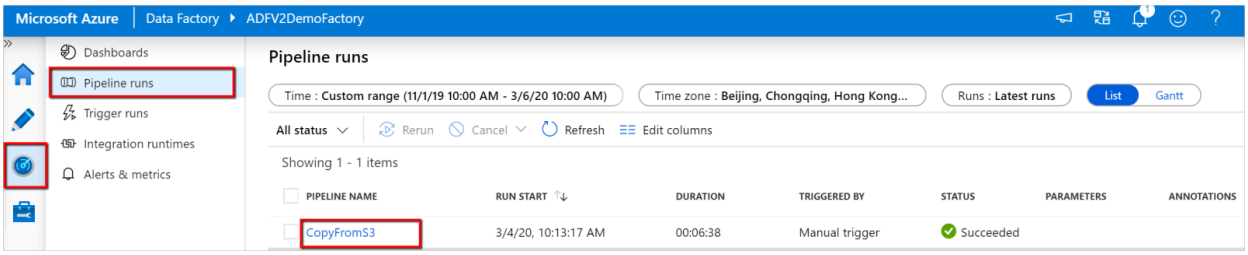
This article outlines how to monitor the copy activity execution in Azure Data Factory and Synapse pipelines. It builds on the copy activity overview article that presents a general overview of copy activity. You can also monitor copy activities generated with the Copy Data Tool as well as Delete activities using the same approach.

Monitor visually

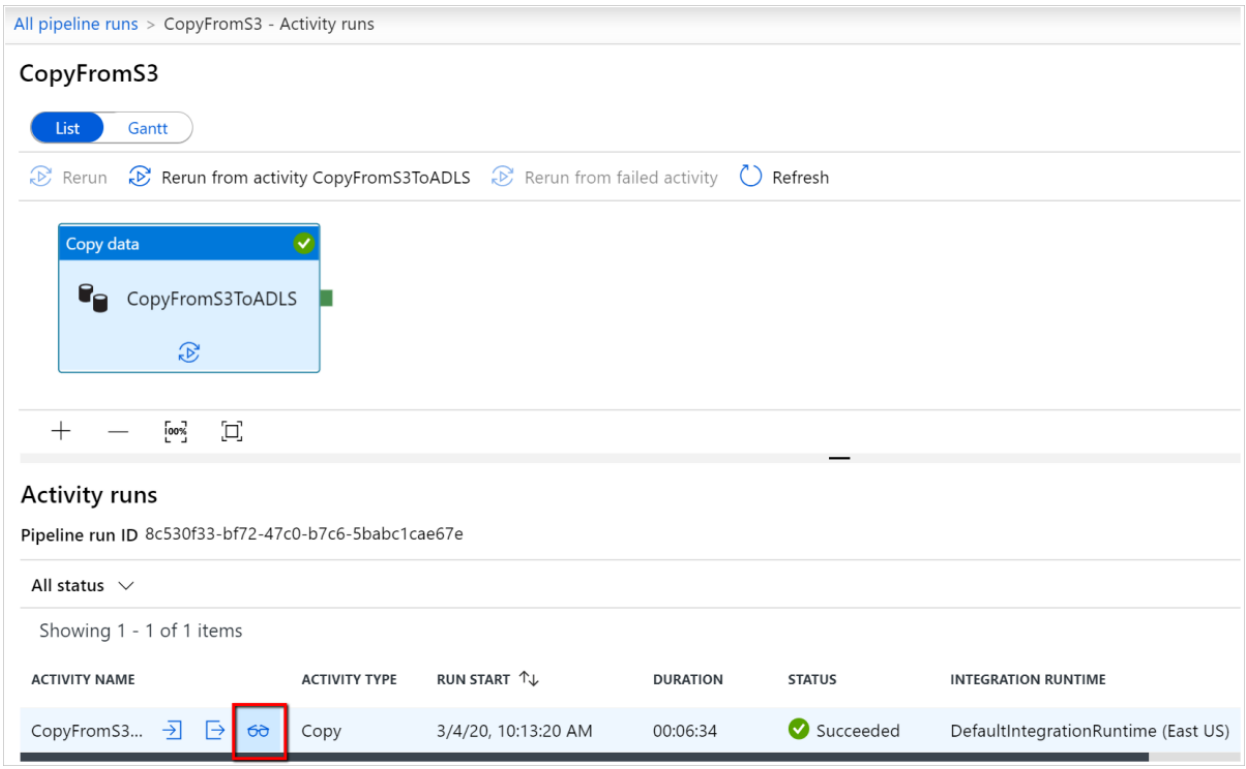
Once you've created and published a pipeline, you can associate it with a trigger or manually kick off an ad hoc run. You can monitor all of your pipeline runs natively in the user experience. Learn about monitoring in general from Visually monitor Azure Data Factory and Synapse pipelines.

To monitor the Copy activity run, go to the **Data Factory Studio** or **Azure Synapse Studio** UI for your service instance. On the **Monitor** tab, you see a list of pipeline runs, click the **pipeline name** link to access the list of activity runs in the pipeline run.

- Azure Data Factory
- Azure Synapse



At this level, you can see links to copy activity input, output, and errors (if the Copy activity run fails), as well as statistics like duration/status. Clicking the **Details** button (eyeglasses) next to the copy activity name will give you deep details on your copy activity execution.

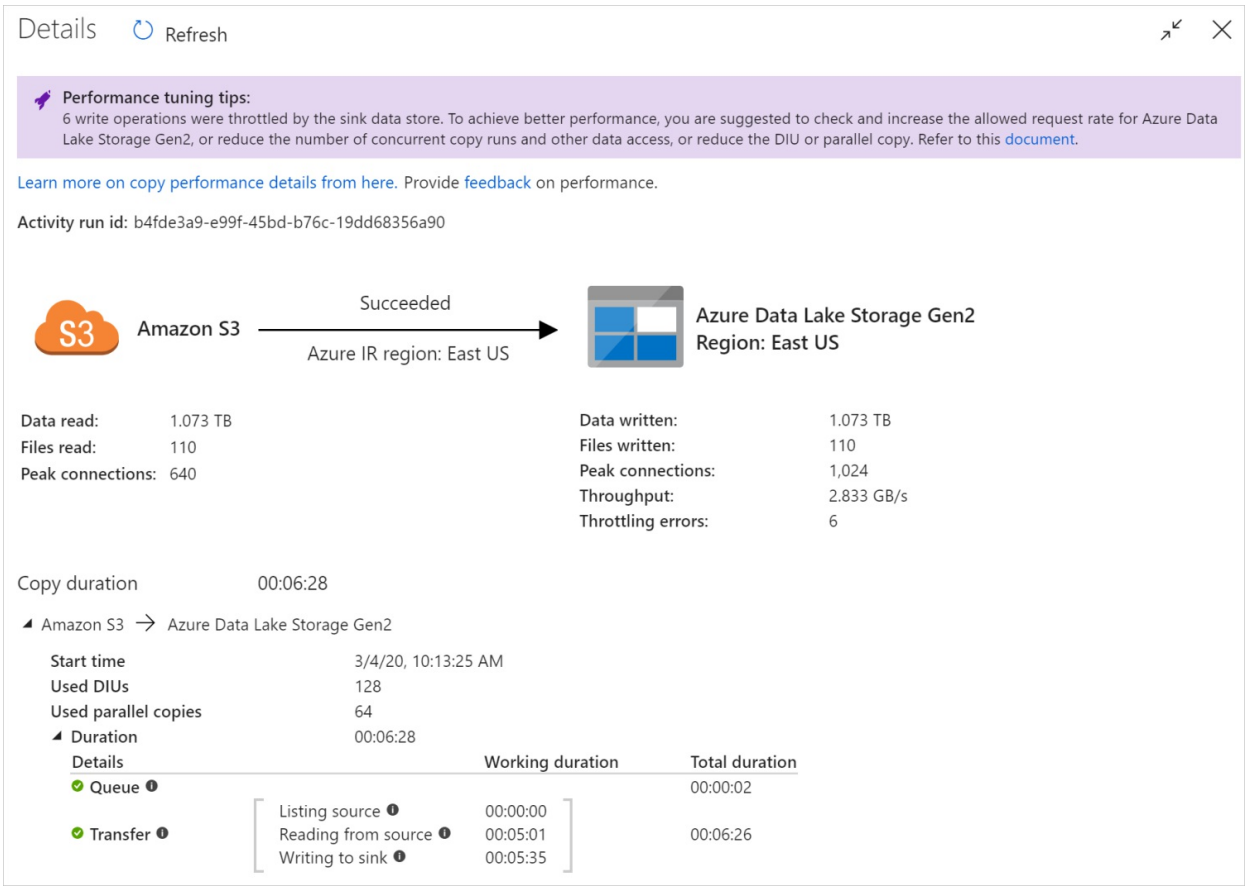


In this graphical monitoring view, the service presents you the copy activity execution information, including data read/written volume, number of files/rows of data copied from source to sink, throughput, the configurations applied for your copy scenario, steps the copy activity goes through with corresponding durations and details, and more. Refer to this table on each possible metric and its detailed description.

In some scenarios, when you run a Copy activity, you'll see **"Performance tuning tips"** at the top of the copy activity monitoring view as shown in the example. The tips tell you the bottleneck identified by the service for the specific copy run, along with suggestion on what to change to boost copy throughput. Learn more about auto performance tuning tips.

The bottom **execution details and durations** describes the key steps your copy activity goes through, which is especially useful for troubleshooting the copy performance. The bottleneck of your copy run is the one with the longest duration. Refer to Troubleshoot copy activity performance on for what each stage represents and the detailed troubleshooting guidance.

Example: Copy from Amazon S3 to Azure Data Lake Storage Gen2



Monitor programmatically

Copy activity execution details and performance characteristics are also returned in the **Copy Activity run result** > **Output** section, which is used to render the UI monitoring view. Following is a complete list of properties that might be returned. You'll see only the properties that are applicable to your copy scenario. For information about how to monitor activity runs programmatically in general, see Programmatically monitor an Azure Data Factory or Synapse pipeline.

PROPERTY NAME	DESCRIPTION	UNIT IN OUTPUT
dataRead	The actual amount of data read from the source.	Int64 value, in bytes
dataWritten	The actual mount of data written/committed to the sink. The size may be different from `dataRead` size, as it relates how each data store stores the data.	Int64 value, in bytes
filesRead	The number of files read from the file-based source.	Int64 value (no unit)
filesWritten	The number of files written/committed to the file-based sink.	Int64 value (no unit)
filesSkipped	The number of files skipped from the file-based source.	Int64 value (no unit)

PROPERTY NAME	DESCRIPTION	UNIT IN OUTPUT
dataConsistencyVerification	Details of data consistency verification where you can see if your copied data has been verified to be consistent between source and destination store. Learn more from this article.	Array
sourcePeakConnections	Peak number of concurrent connections established to the source data store during the Copy activity run.	Int64 value (no unit)
sinkPeakConnections	Peak number of concurrent connections established to the sink data store during the Copy activity run.	Int64 value (no unit)
rowsRead	Number of rows read from the source. This metric does not apply when copying files as-is without parsing them, for example, when source and sink datasets are binary format type, or other format type with identical settings.	Int64 value (no unit)
rowsCopied	Number of rows copied to sink. This metric does not apply when copying files as-is without parsing them, for example, when source and sink datasets are binary format type, or other format type with identical settings.	Int64 value (no unit)
rowsSkipped	Number of incompatible rows that were skipped. You can enable incompatible rows to be skipped by setting `enableSkipIncompatibleRow` to true.	Int64 value (no unit)
copyDuration	Duration of the copy run.	Int32 value, in seconds
throughput	Rate of data transfer, calculated by `dataRead` divided by `copyDuration`.	Floating point number, in KBps
sourcePeakConnections	Peak number of concurrent connections established to the source data store during the Copy activity run.	Int32 value (no unit)
sinkPeakConnections	Peak number of concurrent connections established to the sink data store during the Copy activity run.	Int32 value (no unit)
sqlDwPolyBase	Whether PolyBase is used when data is copied into Azure Synapse Analytics.	Boolean

PROPERTY NAME	DESCRIPTION	UNIT IN OUTPUT
redshiftUnload	Whether UNLOAD is used when data is copied from Redshift.	Boolean
hdfsDistcp	Whether DistCp is used when data is copied from HDFS.	Boolean
effectiveIntegrationRuntime	The integration runtime (IR) or runtimes used to power the activity run, in the format `<IR name> (<region if it's Azure IR>)`.	Text (string)
usedDataIntegrationUnits	The effective Data Integration Units during copy.	Int32 value
usedParallelCopies	The effective parallelCopies during copy.	Int32 value
logPath	Path to the session log of skipped data in the blob storage. See Fault tolerance.	Text (string)
executionDetails	More details on the stages the Copy activity goes through and the corresponding steps, durations, configurations, and so on. We don't recommend that you parse this section because it might change. To better understand how it helps you understand and troubleshoot copy performance, refer to Monitor visually section.	Array
perfRecommendation	Copy performance tuning tips. See Performance tuning tips for details.	Array
billingReference	The billing consumption for the given run. Learn more from Monitor consumption at activity-run level.	Object
durationInQueue	Queueing duration in second before the copy activity starts to execute.	Object

Example:

```
"output": {
    "dataRead": 1180089300500,
    "dataWritten": 1180089300500,
    "filesRead": 110,
    "filesWritten": 110,
    "filesSkipped": 0,
    "sourcePeakConnections": 640,
    "sinkPeakConnections": 1024,
    "copyDuration": 388,
    "throughput": 2970183,
    "errors": [],
    "effectiveIntegrationRuntime": "DefaultIntegrationRuntime (East US)",
```

```
    "usedDataIntegrationUnits": 128,
    "billingReference": "{\"activityType\":\"DataMovement\",\"billableDuration\":
[{\"Managed\":11.733333333333336}]}",
    "usedParallelCopies": 64,
    "dataConsistencyVerification":
    {
        "VerificationResult": "Verified",
        "InconsistentData": "None"
    },
    "executionDetails": [
        {
            "source": {
                "type": "AmazonS3"
            },
            "sink": {
                "type": "AzureBlobFS",
                "region": "East US",
                "throttlingErrors": 6
            },
            "status": "Succeeded",
            "start": "2020-03-04T02:13:25.1454206Z",
            "duration": 388,
            "usedDataIntegrationUnits": 128,
            "usedParallelCopies": 64,
            "profile": {
                "queue": {
                    "status": "Completed",
                    "duration": 2
                },
                "transfer": {
                    "status": "Completed",
                    "duration": 386,
                    "details": {
                        "listingSource": {
                            "type": "AmazonS3",
                            "workingDuration": 0
                        },
                        "readingFromSource": {
                            "type": "AmazonS3",
                            "workingDuration": 301
                        },
                        "writingToSink": {
                            "type": "AzureBlobFS",
                            "workingDuration": 335
                        }
                    }
                }
            },
            "detailedDurations": {
                "queuingDuration": 2,
                "transferDuration": 386
            }
        }
    ],
    "perfRecommendation": [
        {
            "Tip": "6 write operations were throttled by the sink data store. To achieve better performance,
you are suggested to check and increase the allowed request rate for Azure Data Lake Storage Gen2, or reduce
the number of concurrent copy runs and other data access, or reduce the DIU or parallel copy.",
            "ReferUrl": "https://go.microsoft.com/fwlink/?linkid=2102534 ",
            "RuleName": "ReduceThrottlingErrorPerfRecommendationRule"
        }
    ],
    "durationInQueue": {
        "integrationRuntimeQueue": 0
    }
}
```

Next steps

See the other Copy Activity articles:

- Copy activity overview

- Copy activity performance

Delete Activity in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 8 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

You can use the Delete Activity in Azure Data Factory to delete files or folders from on-premises storage stores or cloud storage stores. Use this activity to clean up or archive files when they are no longer needed.

> **WARNING**
> Deleted files or folders cannot be restored (unless the storage has soft-delete enabled). Be cautious when using the Delete activity to delete files or folders.

Best practices

Here are some recommendations for using the Delete activity:

- Back up your files before deleting them with the Delete activity in case you need to restore them in the future.
- Make sure that the service has write permissions to delete folders or files from the storage store.
- Make sure you are not deleting files that are being written at the same time.
- If you want to delete files or folder from an on-premises system, make sure you are using a self-hosted integration runtime with a version greater than 3.14.

Supported data stores

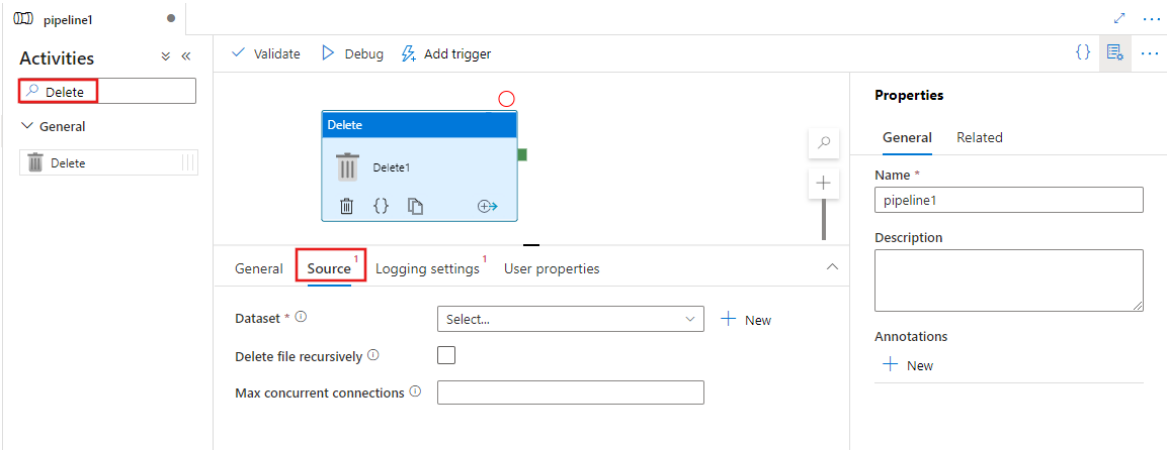
- Azure Blob storage
- Azure Data Lake Storage Gen1
- Azure Data Lake Storage Gen2
- Azure Files
- File System
- FTP
- SFTP
- Amazon S3
- Amazon S3 Compatible Storage
- Google Cloud Storage
- Oracle Cloud Storage
- HDFS

Create a Delete activity with UI

To use a Delete activity in a pipeline, complete the following steps:

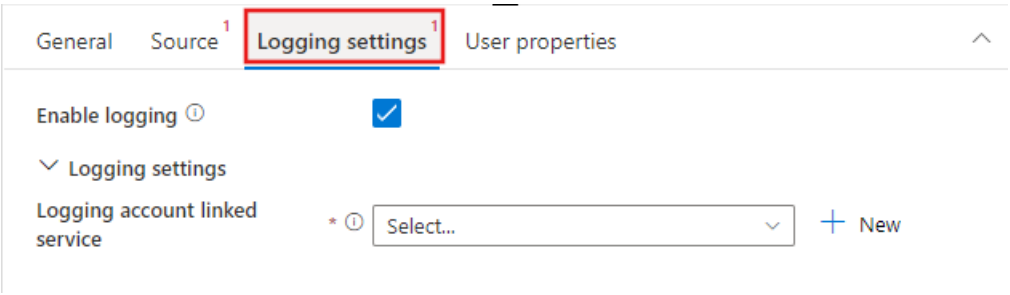
1. Search for *Delete* in the pipeline Activities pane, and drag a Delete activity to the pipeline canvas.
2. Select the new Delete activity on the canvas if it is not already selected, and its **Source** tab, to edit its

details.



3. Select an existing or create a new Dataset specifying the files to be deleted. If multiple files are selected, optionally enable recursive deletion, which deletes data in any child folders as well. You can also specify a maximum number of concurrent connections for the operation.

4. Optionally configure logging by selecting the **Logging settings** tab and selecting an existing or creating a new logging account linked service location to log results of the delete operations performed.



Syntax

```
{
    "name": "DeleteActivity",
    "type": "Delete",
    "typeProperties": {
        "dataset": {
            "referenceName": "<dataset name>",
            "type": "DatasetReference"
        },
        "storeSettings": {
            "type": "<source type>",
            "recursive": true/false,
            "maxConcurrentConnections": <number>
        },
        "enableLogging": true/false,
        "logStorageSettings": {
            "linkedServiceName": {
                "referenceName": "<name of linked service>",
                "type": "LinkedServiceReference"
            },
            "path": "<path to save log file>"
        }
    }
}
```

Type properties

PROPERTY	DESCRIPTION	REQUIRED
dataset	Provides the dataset reference to determine which files or folder to be deleted	Yes
recursive	Indicates whether the files are deleted recursively from the subfolders or only from the specified folder.	No. The default is `false`.
maxConcurrentConnections	The number of the connections to connect to storage store concurrently for deleting folder or files.	No. The default is `1`.
enablelogging	Indicates whether you need to record the folder or file names that have been deleted. If true, you need to further provide a storage account to save the log file, so that you can track the behaviors of the Delete activity by reading the log file.	No
logStorageSettings	Only applicable when enablelogging = true. A group of storage properties that can be specified where you want to save the log file containing the folder or file names that have been deleted by the Delete activity.	No
linkedServiceName	Only applicable when enablelogging = true. The linked service of Azure Storage, Azure Data Lake Storage Gen1, or Azure Data Lake Storage Gen2 to store the log file that contains the folder or file names that have been deleted by the Delete activity. Be aware it must be configured with the same type of Integration Runtime from the one used by delete activity to delete files.	No
path	Only applicable when enablelogging = true. The path to save the log file in your storage account. If you do not provide a path, the service creates a container for you.	No

Monitoring

There are two places where you can see and monitor the results of the Delete activity:

- From the output of the Delete activity.
- From the log file.

Sample output of the Delete activity

```
{
  "datasetName": "AmazonS3",
  "type": "AmazonS3Object",
  "prefix": "test",
  "bucketName": "adf",
  "recursive": true,
  "isWildcardUsed": false,
  "maxConcurrentConnections": 2,
  "filesDeleted": 4,
  "logPath": "https://sample.blob.core.windows.net/mycontainer/5c698705-a6e2-40bf-911e-e0a927de3f07",
  "effectiveIntegrationRuntime": "MyAzureIR (West Central US)",
  "executionDuration": 650
}
```

Sample log file of the Delete activity

NAME	CATEGORY	STATUS	ERROR
test1/yyy.json	File	Deleted	
test2/hello789.txt	File	Deleted	
test2/test3/hello000.txt	File	Deleted	
test2/test3/zzz.json	File	Deleted	

Examples of using the Delete activity

Delete specific folders or files

The store has the following folder structure:

Root/
  Folder_A_1/
    1.txt
    2.txt
    3.csv
  Folder_A_2/
    4.txt
    5.csv
    Folder_B_1/
      6.txt
      7.csv
    Folder_B_2/
      8.txt

Now you are using the Delete activity to delete folder or files by the combination of different property value from the dataset and the Delete activity:

FOLDERPATH	FILENAME	RECURSIVE	OUTPUT

FOLDERPATH	FILENAME	RECURSIVE	OUTPUT
Root/ Folder_A_2	NULL	False	Root/ Folder_A_1/ 1.txt 2.txt 3.csv Folder_A_2/ ~~4.txt~~ ~~5.csv~~ Folder_B_1/ 6.txt 7.csv Folder_B_2/ 8.txt
Root/ Folder_A_2	NULL	True	Root/ Folder_A_1/ 1.txt 2.txt 3.csv ~~Folder_A_2/~~ ~~4.txt~~ ~~5.csv~~ ~~Folder_B_1/~~ ~~6.txt~~ ~~7.csv~~ ~~Folder_B_2/~~ ~~8.txt~~
Root/ Folder_A_2	*.txt	False	Root/ Folder_A_1/ 1.txt 2.txt 3.csv Folder_A_2/ ~~4.txt~~ 5.csv Folder_B_1/ 6.txt 7.csv Folder_B_2/ 8.txt
Root/ Folder_A_2	*.txt	True	Root/ Folder_A_1/ 1.txt 2.txt 3.csv Folder_A_2/ ~~4.txt~~ 5.csv Folder_B_1/ ~~6.txt~~ 7.csv Folder_B_2/ ~~8.txt~~

Periodically clean up the time-partitioned folder or files

You can create a pipeline to periodically clean up the time partitioned folder or files. For example, the folder structure is similar as: `/mycontainer/2018/12/14/*.csv`. You can leverage the service system variable from

schedule trigger to identify which folder or files should be deleted in each pipeline run.

Sample pipeline

```
{
    "name":"cleanup_time_partitioned_folder",
    "properties":{
        "activities":[
            {
                "name":"DeleteOneFolder",
                "type":"Delete",
                "dependsOn":[

                ],
                "policy":{
                    "timeout":"7.00:00:00",
                    "retry":0,
                    "retryIntervalInSeconds":30,
                    "secureOutput":false,
                    "secureInput":false
                },
                "userProperties":[

                ],
                "typeProperties":{
                    "dataset":{
                        "referenceName":"PartitionedFolder",
                        "type":"DatasetReference",
                        "parameters":{
                            "TriggerTime":{
                                "value":"@formatDateTime(pipeline().parameters.TriggerTime, 'yyyy/MM/dd')",
                                "type":"Expression"
                            }
                        }
                    },
                    "logStorageSettings":{
                        "linkedServiceName":{
                            "referenceName":"BloblinkedService",
                            "type":"LinkedServiceReference"
                        },
                        "path":"mycontainer/log"
                    },
                    "enableLogging":true,
                    "storeSettings":{
                        "type":"AzureBlobStorageReadSettings",
                        "recursive":true
                    }
                }
            }
        ],
        "parameters":{
            "TriggerTime":{
                "type":"string"
            }
        },
        "annotations":[

        ]
    }
}
```

Sample dataset

```
{
    "name":"PartitionedFolder",
    "properties":{
        "linkedServiceName":{
            "referenceName":"BloblinkedService",
            "type":"LinkedServiceReference"
        },
        "parameters":{
            "TriggerTime":{
                "type":"string"
            }
        },
        "annotations":[

        ],
        "type":"Binary",
        "typeProperties":{
            "location":{
                "type":"AzureBlobStorageLocation",
                "folderPath":{
                    "value":"@dataset().TriggerTime",
                    "type":"Expression"
                },
                "container":{
                    "value":"mycontainer",
                    "type":"Expression"
                }
            }
        }
    }
}
```

Sample trigger

```
{
    "name": "DailyTrigger",
    "properties": {
        "runtimeState": "Started",
        "pipelines": [
            {
                "pipelineReference": {
                    "referenceName": "cleanup_time_partitioned_folder",
                    "type": "PipelineReference"
                },
                "parameters": {
                    "TriggerTime": "@trigger().scheduledTime"
                }
            }
        ],
        "type": "ScheduleTrigger",
        "typeProperties": {
            "recurrence": {
                "frequency": "Day",
                "interval": 1,
                "startTime": "2018-12-13T00:00:00.000Z",
                "timeZone": "UTC",
                "schedule": {
                    "minutes": [
                        59
                    ],
                    "hours": [
                        23
                    ]
                }
            }
        }
    }
}
```

Clean up the expired files that were last modified before 2018.1.1

You can create a pipeline to clean up the old or expired files by leveraging file attribute filter: "LastModified" in dataset.

Sample pipeline

```
{
    "name":"CleanupExpiredFiles",
    "properties":{
        "activities":[
            {
                "name":"DeleteFilebyLastModified",
                "type":"Delete",
                "dependsOn":[

                ],
                "policy":{
                    "timeout":"7.00:00:00",
                    "retry":0,
                    "retryIntervalInSeconds":30,
                    "secureOutput":false,
                    "secureInput":false
                },
                "userProperties":[

                ],
                "typeProperties":{
                    "dataset":{
                        "referenceName":"BlobFilesLastModifiedBefore201811",
                        "type":"DatasetReference"
                    },
                    "logStorageSettings":{
                        "linkedServiceName":{
                            "referenceName":"BloblinkedService",
                            "type":"LinkedServiceReference"
                        },
                        "path":"mycontainer/log"
                    },
                    "enableLogging":true,
                    "storeSettings":{
                        "type":"AzureBlobStorageReadSettings",
                        "recursive":true,
                        "modifiedDatetimeEnd":"2018-01-01T00:00:00.000Z"
                    }
                }
            }
        ],
        "annotations":[

        ]
    }
}
```

Sample dataset

```
{
    "name":"BlobFilesLastModifiedBefore201811",
    "properties":{
        "linkedServiceName":{
            "referenceName":"BloblinkedService",
            "type":"LinkedServiceReference"
        },
        "annotations":[

        ],
        "type":"Binary",
        "typeProperties":{
            "location":{
                "type":"AzureBlobStorageLocation",
                "fileName":"*",
                "folderPath":"mydirectory",
                "container":"mycontainer"
            }
        }
    }
}
```

Move files by chaining the Copy activity and the Delete activity

You can move a file by using a Copy activity to copy a file and then a Delete activity to delete a file in a pipeline. When you want to move multiple files, you can use the GetMetadata activity + Filter activity + Foreach activity + Copy activity + Delete activity as in the following sample.

> **NOTE**
>
> If you want to move the entire folder by defining a dataset containing a folder path only, and then using a Copy activity and a Delete activity to reference to the same dataset representing a folder, you need to be very careful. You must ensure that there **will not** be any new files arriving into the folder between the copy operation and the delete operation. If new files arrive in the folder at the moment when your copy activity just completed the copy job but the Delete activity has not been started, then the Delete activity might delete the newly arriving file which has NOT been copied to the destination yet by deleting the entire folder.

Sample pipeline

```
{
    "name":"MoveFiles",
    "properties":{
        "activities":[
            {
                "name":"GetFileList",
                "type":"GetMetadata",
                "dependsOn":[

                ],
                "policy":{
                    "timeout":"7.00:00:00",
                    "retry":0,
                    "retryIntervalInSeconds":30,
                    "secureOutput":false,
                    "secureInput":false
                },
                "userProperties":[

                ],
                "typeProperties":{
                    "dataset":{
                        "referenceName":"OneSourceFolder",
                        "type":"DatasetReference",
```

```
                    "parameters":{
                        "Container":{
                            "value":"@pipeline().parameters.SourceStore_Location",
                            "type":"Expression"
                        },
                        "Directory":{
                            "value":"@pipeline().parameters.SourceStore_Directory",
                            "type":"Expression"
                        }
                    }
                },
                "fieldList":[
                    "childItems"
                ],
                "storeSettings":{
                    "type":"AzureBlobStorageReadSettings",
                    "recursive":true
                },
                "formatSettings":{
                    "type":"BinaryReadSettings"
                }
            }
        },
        {
            "name":"FilterFiles",
            "type":"Filter",
            "dependsOn":[
                {
                    "activity":"GetFileList",
                    "dependencyConditions":[
                        "Succeeded"
                    ]
                }
            ],
            "userProperties":[

            ],
            "typeProperties":{
                "items":{
                    "value":"@activity('GetFileList').output.childItems",
                    "type":"Expression"
                },
                "condition":{
                    "value":"@equals(item().type, 'File')",
                    "type":"Expression"
                }
            }
        },
        {
            "name":"ForEachFile",
            "type":"ForEach",
            "dependsOn":[
                {
                    "activity":"FilterFiles",
                    "dependencyConditions":[
                        "Succeeded"
                    ]
                }
            ],
            "userProperties":[

            ],
            "typeProperties":{
                "items":{
                    "value":"@activity('FilterFiles').output.value",
                    "type":"Expression"
                },
                "batchCount":20,
                "activities":[
```

```
                        {
                            "name":"CopyAFile",
                            "type":"Copy",
                            "dependsOn":[

                            ],
                            "policy":{
                                "timeout":"7.00:00:00",
                                "retry":0,
                                "retryIntervalInSeconds":30,
                                "secureOutput":false,
                                "secureInput":false
                            },
                            "userProperties":[

                            ],
                            "typeProperties":{
                                "source":{
                                    "type":"BinarySource",
                                    "storeSettings":{
                                        "type":"AzureBlobStorageReadSettings",
                                        "recursive":false,
                                        "deleteFilesAfterCompletion":false
                                    },
                                    "formatSettings":{
                                        "type":"BinaryReadSettings"
                                    },
                                    "recursive":false
                                },
                                "sink":{
                                    "type":"BinarySink",
                                    "storeSettings":{
                                        "type":"AzureBlobStorageWriteSettings"
                                    }
                                },
                                "enableStaging":false,
                                "dataIntegrationUnits":0
                            },
                            "inputs":[
                                {
                                    "referenceName":"OneSourceFile",
                                    "type":"DatasetReference",
                                    "parameters":{
                                        "Container":{
                                            "value":"@pipeline().parameters.SourceStore_Location",
                                            "type":"Expression"
                                        },
                                        "Directory":{
                                            "value":"@pipeline().parameters.SourceStore_Directory",
                                            "type":"Expression"
                                        },
                                        "filename":{
                                            "value":"@item().name",
                                            "type":"Expression"
                                        }
                                    }
                                }
                            ],
                            "outputs":[
                                {
                                    "referenceName":"OneDestinationFile",
                                    "type":"DatasetReference",
                                    "parameters":{
                                        "Container":{
                                            "value":"@pipeline().parameters.DestinationStore_Location",
                                            "type":"Expression"
                                        },
                                        "Directory":{
                                            "value":"@pipeline().parameters.DestinationStore_Directory",
```

```
                                    "type":"Expression"
                                },
                                "filename":{
                                    "value":"@item().name",
                                    "type":"Expression"
                                }
                            }
                        }
                    ]
                },
                {
                    "name":"DeleteAFile",
                    "type":"Delete",
                    "dependsOn":[
                        {
                            "activity":"CopyAFile",
                            "dependencyConditions":[
                                "Succeeded"
                            ]
                        }
                    ],
                    "policy":{
                        "timeout":"7.00:00:00",
                        "retry":0,
                        "retryIntervalInSeconds":30,
                        "secureOutput":false,
                        "secureInput":false
                    },
                    "userProperties":[

                    ],
                    "typeProperties":{
                        "dataset":{
                            "referenceName":"OneSourceFile",
                            "type":"DatasetReference",
                            "parameters":{
                                "Container":{
                                    "value":"@pipeline().parameters.SourceStore_Location",
                                    "type":"Expression"
                                },
                                "Directory":{
                                    "value":"@pipeline().parameters.SourceStore_Directory",
                                    "type":"Expression"
                                },
                                "filename":{
                                    "value":"@item().name",
                                    "type":"Expression"
                                }
                            }
                        },
                        "logStorageSettings":{
                            "linkedServiceName":{
                                "referenceName":"BloblinkedService",
                                "type":"LinkedServiceReference"
                            },
                            "path":"container/log"
                        },
                        "enableLogging":true,
                        "storeSettings":{
                            "type":"AzureBlobStorageReadSettings",
                            "recursive":true
                        }
                    }
                }
            ]
        }
    }
],
"parameters":{
```

```
            "parameters":{
                "SourceStore_Location":{
                    "type":"String"
                },
                "SourceStore_Directory":{
                    "type":"String"
                },
                "DestinationStore_Location":{
                    "type":"String"
                },
                "DestinationStore_Directory":{
                    "type":"String"
                }
            },
            "annotations":[

            ]
        }
}
```

Sample datasets

Dataset used by GetMetadata activity to enumerate the file list.

```
{
    "name":"OneSourceFolder",
    "properties":{
        "linkedServiceName":{
            "referenceName":"AzureStorageLinkedService",
            "type":"LinkedServiceReference"
        },
        "parameters":{
            "Container":{
                "type":"String"
            },
            "Directory":{
                "type":"String"
            }
        },
        "annotations":[

        ],
        "type":"Binary",
        "typeProperties":{
            "location":{
                "type":"AzureBlobStorageLocation",
                "folderPath":{
                    "value":"@{dataset().Directory}",
                    "type":"Expression"
                },
                "container":{
                    "value":"@{dataset().Container}",
                    "type":"Expression"
                }
            }
        }
    }
}
```

Dataset for data source used by copy activity and the Delete activity.

```
{
    "name":"OneSourceFile",
    "properties":{
        "linkedServiceName":{
            "referenceName":"AzureStorageLinkedService",
            "type":"LinkedServiceReference"
        },
        "parameters":{
            "Container":{
                "type":"String"
            },
            "Directory":{
                "type":"String"
            },
            "filename":{
                "type":"string"
            }
        },
        "annotations":[

        ],
        "type":"Binary",
        "typeProperties":{
            "location":{
                "type":"AzureBlobStorageLocation",
                "fileName":{
                    "value":"@dataset().filename",
                    "type":"Expression"
                },
                "folderPath":{
                    "value":"@{dataset().Directory}",
                    "type":"Expression"
                },
                "container":{
                    "value":"@{dataset().Container}",
                    "type":"Expression"
                }
            }
        }
    }
}
```

Dataset for data destination used by copy activity.

```
{
    "name":"OneDestinationFile",
    "properties":{
        "linkedServiceName":{
            "referenceName":"AzureStorageLinkedService",
            "type":"LinkedServiceReference"
        },
        "parameters":{
            "Container":{
                "type":"String"
            },
            "Directory":{
                "type":"String"
            },
            "filename":{
                "type":"string"
            }
        },
        "annotations":[

        ],
        "type":"Binary",
        "typeProperties":{
            "location":{
                "type":"AzureBlobStorageLocation",
                "fileName":{
                    "value":"@dataset().filename",
                    "type":"Expression"
                },
                "folderPath":{
                    "value":"@{dataset().Directory}",
                    "type":"Expression"
                },
                "container":{
                    "value":"@{dataset().Container}",
                    "type":"Expression"
                }
            }
        }
    }
}
```

You can also get the template to move files from here.

Known limitation

- Delete activity does not support deleting list of folders described by wildcard.
- When using file attribute filter in delete activity: modifiedDatetimeStart and modifiedDatetimeEnd to select files to be deleted, make sure to set "wildcardFileName": "*" in delete activity as well.

Next steps

Learn more about moving files in Azure Data Factory and Synapse pipelines.

- Copy Data tool

Copy Data tool in Azure Data Factory and Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

The Copy Data tool eases and optimizes the process of ingesting data into a data lake, which is usually a first step in an end-to-end data integration scenario. It saves time, especially when you use the service to ingest data from a data source for the first time. Some of the benefits of using this tool are:

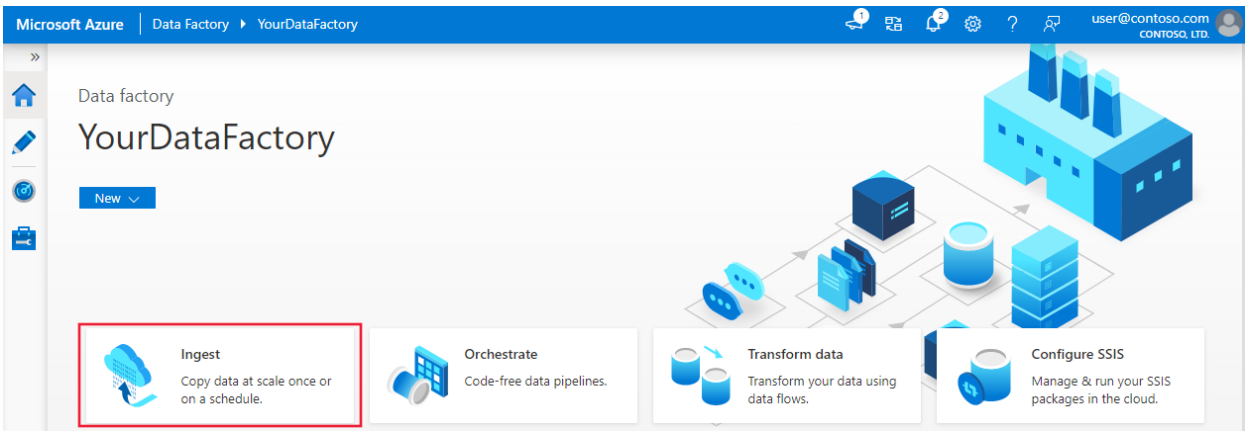
- When using the Copy Data tool, you do not need understand service definitions for linked services, datasets, pipelines, activities, and triggers.
- The flow of Copy Data tool is intuitive for loading data into a data lake. The tool automatically creates all the necessary resources to copy data from the selected source data store to the selected destination/sink data store.
- The Copy Data tool helps you validate the data that is being ingested at the time of authoring, which helps you avoid any potential errors at the beginning itself.
- If you need to implement complex business logic to load data into a data lake, you can still edit the resources created by the Copy Data tool by using the per-activity authoring in the UI.

The following table provides guidance on when to use the Copy Data tool vs. per-activity authoring in the UI:

COPY DATA TOOL	PER ACTIVITY (COPY ACTIVITY) AUTHORING
You want to easily build a data loading task without learning about entities (linked services, datasets, pipelines, etc.)	You want to implement complex and flexible logic for loading data into lake.
You want to quickly load a large number of data artifacts into a data lake.	You want to chain Copy activity with subsequent activities for cleansing or processing data.

To start the Copy Data tool, click the **Ingest** tile on the home page of your the Data Factory or Synapse Studio UI.

- Azure Data Factory
- Synapse Analytics



After you launch copy data tool, you will see two types of the tasks: one is **built-in copy task** and another is **metadata driven copy task**. The built-in copy task leads you to create a pipeline within five minutes to

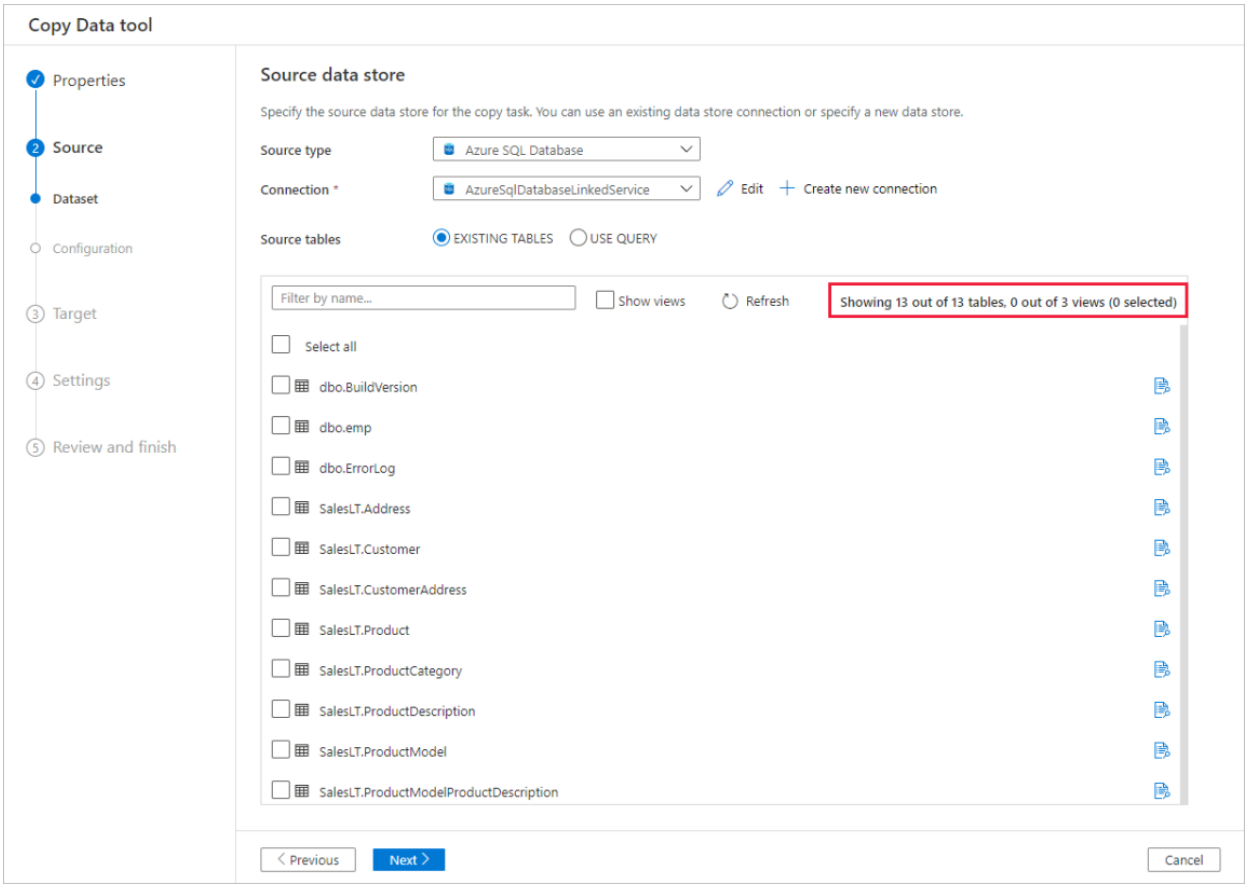
replicate data without learning about entities. The metadata driven copy task to ease your journey of creating parameterized pipelines and external control table in order to manage to copy large amounts of objects (for example, thousands of tables) at scale. You can see more details in metadata driven copy data.

Intuitive flow for loading data into a data lake

This tool allows you to easily move data from a wide variety of sources to destinations in minutes with an intuitive flow:

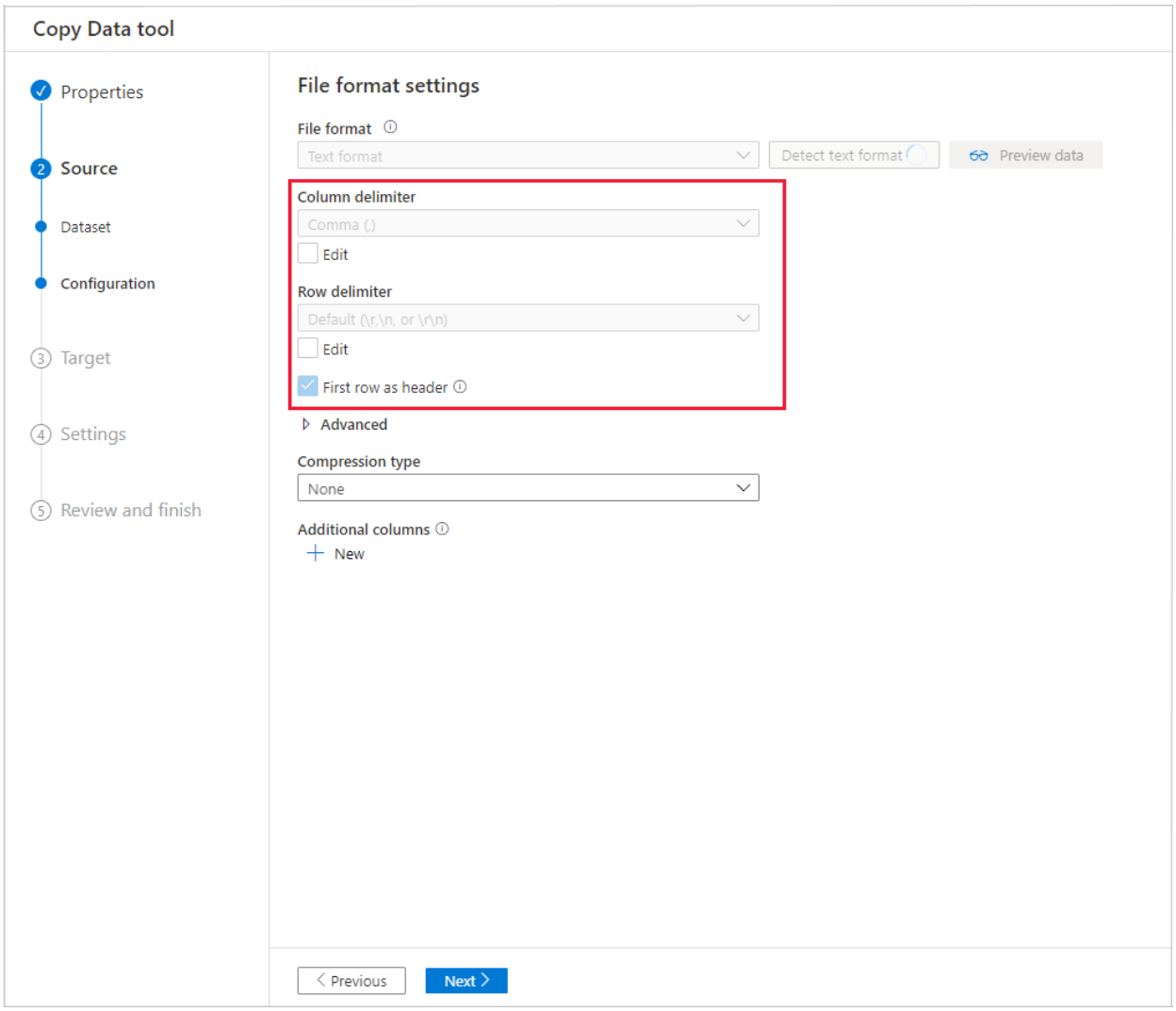
1. Configure settings for the **source**.
2. Configure settings for the **destination**.
3. Configure **advanced settings** for the copy operation such as column mapping, performance settings, and fault tolerance settings.
4. Specify a **schedule** for the data loading task.
5. Review **summary** of entities to be created.
6. **Edit** the pipeline to update settings for the copy activity as needed.

   The tool is designed with big data in mind from the start, with support for diverse data and object types. You can use it to move hundreds of folders, files, or tables. The tool supports automatic data preview, schema capture and automatic mapping, and data filtering as well.

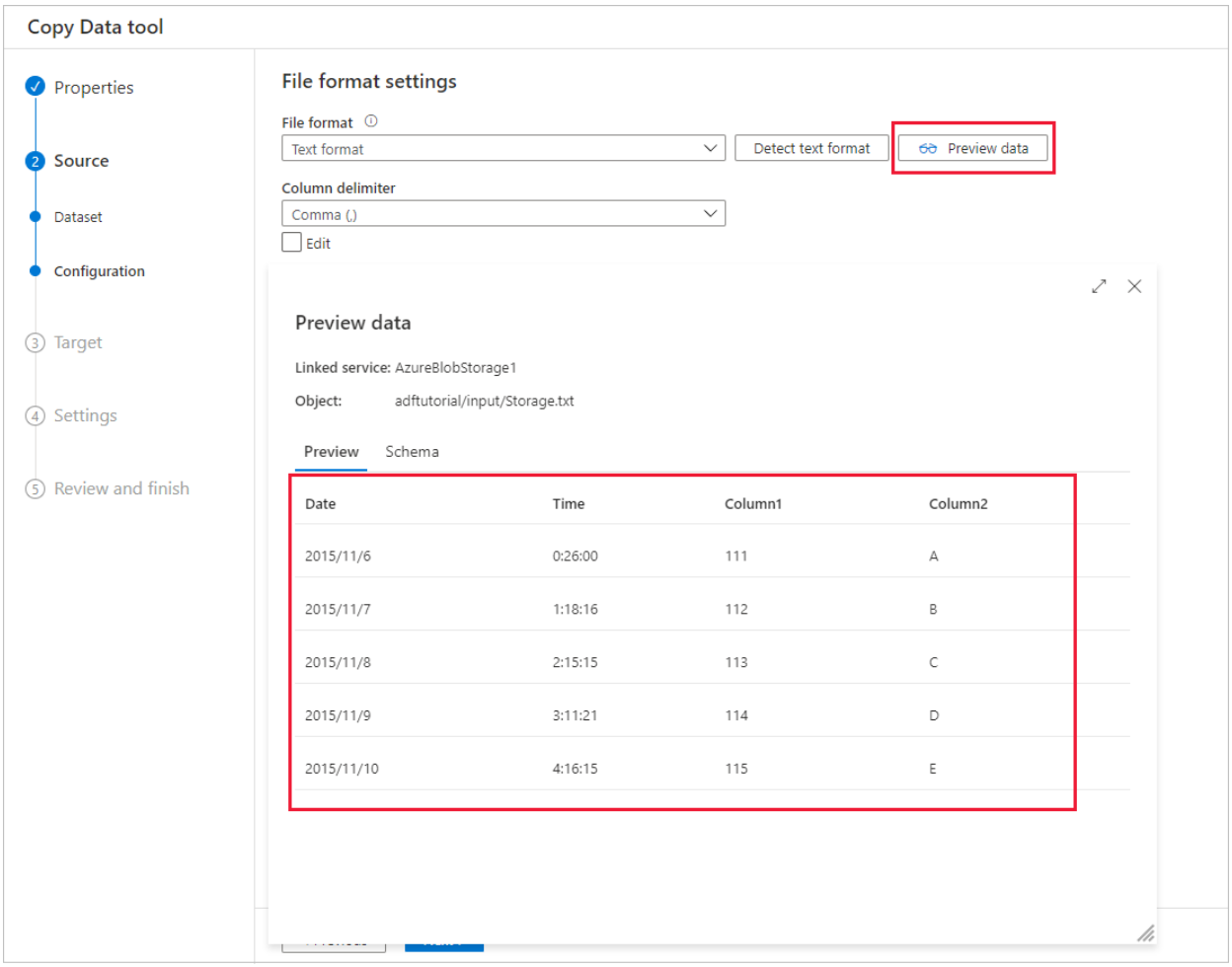


Automatic data preview

You can preview part of the data from the selected source data store, which allows you to validate the data that is being copied. In addition, if the source data is in a text file, the Copy Data tool parses the text file to automatically detect the row and column delimiters, and schema.



After the detection, select **Preview data**:



Schema capture and automatic mapping

The schema of data source may not be same as the schema of data destination in many cases. In this scenario, you need to map columns from the source schema to columns from the destination schema.

The Copy Data tool monitors and learns your behavior when you are mapping columns between source and destination stores. After you pick one or a few columns from source data store, and map them to the destination schema, the Copy Data tool starts to analyze the pattern for column pairs you picked from both sides. Then, it applies the same pattern to the rest of the columns. Therefore, you see all the columns have been mapped to the destination in a way you want just after several clicks. If you are not satisfied with the choice of column mapping provided by Copy Data tool, you can ignore it and continue with manually mapping the columns. Meanwhile, the Copy Data tool constantly learns and updates the pattern, and ultimately reaches the right pattern for the column mapping you want to achieve.

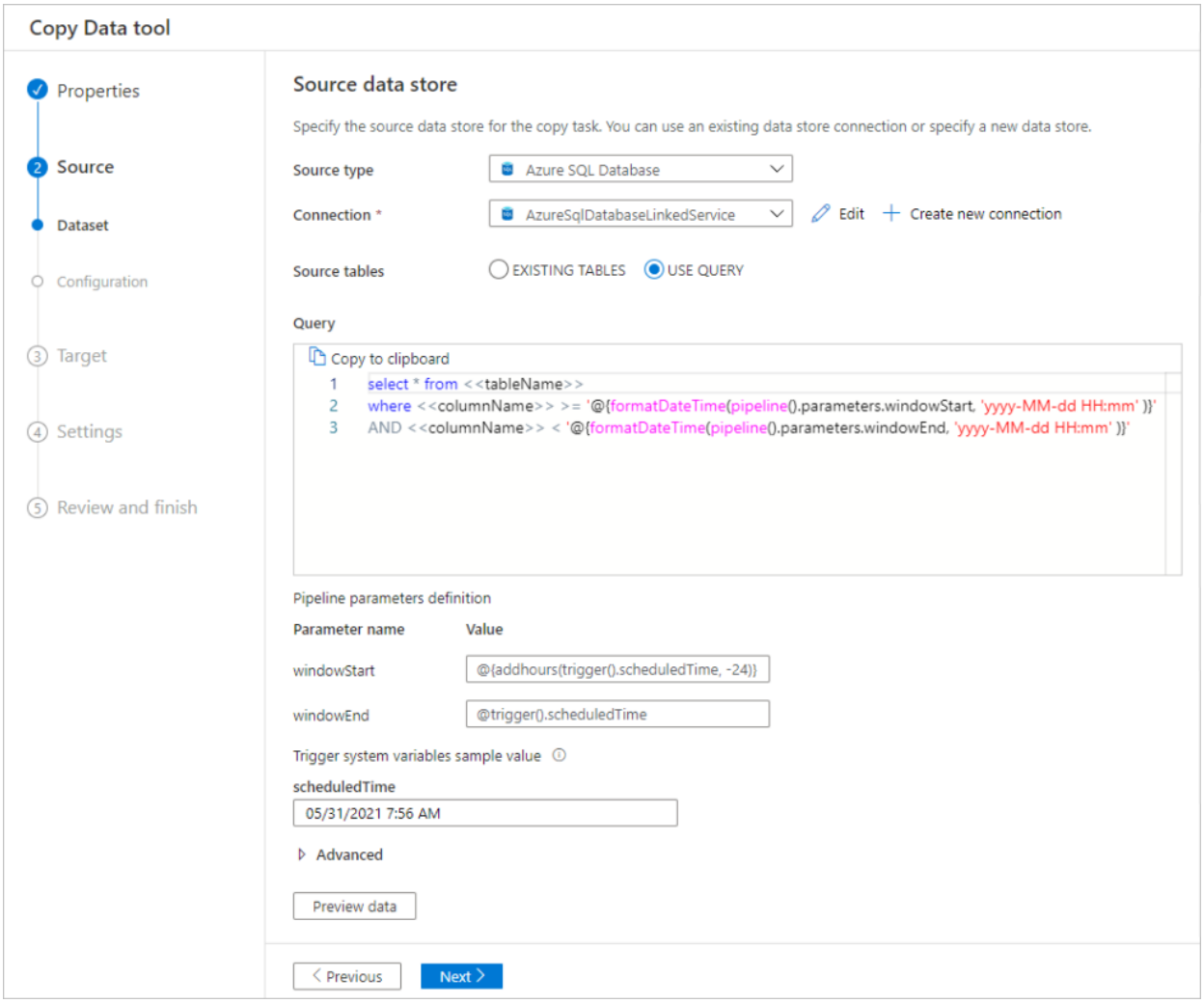
> **NOTE**
> When copying data from SQL Server or Azure SQL Database into Azure Synapse Analytics, if the table does not exist in the destination store, Copy Data tool supports creation of the table automatically by using the source schema.

Filter data

You can filter source data to select only the data that needs to be copied to the sink data store. Filtering reduces the volume of the data to be copied to the sink data store and therefore enhances the throughput of the copy operation. Copy Data tool provides a flexible way to filter data in a relational database by using the SQL query language, or files in an Azure blob folder.

Filter data in a database

The following screenshot shows a SQL query to filter the data.



Filter data in an Azure blob folder

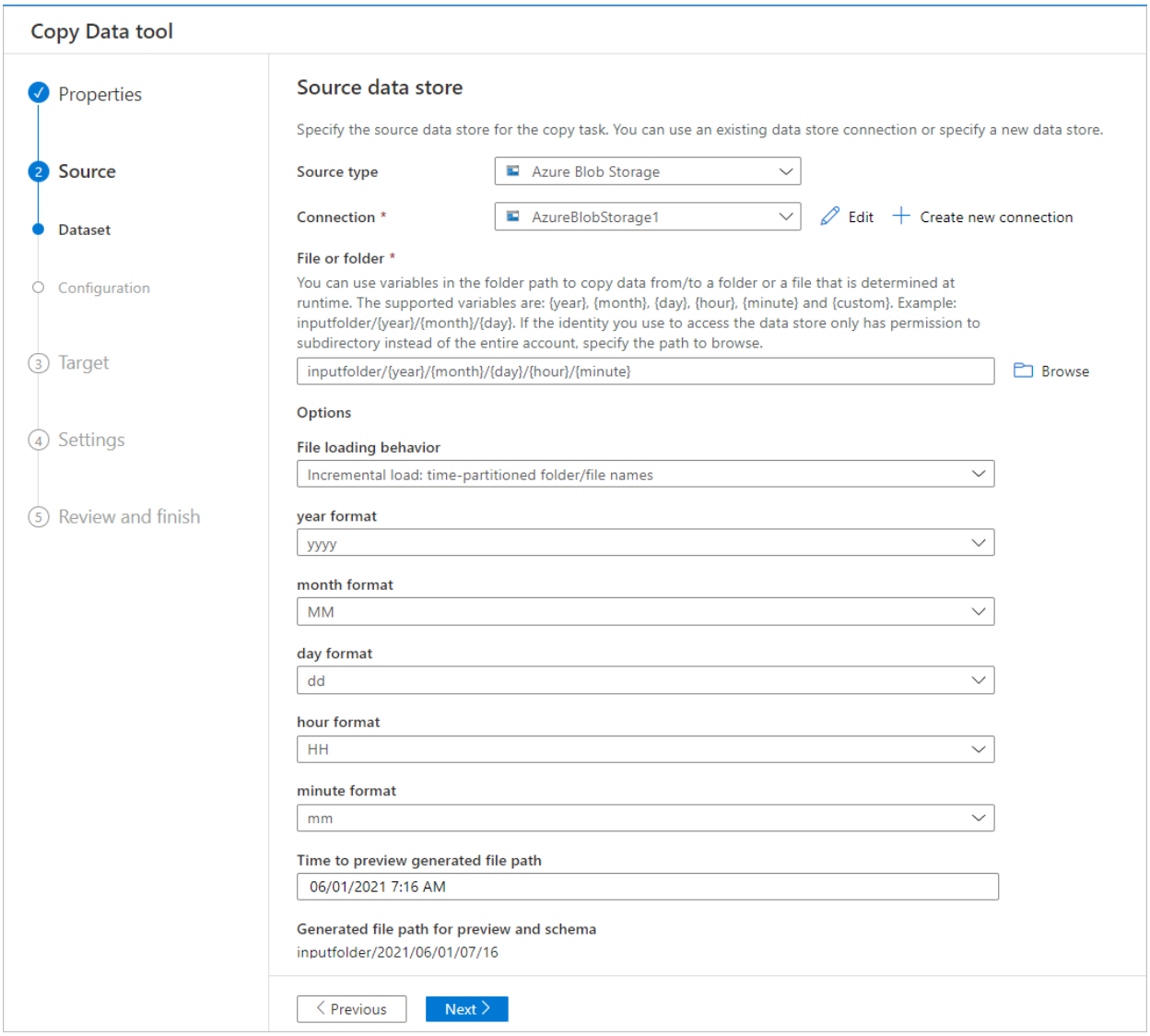
You can use variables in the folder path to copy data from a folder. The supported variables are: **{year}**, **{month}**, **{day}**, **{hour}**, and **{minute}**. For example: inputfolder/{year}/{month}/{day}.

Suppose that you have input folders in the following format:

```
2016/03/01/01
2016/03/01/02
2016/03/01/03
...
```

Click the **Browse** button for **File or folder**, browse to one of these folders (for example, 2016->03->01->02), and click **Choose**. You should see 2016/03/01/02 in the text box.

Then, replace **2016** with **{year}**, **03** with **{month}**, **01** with **{day}**, and **02** with **{hour}**, and press the **Tab** key. When you select **Incremental load: time-partitioned folder/file names** in the **File loading behavior** section and you select **Schedule** or **Tumbling window** on the **Properties** page, you should see drop-down lists to select the format for these four variables:

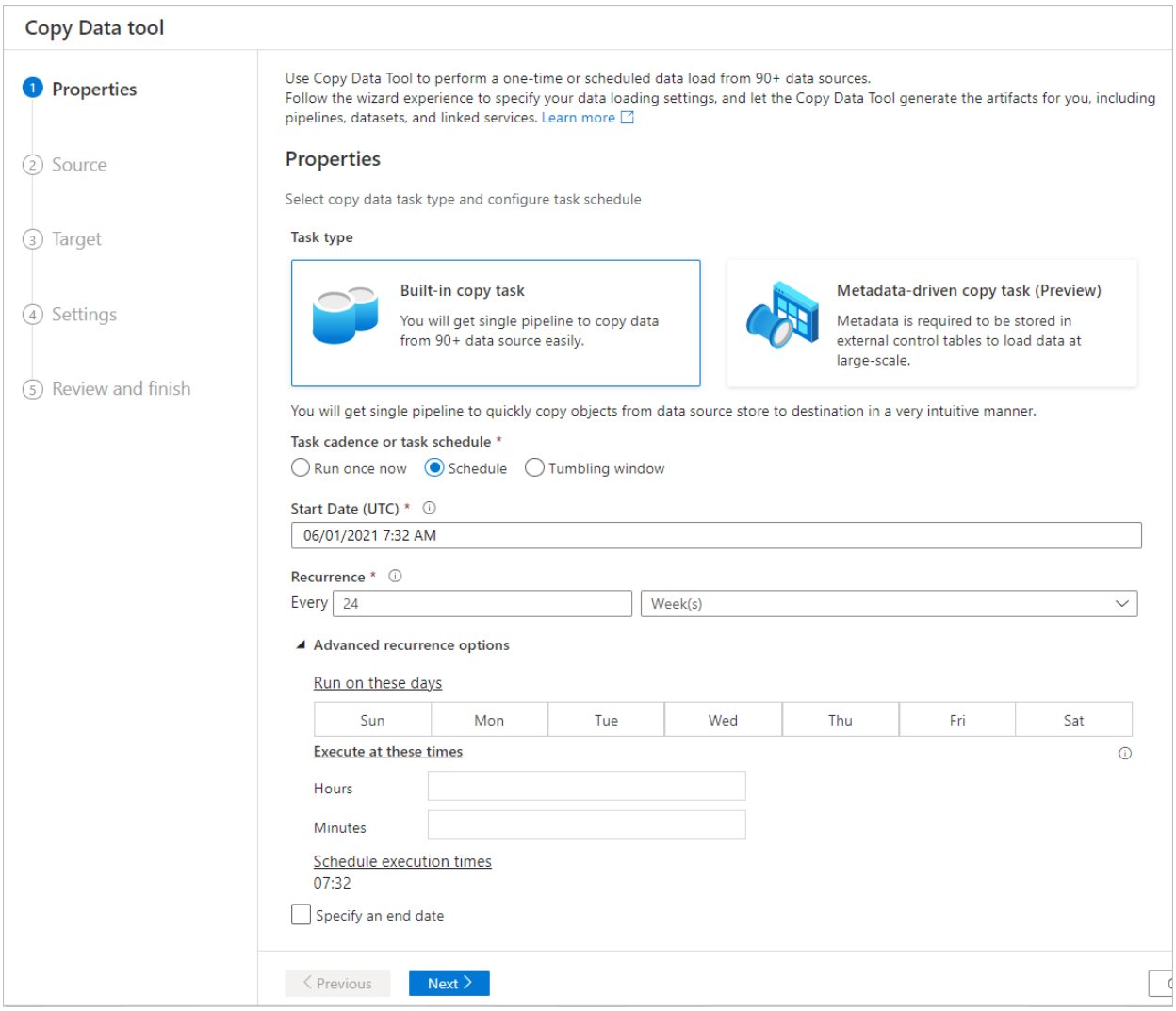


The Copy Data tool generates parameters with expressions, functions, and system variables that can be used to represent {year}, {month}, {day}, {hour}, and {minute} when creating pipeline.

Scheduling options

You can run the copy operation once or on a schedule (hourly, daily, and so on). These options can be used for the connectors across different environments, including on-premises, cloud, and local desktop.

A one-time copy operation enables data movement from a source to a destination only once. It applies to data of any size and any supported format. The scheduled copy allows you to copy data on a recurrence that you specify. You can use rich settings (like retry, timeout, and alerts) to configure the scheduled copy.



Next steps

Try these tutorials that use the Copy Data tool:

- Quickstart: create a data factory using the Copy Data tool
- Tutorial: copy data in Azure using the Copy Data tool
- Tutorial: copy on-premises data to Azure using the Copy Data tool

Supported file formats and compression codecs by copy activity in Azure Data Factory and Azure Synapse pipelines

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article applies to the following connectors: Amazon S3, Amazon S3 Compatible Storage, Azure Blob, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure Files, File System, FTP, Google Cloud Storage, HDFS, HTTP, Oracle Cloud Storage and SFTP.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

You can use the Copy activity to copy files as-is between two file-based data stores, in which case the data is copied efficiently without any serialization or deserialization.

In addition, you can also parse or generate files of a given format. For example, you can perform the following:

- Copy data from a SQL Server database and write to Azure Data Lake Storage Gen2 in Parquet format.
- Copy files in text (CSV) format from an on-premises file system and write to Azure Blob storage in Avro format.
- Copy zipped files from an on-premises file system, decompress them on-the-fly, and write extracted files to Azure Data Lake Storage Gen2.
- Copy data in Gzip compressed-text (CSV) format from Azure Blob storage and write it to Azure SQL Database.
- Many more activities that require serialization/deserialization or compression/decompression.

Next steps

See the other Copy Activity articles:

- Copy activity overview
- Copy activity performance

Copy activity performance and scalability guide

8/19/2022 • 7 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Sometimes you want to perform a large-scale data migration from data lake or enterprise data warehouse (EDW), to Azure. Other times you want to ingest large amounts of data, from different sources into Azure, for big data analytics. In each case, it is critical to achieve optimal performance and scalability.

Azure Data Factory and Azure Synapse Analytics pipelines provide a mechanism to ingest data, with the following advantages:

- Handles large amounts of data
- Is highly performant
- Is cost-effective

These advantages are an excellent fit for data engineers who want to build scalable data ingestion pipelines that are highly performant.

After reading this article, you will be able to answer the following questions:

- What level of performance and scalability can I achieve using copy activity for data migration and data ingestion scenarios?
- What steps should I take to tune the performance of the copy activity?
- What performance optimizations can I utilize for a single copy activity run?
- What other external factors to consider when optimizing copy performance?

> **NOTE**
> If you aren't familiar with the copy activity in general, see the copy activity overview before you read this article.

Copy performance and scalability achievable using Azure Data Factory and Synapse pipelines

Azure Data Factory and Synapse pipelines offer a serverless architecture that allows parallelism at different levels.

This architecture allows you to develop pipelines that maximize data movement throughput for your environment. These pipelines fully utilize the following resources:

- Network bandwidth between the source and destination data stores
- Source or destination data store input/output operations per second (IOPS) and bandwidth

This full utilization means you can estimate the overall throughput by measuring the minimum throughput available with the following resources:

- Source data store
- Destination data store
- Network bandwidth in between the source and destination data stores

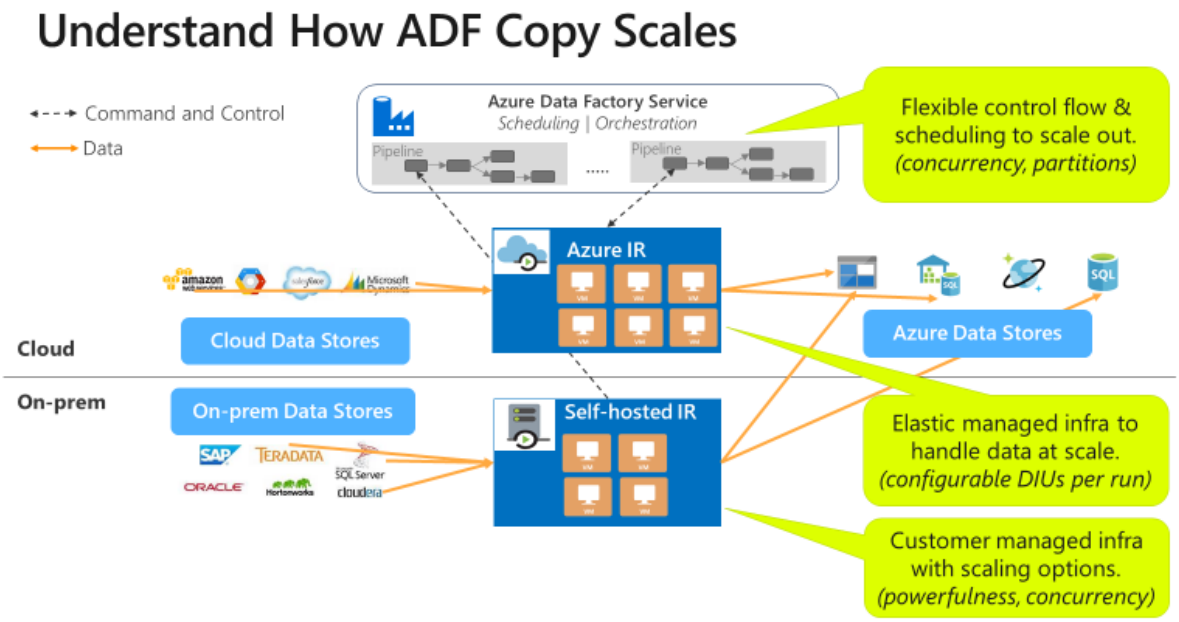
The table below shows the calculation of data movement duration. The duration in each cell is calculated based

on a given network and data store bandwidth and a given data payload size.

> **NOTE**
>
> The duration provided below are meant to represent achievable performance in an end-to-end data integration solution by using one or more performance optimization techniques described in Copy performance optimization features, including using ForEach to partition and spawn off multiple concurrent copy activities. We recommend you to follow steps laid out in Performance tuning steps to optimize copy performance for your specific dataset and system configuration. You should use the numbers obtained in your performance tuning tests for production deployment planning, capacity planning, and billing projection.

DATA SIZE / BANDWIDTH	50 MBPS	100 MBPS	500 MBPS	1 GBPS	5 GBPS	10 GBPS	50 GBPS
1 GB	2.7 min	1.4 min	0.3 min	0.1 min	0.03 min	0.01 min	0.0 min
10 GB	27.3 min	13.7 min	2.7 min	1.3 min	0.3 min	0.1 min	0.03 min
100 GB	4.6 hrs	2.3 hrs	0.5 hrs	0.2 hrs	0.05 hrs	0.02 hrs	0.0 hrs
1 TB	46.6 hrs	23.3 hrs	4.7 hrs	2.3 hrs	0.5 hrs	0.2 hrs	0.05 hrs
10 TB	19.4 days	9.7 days	1.9 days	0.9 days	0.2 days	0.1 days	0.02 days
100 TB	194.2 days	97.1 days	19.4 days	9.7 days	1.9 days	1 day	0.2 days
1 PB	64.7 mo	32.4 mo	6.5 mo	3.2 mo	0.6 mo	0.3 mo	0.06 mo
10 PB	647.3 mo	323.6 mo	64.7 mo	31.6 mo	6.5 mo	3.2 mo	0.6 mo

Copy is scalable at different levels:



- Control flow can start multiple copy activities in parallel, for example using For Each loop.
- A single copy activity can take advantage of scalable compute resources.
  - When using Azure integration runtime (IR), you can specify up to 256 data integration units (DIUs) for each copy activity, in a serverless manner.
  - When using self-hosted IR, you can take either of the following approaches:
    - Manually scale up the machine.
    - Scale out to multiple machines (up to 4 nodes), and a single copy activity will partition its file set across all nodes.
- A single copy activity reads from and writes to the data store using multiple threads in parallel.

Performance tuning steps

Take the following steps to tune the performance of your service with the copy activity:

1. **Pick up a test dataset and establish a baseline.**

   During development, test your pipeline by using the copy activity against a representative data sample. The dataset you choose should represent your typical data patterns along the following attributes:

   - Folder structure
   - File pattern
   - Data schema

   And your dataset should be big enough to evaluate copy performance. A good size takes at least 10 minutes for copy activity to complete. Collect execution details and performance characteristics following copy activity monitoring.

2. **How to maximize performance of a single copy activity**:

   We recommend you to first maximize performance using a single copy activity.

   - **If the copy activity is being executed on an *Azure* integration runtime:**

     Start with default values for Data Integration Units (DIU) and parallel copy settings.

   - **If the copy activity is being executed on a *self-hosted* integration runtime:**

We recommend that you use a dedicated machine to host IR. The machine should be separate from the server hosting the data store. Start with default values for parallel copy setting and using a single node for the self-hosted IR.

Conduct a performance test run. Take a note of the performance achieved. Include the actual values used, such as DIUs and parallel copies. Refer to copy activity monitoring on how to collect run results and performance settings used. Learn how to troubleshoot copy activity performance to identify and resolve the bottleneck.

Iterate to conduct additional performance test runs following the troubleshooting and tuning guidance. Once single copy activity runs cannot achieve better throughput, consider whether to maximize aggregate throughput by running multiple copies concurrently. This option is discussed in the next numbered bullet.

3. **How to maximize aggregate throughput by running multiple copies concurrently:**

   By now you have maximized the performance of a single copy activity. If you have not yet achieved the throughput upper limits of your environment, you can run multiple copy activities in parallel. You can run in parallel by using control flow constructs. One such construct is the For Each loop. For more information, see the following articles about solution templates:

   - Copy files from multiple containers
   - Migrate data from Amazon S3 to ADLS Gen2
   - Bulk copy with a control table

4. **Expand the configuration to your entire dataset.**

   When you're satisfied with the execution results and performance, you can expand the definition and pipeline to cover your entire dataset.

Troubleshoot copy activity performance

Follow the Performance tuning steps to plan and conduct performance test for your scenario. And learn how to troubleshoot each copy activity run's performance issue from Troubleshoot copy activity performance.

Copy performance optimization features

The service provides the following performance optimization features:

- Data Integration Units
- Self-hosted integration runtime scalability
- Parallel copy
- Staged copy

Data Integration Units

A Data Integration Unit (DIU) is a measure that represents the power of a single unit in Azure Data Factory and Synapse pipelines. Power is a combination of CPU, memory, and network resource allocation. DIU only applies to Azure integration runtime. DIU does not apply to self-hosted integration runtime. Learn more here.

Self-hosted integration runtime scalability

You might want to host an increasing concurrent workload. Or you might want to achieve higher performance in your present workload level. You can enhance the scale of processing by the following approaches:

- You can scale *up* the self-hosted IR, by increasing the number of concurrent jobs that can run on a node. Scale up works only if the processor and memory of the node are being less than fully utilized.
- You can scale *out* the self-hosted IR, by adding more nodes (machines).

For more information, see:

- Copy activity performance optimization features: Self-hosted integration runtime scalability
- Create and configure a self-hosted integration runtime: Scale considerations

Parallel copy

You can set the `parallelCopies` property to indicate the parallelism you want the copy activity to use. Think of this property as the maximum number of threads within the copy activity. The threads operate in parallel. The threads either read from your source, or write to your sink data stores. Learn more.

Staged copy

A data copy operation can send the data *directly* to the sink data store. Alternatively, you can choose to use Blob storage as an *interim staging* store. Learn more.

Next steps

See the other copy activity articles:

- Copy activity overview
- Troubleshoot copy activity performance
- Copy activity performance optimization features
- Use Azure Data Factory to migrate data from your data lake or data warehouse to Azure
- Migrate data from Amazon S3 to Azure Storage

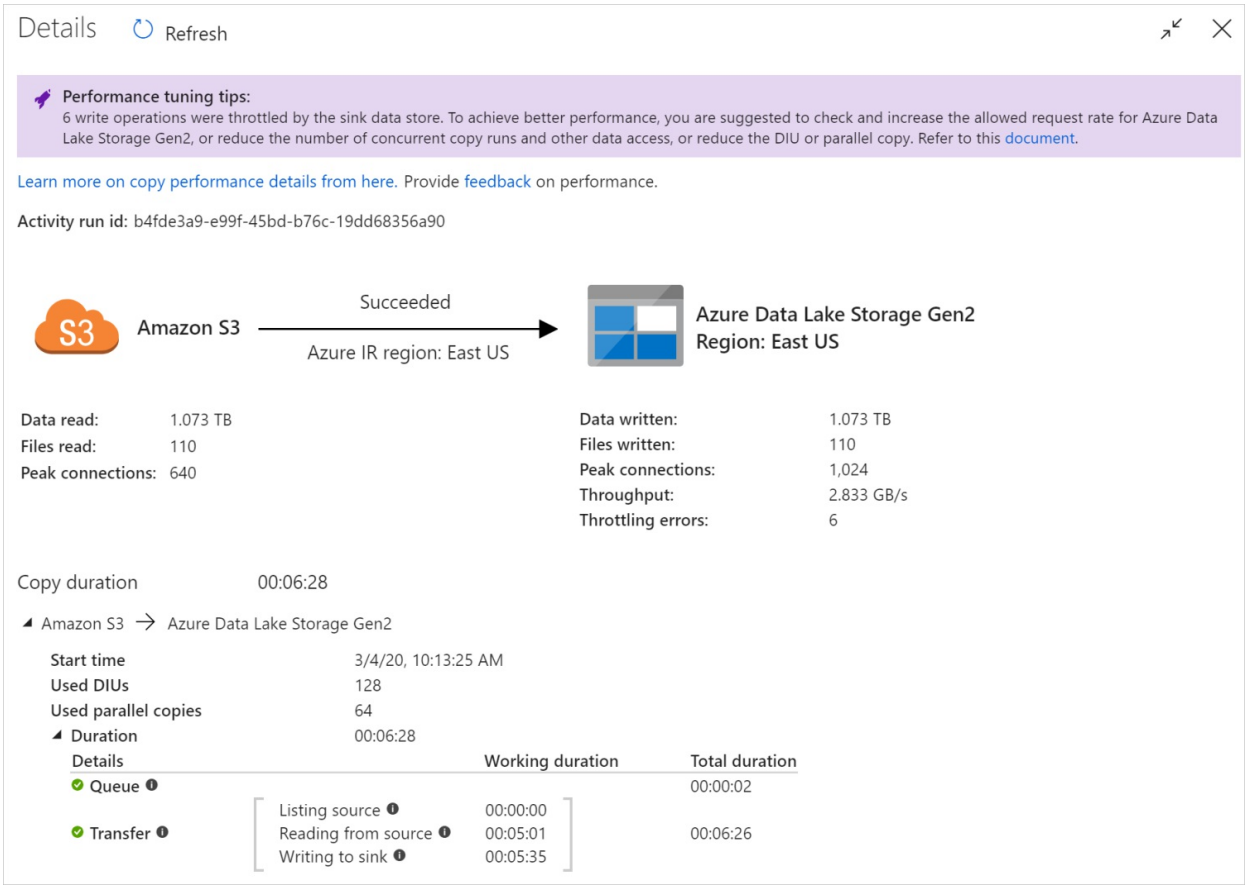
Troubleshoot copy activity performance

8/19/2022 • 15 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to troubleshoot copy activity performance issue in Azure Data Factory.

After you run a copy activity, you can collect the run result and performance statistics in copy activity monitoring view. The following is an example.



Performance tuning tips

In some scenarios, when you run a copy activity, you'll see **"Performance tuning tips"** at the top as shown in the above example. The tips tell you the bottleneck identified by the service for this particular copy run, along with suggestion on how to boost copy throughput. Try making the recommended change, then run the copy again.

As a reference, currently the performance tuning tips provide suggestions for the following cases:

CATEGORY	PERFORMANCE TUNING TIPS
Data store specific	Loading data into **Azure Synapse Analytics**: suggest using PolyBase or COPY statement if it's not used.
	Copying data from/to **Azure SQL Database**: when DTU is under high utilization, suggest upgrading to higher tier.

CATEGORY	PERFORMANCE TUNING TIPS
	Copying data from/to **Azure Cosmos DB**: when RU is under high utilization, suggest upgrading to larger RU.
	Copying data from **SAP Table**: when copying large amount of data, suggest leveraging SAP connector's partition option to enable parallel load and increase the max partition number.
	Ingesting data from **Amazon Redshift**: suggest using UNLOAD if it's not used.
Data store throttling	If a number of read/write operations are throttled by the data store during copy, suggest checking and increase the allowed request rate for the data store, or reduce the concurrent workload.
Integration runtime	If you use a **Self-hosted Integration Runtime (IR)** and copy activity waits long in the queue until the IR has available resource to execute, suggest scaling out/up your IR.
	If you use an **Azure Integration Runtime** that is in a not optimal region resulting in slow read/write, suggest configuring to use an IR in another region.
Fault tolerance	If you configure fault tolerance and skipping incompatible rows results in slow performance, suggest ensuring source and sink data are compatible.
Staged copy	If staged copy is configured but not helpful for your source-sink pair, suggest removing it.
Resume	When copy activity is resumed from last failure point but you happen to change the DIU setting after the original run, note the new DIU setting doesn't take effect.

Understand copy activity execution details

The execution details and durations at the bottom of the copy activity monitoring view describes the key stages your copy activity goes through (see example at the beginning of this article), which is especially useful for troubleshooting the copy performance. The bottleneck of your copy run is the one with the longest duration. Refer to the following table on each stage's definition, and learn how to Troubleshoot copy activity on Azure IR and Troubleshoot copy activity on Self-hosted IR with such info.

STAGE	DESCRIPTION
Queue	The elapsed time until the copy activity actually starts on the integration runtime.
Pre-copy script	The elapsed time between copy activity starting on IR and copy activity finishing executing the pre-copy script in sink data store. Apply when you configure the pre-copy script for database sinks, e.g. when writing data into Azure SQL Database do clean up before copy new data.

STAGE	DESCRIPTION
Transfer	The elapsed time between the end of the previous step and the IR transferring all the data from source to sink. Note the sub-steps under transfer run in parallel, and some operations are not shown now e.g. parsing/generating file format. - **Time to first byte:** The time elapsed between the end of the previous step and the time when the IR receives the first byte from the source data store. Applies to non-file-based sources. - **Listing source:** The amount of time spent on enumerating source files or data partitions. The latter applies when you configure partition options for database sources, e.g. when copy data from databases like Oracle/SAP HANA/Teradata/Netezza/etc. -**Reading from source:** The amount of time spent on retrieving data from source data store. - **Writing to sink:** The amount of time spent on writing data to sink data store. Note some connectors do not have this metric at the moment, including Azure Cognitive Search, Azure Data Explorer, Azure Table storage, Oracle, SQL Server, Common Data Service, Dynamics 365, Dynamics CRM, Salesforce/Salesforce Service Cloud.

Troubleshoot copy activity on Azure IR

Follow the Performance tuning steps to plan and conduct performance test for your scenario.

When the copy activity performance doesn't meet your expectation, to troubleshoot single copy activity running on Azure Integration Runtime, if you see performance tuning tips shown up in the copy monitoring view, apply the suggestion and try again. Otherwise, understand copy activity execution details, check which stage has the **longest** duration, and apply the guidance below to boost copy performance:

- **"Pre-copy script" experienced long duration:** it means the pre-copy script running on sink database takes long to finish. Tune the specified pre-copy script logic to enhance the performance. If you need further help on improving the script, contact your database team.
- **"Transfer - Time to first byte" experienced long working duration**: it means your source query takes long to return any data. Check and optimize the query or server. If you need further help, contact your data store team.
- **"Transfer - Listing source" experienced long working duration**: it means enumerating source files or source database data partitions is slow.
  - When copying data from file-based source, if you use **wildcard filter** on folder path or file name (`wildcardFolderPath` or `wildcardFileName`), or use **file last modified time filter** (`modifiedDatetimeStart` or `modifiedDatetimeEnd`), note such filter would result in copy activity listing all the files under the specified folder to client side then apply the filter. Such file enumeration could become the bottleneck especially when only small set of files met the filter rule.
    - Check whether you can copy files based on datetime partitioned file path or name. Such way doesn't bring burden on listing source side.
    - Check if you can use data store's native filter instead, specifically "**prefix**" for Amazon S3/Azure Blob storage/Azure Files and "**listAfter/listBefore**" for ADLS Gen1. Those filters are data store server-side filter and would have much better performance.

    - Consider to split single large data set into several smaller data sets, and let those copy jobs run concurrently each tackles portion of data. You can do this with Lookup/GetMetadata + ForEach + Copy. Refer to Copy files from multiple containers or Migrate data from Amazon S3 to ADLS Gen2 solution templates as general example.
  - Check if the service reports any throttling error on source or if your data store is under high utilization state. If so, either reduce your workloads on the data store, or try contacting your data store administrator to increase the throttling limit or available resource.
  - Use Azure IR in the same or close to your source data store region.
- **"Transfer - reading from source" experienced long working duration**:
  - Adopt connector-specific data loading best practice if applies. For example, when copying data from Amazon Redshift, configure to use Redshift UNLOAD.
  - Check if the service reports any throttling error on source or if your data store is under high utilization. If so, either reduce your workloads on the data store, or try contacting your data store administrator to increase the throttling limit or available resource.
  - Check your copy source and sink pattern:
    - If your copy pattern supports larger than 4 Data Integration Units (DIUs) - refer to this section on details, generally you can try increasing DIUs to get better performance.
    - Otherwise, consider to split single large data set into several smaller data sets, and let those copy jobs run concurrently each tackles portion of data. You can do this with Lookup/GetMetadata + ForEach + Copy. Refer to Copy files from multiple containers, Migrate data from Amazon S3 to ADLS Gen2, or Bulk copy with a control table solution templates as general example.
  - Use Azure IR in the same or close to your source data store region.
- **"Transfer - writing to sink" experienced long working duration**:
  - Adopt connector-specific data loading best practice if applies. For example, when copying data into Azure Synapse Analytics, use PolyBase or COPY statement.
  - Check if the service reports any throttling error on sink or if your data store is under high utilization. If so, either reduce your workloads on the data store, or try contacting your data store administrator to increase the throttling limit or available resource.
  - Check your copy source and sink pattern:
    - If your copy pattern supports larger than 4 Data Integration Units (DIUs) - refer to this section on details, generally you can try increasing DIUs to get better performance.
    - Otherwise, gradually tune the parallel copies, note that too many parallel copies may even hurt the performance.
  - Use Azure IR in the same or close to your sink data store region.

Troubleshoot copy activity on Self-hosted IR

Follow the Performance tuning steps to plan and conduct performance test for your scenario.

When the copy performance doesn't meet your expectation, to troubleshoot single copy activity running on Azure Integration Runtime, if you see performance tuning tips shown up in the copy monitoring view, apply the suggestion and try again. Otherwise, understand copy activity execution details, check which stage has the **longest** duration, and apply the guidance below to boost copy performance:

- **"Queue" experienced long duration:** it means the copy activity waits long in the queue until your Self-hosted IR has resource to execute. Check the IR capacity and usage, and scale up or out according to your workload.
- **"Transfer - Time to first byte" experienced long working duration**: it means your source query takes long to return any data. Check and optimize the query or server. If you need further help, contact your data store team.
- **"Transfer - Listing source" experienced long working duration**: it means enumerating source files or source database data partitions is slow.
    - Check if the Self-hosted IR machine has low latency connecting to source data store. If your source is in Azure, you can use this tool to check the latency from the Self-hosted IR machine to the Azure region, the less the better.
    - When copying data from file-based source, if you use **wildcard filter** on folder path or file name (`wildcardFolderPath` or `wildcardFileName`), or use **file last modified time filter** (`modifiedDatetimeStart` or `modifiedDatetimeEnd`), note such filter would result in copy activity listing all the files under the specified folder to client side then apply the filter. Such file enumeration could become the bottleneck especially when only small set of files met the filter rule.
        - Check whether you can copy files based on datetime partitioned file path or name. Such way doesn't bring burden on listing source side.
        - Check if you can use data store's native filter instead, specifically "**prefix**" for Amazon S3/Azure Blob storage/Azure Files and "**listAfter/listBefore**" for ADLS Gen1. Those filters are data store server-side filter and would have much better performance.
        - Consider to split single large data set into several smaller data sets, and let those copy jobs run concurrently each tackles portion of data. You can do this with Lookup/GetMetadata + ForEach + Copy. Refer to Copy files from multiple containers or Migrate data from Amazon S3 to ADLS Gen2 solution templates as general example.
    - Check if the service reports any throttling error on source or if your data store is under high utilization state. If so, either reduce your workloads on the data store, or try contacting your data store administrator to increase the throttling limit or available resource.
- **"Transfer - reading from source" experienced long working duration**:
    - Check if the Self-hosted IR machine has low latency connecting to source data store. If your source is in Azure, you can use this tool to check the latency from the Self-hosted IR machine to the Azure regions, the less the better.
    - Check if the Self-hosted IR machine has enough inbound bandwidth to read and transfer the data efficiently. If your source data store is in Azure, you can use this tool to check the download speed.
    - Check the Self-hosted IR's CPU and memory usage trend in Azure portal -> your data factory or Synapse workspace -> overview page. Consider to scale up/out IR if the CPU usage is high or available memory is low.
    - Adopt connector-specific data loading best practice if applies. For example:
        - When copying data from Oracle, Netezza, Teradata, SAP HANA, SAP Table, and SAP Open Hub), enable data partition options to copy data in parallel.
        - When copying data from HDFS, configure to use DistCp.
        - When copying data from Amazon Redshift, configure to use Redshift UNLOAD.

- Check if the service reports any throttling error on source or if your data store is under high utilization. If so, either reduce your workloads on the data store, or try contacting your data store administrator to increase the throttling limit or available resource.
- Check your copy source and sink pattern:
  - If you copy data from partition-option-enabled data stores, consider to gradually tune the parallel copies, note that too many parallel copies may even hurt the performance.
  - Otherwise, consider to split single large data set into several smaller data sets, and let those copy jobs run concurrently each tackles portion of data. You can do this with Lookup/GetMetadata + ForEach + Copy. Refer to Copy files from multiple containers, Migrate data from Amazon S3 to ADLS Gen2, or Bulk copy with a control table solution templates as general example.

- **"Transfer - writing to sink" experienced long working duration**:
  - Adopt connector-specific data loading best practice if applies. For example, when copying data into Azure Synapse Analytics, use PolyBase or COPY statement.
  - Check if the Self-hosted IR machine has low latency connecting to sink data store. If your sink is in Azure, you can use this tool to check the latency from the Self-hosted IR machine to the Azure region, the less the better.
  - Check if the Self-hosted IR machine has enough outbound bandwidth to transfer and write the data efficiently. If your sink data store is in Azure, you can use this tool to check the upload speed.
  - Check if the Self-hosted IR's CPU and memory usage trend in Azure portal -> your data factory or Synapse workspace -> overview page. Consider to scale up/out IR if the CPU usage is high or available memory is low.
  - Check if the service reports any throttling error on sink or if your data store is under high utilization. If so, either reduce your workloads on the data store, or try contacting your data store administrator to increase the throttling limit or available resource.
  - Consider to gradually tune the parallel copies, note that too many parallel copies may even hurt the performance.

Connector and IR performance

This section explores some performance troubleshooting guides for particular connector type or integration runtime.

Activity execution time varies using Azure IR vs Azure VNet IR

Activity execution time varies when the dataset is based on different Integration Runtime.

- **Symptoms**: Simply toggling the Linked Service dropdown in the dataset performs the same pipeline activities, but has drastically different run-times. When the dataset is based on the Managed Virtual Network Integration Runtime, it takes more time on average than the run when based on the Default Integration Runtime.
- **Cause**: Checking the details of pipeline runs, you can see that the slow pipeline is running on Managed VNet (Virtual Network) IR while the normal one is running on Azure IR. By design, Managed VNet IR takes longer queue time than Azure IR as we are not reserving one compute node per service instance, so there is a warm up for each copy activity to start, and it occurs primarily on VNet join rather than Azure IR.

Low performance when loading data into Azure SQL Database

- **Symptoms**: Copying data in to Azure SQL Database turns to be slow.
- **Cause**: The root cause of the issue is mostly triggered by the bottleneck of Azure SQL Database side. Following are some possible causes:
  - Azure SQL Database tier is not high enough.
  - Azure SQL Database DTU usage is close to 100%. You can monitor the performance and consider to upgrade the Azure SQL Database tier.
  - Indexes are not set properly. Remove all the indexes before data load and recreate them after load complete.
  - WriteBatchSize is not large enough to fit schema row size. Try to enlarge the property for the issue.
  - Instead of bulk insert, stored procedure is being used, which is expected to have worse performance.

Timeout or slow performance when parsing large Excel file

- **Symptoms**:
  - When you create Excel dataset and import schema from connection/store, preview data, list, or refresh worksheets, you may hit timeout error if the excel file is large in size.
  - When you use copy activity to copy data from large Excel file (>= 100 MB) into other data store, you may experience slow performance or OOM issue.
- **Cause**:
  - For operations like importing schema, previewing data, and listing worksheets on excel dataset, the timeout is 100 s and static. For large Excel file, these operations may not finish within the timeout value.
  - The copy activity reads the whole Excel file into memory then locate the specified worksheet and cells to read data. This behavior is due to the underlying SDK the service uses.
- **Resolution**:
  - For importing schema, you can generate a smaller sample file, which is a subset of original file, and choose "import schema from sample file" instead of "import schema from connection/store".
  - For listing worksheet, in the worksheet dropdown, you can click "Edit" and input the sheet name/index instead.
  - To copy large excel file (>100 MB) into other store, you can use Data Flow Excel source which sport streaming read and perform better.

Other references

Here is performance monitoring and tuning references for some of the supported data stores:

- Azure Blob storage: Scalability and performance targets for Blob storage and Performance and scalability checklist for Blob storage.
- Azure Table storage: Scalability and performance targets for Table storage and Performance and scalability checklist for Table storage.
- Azure SQL Database: You can monitor the performance and check the Database Transaction Unit (DTU) percentage.
- Azure Synapse Analytics: Its capability is measured in Data Warehouse Units (DWUs). See Manage compute power in Azure Synapse Analytics (Overview).

- Azure Cosmos DB: Performance levels in Azure Cosmos DB.
- SQL Server: Monitor and tune for performance.
- On-premises file server: Performance tuning for file servers.

Next steps

See the other copy activity articles:

- Copy activity overview
- Copy activity performance and scalability guide
- Copy activity performance optimization features
- Migrate data from your data lake or data warehouse to Azure
- Migrate data from Amazon S3 to Azure Storage

Copy activity performance optimization features

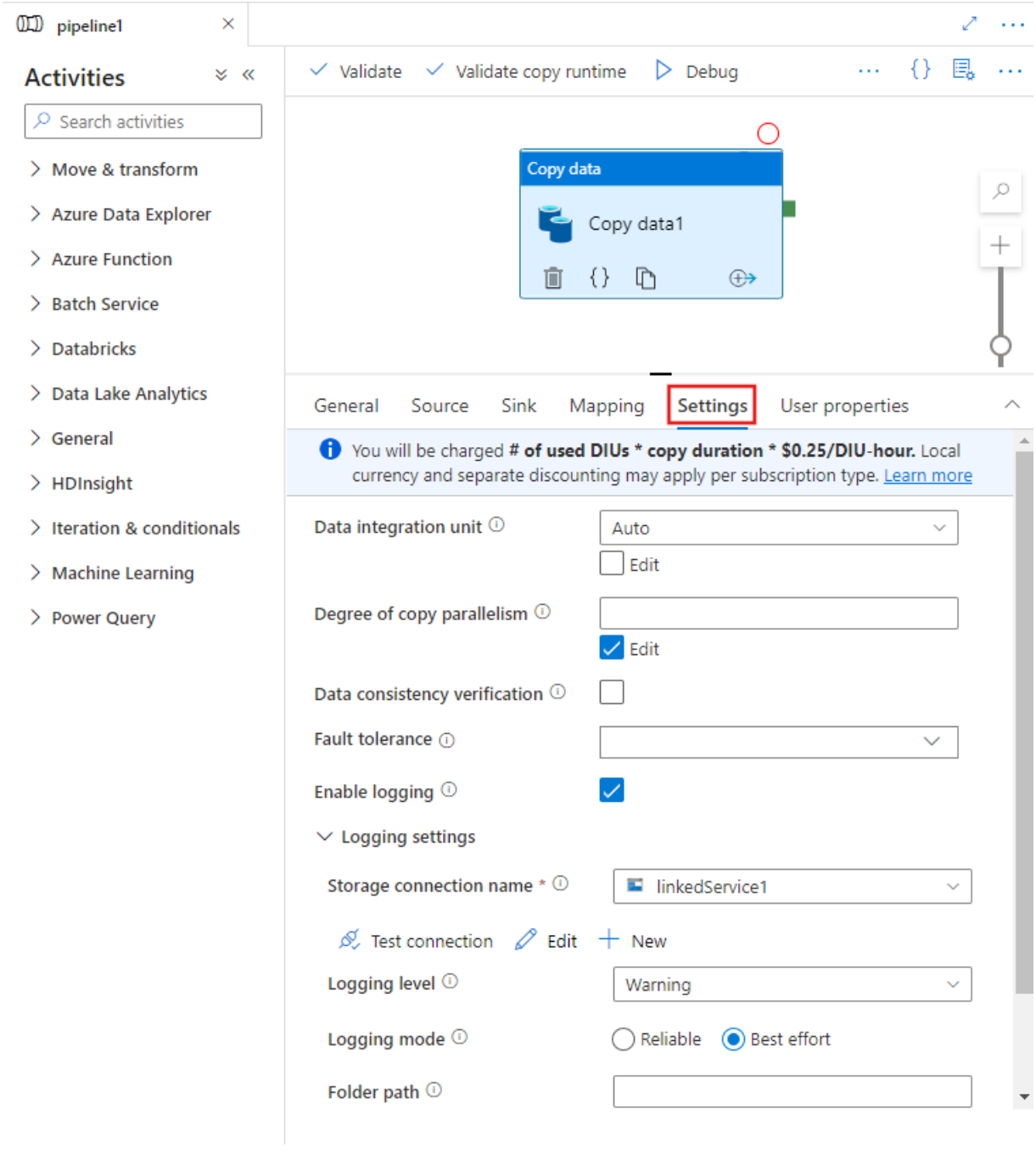
8/19/2022 • 11 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines the copy activity performance optimization features that you can leverage in Azure Data Factory and Synapse pipelines.

Configuring performance features with UI

When you select a Copy activity on the pipeline editor canvas and choose the Settings tab in the activity configuration area below the canvas, you will see options to configure all of the performance features detailed below.



Data Integration Units

A Data Integration Unit is a measure that represents the power (a combination of CPU, memory, and network resource allocation) of a single unit within the service. Data Integration Unit only applies to Azure integration runtime, but not self-hosted integration runtime.

The allowed DIUs to empower a copy activity run is **between 2 and 256**. If not specified or you choose "Auto" on the UI, the service dynamically applies the optimal DIU setting based on your source-sink pair and data pattern. The following table lists the supported DIU ranges and default behavior in different copy scenarios:

COPY SCENARIO	SUPPORTED DIU RANGE	DEFAULT DIUS DETERMINED BY SERVICE
Between file stores	- **Copy from or to single file**: 2-4 - **Copy from and to multiple files**: 2-256 depending on the number and size of the files For example, if you copy data from a folder with 4 large files and choose to preserve hierarchy, the max effective DIU is 16; when you choose to merge file, the max effective DIU is 4.	Between 4 and 32 depending on the number and size of the files
From file store to non-file store	- **Copy from single file**: 2-4 - **Copy from multiple files**: 2-256 depending on the number and size of the files For example, if you copy data from a folder with 4 large files, the max effective DIU is 16.	- **Copy into Azure SQL Database or Azure Cosmos DB**: between 4 and 16 depending on the sink tier (DTUs/RUs) and source file pattern - **Copy into Azure Synapse Analytics** using PolyBase or COPY statement: 2 - Other scenario: 4
From non-file store to file store	- **Copy from partition-option-enabled data stores** (including Azure Database for PostgreSQL, Azure SQL Database, Azure SQL Managed Instance, Azure Synapse Analytics, Oracle, Netezza, SQL Server, and Teradata): 2-256 when writing to a folder, and 2-4 when writing to one single file. Note per source data partition can use up to 4 DIUs. - **Other scenarios**: 2-4	- **Copy from REST or HTTP**: 1 - **Copy from Amazon Redshift** using UNLOAD: 2 - **Other scenario**: 4
Between non-file stores	- **Copy from partition-option-enabled data stores** (including Azure Database for PostgreSQL, Azure SQL Database, Azure SQL Managed Instance, Azure Synapse Analytics, Oracle, Netezza, SQL Server, and Teradata): 2-256 when writing to a folder, and 2-4 when writing to one single file. Note per source data partition can use up to 4 DIUs. - **Other scenarios**: 2-4	- **Copy from REST or HTTP**: 1 - **Other scenario**: 4

You can see the DIUs used for each copy run in the copy activity monitoring view or activity output. For more information, see Copy activity monitoring. To override this default, specify a value for the `dataIntegrationUnits` property as follows. The *actual number of DIUs* that the copy operation uses at run time is equal to or less than

the configured value, depending on your data pattern.

You will be charged **# of used DIUs * copy duration * unit price/DIU-hour**. See the current prices here. Local currency and separate discounting may apply per subscription type.

Example:

```
"activities":[
    {
        "name": "Sample copy activity",
        "type": "Copy",
        "inputs": [...],
        "outputs": [...],
        "typeProperties": {
            "source": {
                "type": "BlobSource",
            },
            "sink": {
                "type": "AzureDataLakeStoreSink"
            },
            "dataIntegrationUnits": 128
        }
    }
]
```

Self-hosted integration runtime scalability

If you would like to achieve higher throughput, you can either scale up or scale out the Self-hosted IR:

- If the CPU and available memory on the Self-hosted IR node are not fully utilized, but the execution of concurrent jobs is reaching the limit, you should scale up by increasing the number of concurrent jobs that can run on a node. See here for instructions.
- If on the other hand, the CPU is high on the Self-hosted IR node or available memory is low, you can add a new node to help scale out the load across the multiple nodes. See here for instructions.

Note in the following scenarios, single copy activity execution can leverage multiple Self-hosted IR nodes:

- Copy data from file-based stores, depending on the number and size of the files.
- Copy data from partition-option-enabled data store (including Azure SQL Database, Azure SQL Managed Instance, Azure Synapse Analytics, Oracle, Netezza, SAP HANA, SAP Open Hub, SAP Table, SQL Server, and Teradata), depending on the number of data partitions.

Parallel copy

You can set parallel copy (`parallelCopies` property) on copy activity to indicate the parallelism that you want the copy activity to use. You can think of this property as the maximum number of threads within the copy activity that read from your source or write to your sink data stores in parallel.

The parallel copy is orthogonal to Data Integration Units or Self-hosted IR nodes. It is counted across all the DIUs or Self-hosted IR nodes.

For each copy activity run, by default the service dynamically applies the optimal parallel copy setting based on your source-sink pair and data pattern.

TIP

The default behavior of parallel copy usually gives you the best throughput, which is auto-determined by the service based on your source-sink pair, data pattern and number of DIUs or the Self-hosted IR's CPU/memory/node count. Refer to Troubleshoot copy activity performance on when to tune parallel copy.

The following table lists the parallel copy behavior:

COPY SCENARIO	PARALLEL COPY BEHAVIOR
Between file stores	`parallelCopies` determines the parallelism **at the file level**. The chunking within each file happens underneath automatically and transparently. It's designed to use the best suitable chunk size for a given data store type to load data in parallel. The actual number of parallel copies copy activity uses at run time is no more than the number of files you have. If the copy behavior is **mergeFile** into file sink, the copy activity can't take advantage of file-level parallelism.
From file store to non-file store	- When copying data into Azure SQL Database or Azure Cosmos DB, default parallel copy also depend on the sink tier (number of DTUs/RUs). - When copying data into Azure Table, default parallel copy is 4.
From non-file store to file store	- When copying data from partition-option-enabled data store (including Azure SQL Database, Azure SQL Managed Instance, Azure Synapse Analytics, Oracle, Amazon RDS for Oracle, Netezza, SAP HANA, SAP Open Hub, SAP Table, SQL Server, Amazon RDS for SQL Server and Teradata), default parallel copy is 4. The actual number of parallel copies copy activity uses at run time is no more than the number of data partitions you have. When use Self-hosted Integration Runtime and copy to Azure Blob/ADLS Gen2, note the max effective parallel copy is 4 or 5 per IR node. - For other scenarios, parallel copy doesn't take effect. Even if parallelism is specified, it's not applied.
Between non-file stores	- When copying data into Azure SQL Database or Azure Cosmos DB, default parallel copy also depend on the sink tier (number of DTUs/RUs). - When copying data from partition-option-enabled data store (including Azure SQL Database, Azure SQL Managed Instance, Azure Synapse Analytics, Oracle, Amazon RDS for Oracle, Netezza, SAP HANA, SAP Open Hub, SAP Table, SQL Server, Amazon RDS for SQL Server and Teradata), default parallel copy is 4. - When copying data into Azure Table, default parallel copy is 4.

To control the load on machines that host your data stores, or to tune copy performance, you can override the default value and specify a value for the `parallelCopies` property. The value must be an integer greater than or equal to 1. At run time, for the best performance, the copy activity uses a value that is less than or equal to the value that you set.

When you specify a value for the `parallelCopies` property, take the load increase on your source and sink data stores into account. Also consider the load increase to the self-hosted integration runtime if the copy activity is

empowered by it. This load increase happens especially when you have multiple activities or concurrent runs of the same activities that run against the same data store. If you notice that either the data store or the self-hosted integration runtime is overwhelmed with the load, decrease the `parallelCopies` value to relieve the load.

Example:

```
"activities":[
    {
        "name": "Sample copy activity",
        "type": "Copy",
        "inputs": [...],
        "outputs": [...],
        "typeProperties": {
            "source": {
                "type": "BlobSource",
            },
            "sink": {
                "type": "AzureDataLakeStoreSink"
            },
            "parallelCopies": 32
        }
    }
]
```

Staged copy

When you copy data from a source data store to a sink data store, you might choose to use Azure Blob storage or Azure Data Lake Storage Gen2 as an interim staging store. Staging is especially useful in the following cases:

- **You want to ingest data from various data stores into Azure Synapse Analytics via PolyBase, copy data from/to Snowflake, or ingest data from Amazon Redshift/HDFS performantly.** Learn more details from:
  - Use PolyBase to load data into Azure Synapse Analytics.
  - Snowflake connector
  - Amazon Redshift connector
  - HDFS connector
- **You don't want to open ports other than port 80 and port 443 in your firewall because of corporate IT policies.** For example, when you copy data from an on-premises data store to an Azure SQL Database or an Azure Synapse Analytics, you need to activate outbound TCP communication on port 1433 for both the Windows firewall and your corporate firewall. In this scenario, staged copy can take advantage of the self-hosted integration runtime to first copy data to a staging storage over HTTP or HTTPS on port 443, then load the data from staging into SQL Database or Azure Synapse Analytics. In this flow, you don't need to enable port 1433.
- **Sometimes it takes a while to perform a hybrid data movement (that is, to copy from an on-premises data store to a cloud data store) over a slow network connection.** To improve performance, you can use staged copy to compress the data on-premises so that it takes less time to move data to the staging data store in the cloud. Then you can decompress the data in the staging store before you load into the destination data store.

How staged copy works

When you activate the staging feature, first the data is copied from the source data store to the staging storage (bring your own Azure Blob or Azure Data Lake Storage Gen2). Next, the data is copied from the staging to the sink data store. The copy activity automatically manages the two-stage flow for you, and also cleans up temporary data from the staging storage after the data movement is complete.

When you activate data movement by using a staging store, you can specify whether you want the data to be compressed before you move data from the source data store to the staging store and then decompressed before you move data from an interim or staging data store to the sink data store.

Currently, you can't copy data between two data stores that are connected via different Self-hosted IRs, neither with nor without staged copy. For such scenario, you can configure two explicitly chained copy activities to copy from source to staging then from staging to sink.

Configuration

Configure the **enableStaging** setting in the copy activity to specify whether you want the data to be staged in storage before you load it into a destination data store. When you set **enableStaging** to `TRUE`, specify the additional properties listed in the following table.

PROPERTY	DESCRIPTION	DEFAULT VALUE	REQUIRED
enableStaging	Specify whether you want to copy data via an interim staging store.	False	No
linkedServiceName	Specify the name of an Azure Blob storage or Azure Data Lake Storage Gen2 linked service, which refers to the instance of Storage that you use as an interim staging store.	N/A	Yes, when **enableStaging** is set to TRUE
path	Specify the path that you want to contain the staged data. If you don't provide a path, the service creates a container to store temporary data.	N/A	No
enableCompression	Specifies whether data should be compressed before it's copied to the destination. This setting reduces the volume of data being transferred.	False	No

> **NOTE**
> If you use staged copy with compression enabled, the service principal or MSI authentication for staging blob linked service isn't supported.

Here's a sample definition of a copy activity with the properties that are described in the preceding table:

```
"activities":[
    {
        "name": "CopyActivityWithStaging",
        "type": "Copy",
        "inputs": [...],
        "outputs": [...],
        "typeProperties": {
            "source": {
                "type": "OracleSource",
            },
            "sink": {
                "type": "SqlDWSink"
            },
            "enableStaging": true,
            "stagingSettings": {
                "linkedServiceName": {
                    "referenceName": "MyStagingStorage",
                    "type": "LinkedServiceReference"
                },
                "path": "stagingcontainer/path"
            }
        }
    }
]
```

Staged copy billing impact

You're charged based on two steps: copy duration and copy type.

- When you use staging during a cloud copy, which is copying data from a cloud data store to another cloud data store, both stages empowered by Azure integration runtime, you're charged the [sum of copy duration for step 1 and step 2] x [cloud copy unit price].
- When you use staging during a hybrid copy, which is copying data from an on-premises data store to a cloud data store, one stage empowered by a self-hosted integration runtime, you're charged for [hybrid copy duration] x [hybrid copy unit price] + [cloud copy duration] x [cloud copy unit price].

Next steps

See the other copy activity articles:

- Copy activity overview
- Copy activity performance and scalability guide
- Troubleshoot copy activity performance
- Use Azure Data Factory to migrate data from your data lake or data warehouse to Azure
- Migrate data from Amazon S3 to Azure Storage

Preserve metadata and ACLs using copy activity in Azure Data Factory or Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

When you use Azure Data Factory or Synapse Analytics pipelines copy activity to copy data from source to sink, in the following scenarios, you can also preserve the metadata and ACLs along.

Preserve metadata for lake migration

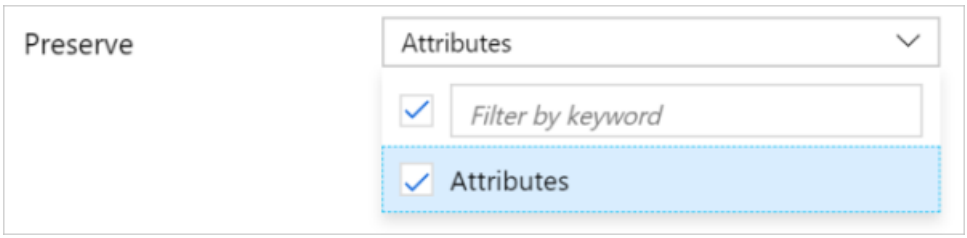
When you migrate data from one data lake to another including Amazon S3, Azure Blob, Azure Data Lake Storage Gen2, and Azure Files, you can choose to preserve the file metadata along with data.

Copy activity supports preserving the following attributes during data copy:

- **All the customer specified metadata**
- And the following **five data store built-in system properties**: `contentType`, `contentLanguage` (except for Amazon S3), `contentEncoding`, `contentDisposition`, `cacheControl`.

Handle differences in metadata: Amazon S3 and Azure Storage allow different sets of characters in the keys of customer specified metadata. When you choose to preserve metadata using copy activity, the service automatically replaces the invalid characters with '_'.

When you copy files as-is from Amazon S3/Azure Data Lake Storage Gen2/Azure Blob storage/Azure Files to Azure Data Lake Storage Gen2/Azure Blob storage/Azure Files with binary format, you can find the **Preserve** option on the **Copy Activity** > **Settings** tab for activity authoring or the **Settings** page in Copy Data Tool.



Here's an example of copy activity JSON configuration (see `preserve`):

```
"activities":[
    {
        "name": "CopyAndPreserveMetadata",
        "type": "Copy",
        "typeProperties": {
            "source": {
                "type": "BinarySource",
                "storeSettings": {
                    "type": "AmazonS3ReadSettings",
                    "recursive": true
                }
            },
            "sink": {
                "type": "BinarySink",
                "storeSettings": {
                    "type": "AzureBlobFSWriteSettings"
                }
            },
            "preserve": [
                "Attributes"
            ]
        },
        "inputs": [
            {
                "referenceName": "<Binary dataset Amazon S3/Azure Blob/ADLS Gen2 source>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Binary dataset for Azure Blob/ADLS Gen2 sink>",
                "type": "DatasetReference"
            }
        ]
    }
]
```

Preserve ACLs from Data Lake Storage Gen1/Gen2 to Gen2

When you upgrade from Azure Data Lake Storage Gen1 to Gen2 or copy data between ADLS Gen2, you can choose to preserve the POSIX access control lists (ACLs) along with data files. For more information on access control, see Access control in Azure Data Lake Storage Gen1 and Access control in Azure Data Lake Storage Gen2.

Copy activity supports preserving the following types of ACLs during data copy. You can select one or more types:

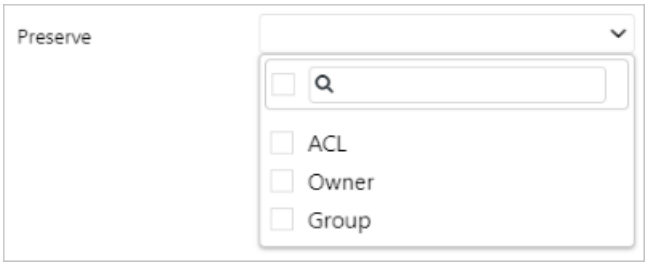
- **ACL**: Copy and preserve POSIX access control lists on files and directories. It copies the full existing ACLs from source to sink.
- **Owner**: Copy and preserve the owning user of files and directories. Super-user access to sink Data Lake Storage Gen2 is required.
- **Group**: Copy and preserve the owning group of files and directories. Super-user access to sink Data Lake Storage Gen2 or the owning user (if the owning user is also a member of the target group) is required.

If you specify to copy from a folder, the service replicates the ACLs for that given folder and the files and directories under it, if `recursive` is set to true. If you specify to copy from a single file, the ACLs on that file are copied.

> **NOTE**
>
> When you use the copy activity to preserve ACLs from Data Lake Storage Gen1/Gen2 to Gen2, the existing ACLs on sink Gen2's corresponding folder/files will be overwritten.

> **IMPORTANT**
>
> When you choose to preserve ACLs, make sure you grant high enough permissions for the service to operate against your sink Data Lake Storage Gen2 account. For example, use account key authentication or assign the Storage Blob Data Owner role to the service principal or managed identity.

When you configure source as Data Lake Storage Gen1/Gen2 with binary format or the binary copy option, and sink as Data Lake Storage Gen2 with binary format or the binary copy option, you can find the **Preserve** option on the **Settings** page in Copy Data Tool or on the **Copy Activity** > **Settings** tab for activity authoring.



Here's an example of copy activity JSON configuration (see `preserve`):

```
"activities":[
    {
        "name": "CopyAndPreserveACLs",
        "type": "Copy",
        "typeProperties": {
            "source": {
                "type": "BinarySource",
                "storeSettings": {
                    "type": "AzureDataLakeStoreReadSettings",
                    "recursive": true
                }
            },
            "sink": {
                "type": "BinarySink",
                "storeSettings": {
                    "type": "AzureBlobFSWriteSettings"
                }
            },
            "preserve": [
                "ACL",
                "Owner",
                "Group"
            ]
        },
        "inputs": [
            {
                "referenceName": "<Binary dataset name for Azure Data Lake Storage Gen1/Gen2 source>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Binary dataset name for Azure Data Lake Storage Gen2 sink>",
                "type": "DatasetReference"
            }
        ]
    }
]
```

Next steps

See the other Copy Activity articles:

- Copy activity overview
- Copy activity performance

Schema and data type mapping in copy activity

8/19/2022 • 13 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article describes how the Azure Data Factory copy activity perform schema mapping and data type mapping from source data to sink data.

Schema mapping

Default mapping

By default, copy activity maps source data to sink **by column names** in case-sensitive manner. If sink doesn't exist, for example, writing to file(s), the source field names will be persisted as sink names. If the sink already exists, it must contain all columns being copied from the source. Such default mapping supports flexible schemas and schema drift from source to sink from execution to execution - all the data returned by source data store can be copied to sink.

If your source is text file without header line, explicit mapping is required as the source doesn't contain column names.

Explicit mapping

You can also specify explicit mapping to customize the column/field mapping from source to sink based on your need. With explicit mapping, you can copy only partial source data to sink, or map source data to sink with different names, or reshape tabular/hierarchical data. Copy activity:

1. Reads the data from source and determine the source schema.
2. Applies your defined mapping.
3. Writes the data to sink.

Learn more about:

- Tabular source to tabular sink
- Hierarchical source to tabular sink
- Tabular/Hierarchical source to hierarchical sink

You can configure the mapping on the Authoring UI -> copy activity -> mapping tab, or programmatically specify the mapping in copy activity -> `translator` property. The following properties are supported in `translator` -> `mappings` array -> objects -> `source` and `sink`, which points to the specific column/field to map data.

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the source or sink column/field. Apply for tabular source and sink.	Yes
ordinal	Column index. Start from 1. Apply and required when using delimited text without header line.	No

PROPERTY	DESCRIPTION	REQUIRED
path	JSON path expression for each field to extract or map. Apply for hierarchical source and sink, for example, Cosmos DB, MongoDB, or REST connectors. For fields under the root object, the JSON path starts with root `$`; for fields inside the array chosen by `collectionReference` property, JSON path starts from the array element without `$`.	No
type	Interim data type of the source or sink column. In general, you don't need to specify or change this property. Learn more about data type mapping.	No
culture	Culture of the source or sink column. Apply when type is `Datetime` or `Datetimeoffset`. The default is `en-us`. In general, you don't need to specify or change this property. Learn more about data type mapping.	No
format	Format string to be used when type is `Datetime` or `Datetimeoffset`. Refer to Custom Date and Time Format Strings on how to format datetime. In general, you don't need to specify or change this property. Learn more about data type mapping.	No

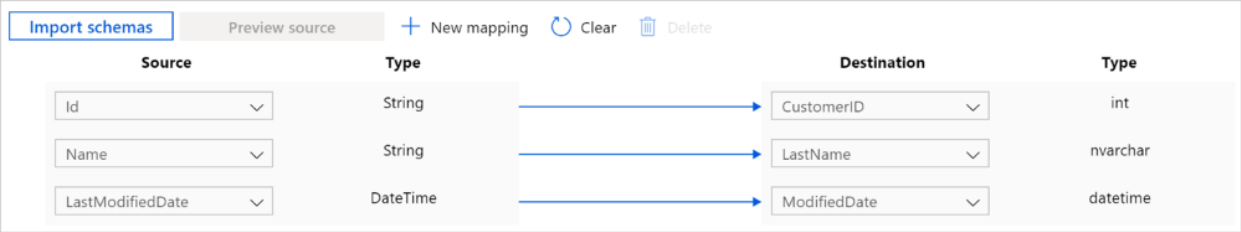
The following properties are supported under `translator` in addition to `mappings`:

PROPERTY	DESCRIPTION	REQUIRED
collectionReference	Apply when copying data from hierarchical source, for example, Cosmos DB, MongoDB, or REST connectors. If you want to iterate and extract data from the objects **inside an array field** with the same pattern and convert to per row per object, specify the JSON path of that array to do cross-apply.	No

Tabular source to tabular sink

For example, to copy data from Salesforce to Azure SQL Database and explicitly map three columns:

1. On copy activity -> mapping tab, click **Import schemas** button to import both source and sink schemas.
2. Map the needed fields and exclude/delete the rest.



The same mapping can be configured as the following in copy activity payload (see `translator`):

```
{
    "name": "CopyActivityTabularToTabular",
    "type": "Copy",
    "typeProperties": {
        "source": { "type": "SalesforceSource" },
        "sink": { "type": "SqlSink" },
        "translator": {
            "type": "TabularTranslator",
            "mappings": [
                {
                    "source": { "name": "Id" },
                    "sink": { "name": "CustomerID" }
                },
                {
                    "source": { "name": "Name" },
                    "sink": { "name": "LastName" }
                },
                {
                    "source": { "name": "LastModifiedDate" },
                    "sink": { "name": "ModifiedDate" }
                }
            ]
        }
    },
    ...
}
```

To copy data from delimited text file(s) without header line, the columns are represented by ordinal instead of names.

```
{
    "name": "CopyActivityTabularToTabular",
    "type": "Copy",
    "typeProperties": {
        "source": { "type": "DelimitedTextSource" },
        "sink": { "type": "SqlSink" },
        "translator": {
            "type": "TabularTranslator",
            "mappings": [
                {
                    "source": { "ordinal": "1" },
                    "sink": { "name": "CustomerID" }
                },
                {
                    "source": { "ordinal": "2" },
                    "sink": { "name": "LastName" }
                },
                {
                    "source": { "ordinal": "3" },
                    "sink": { "name": "ModifiedDate" }
                }
            ]
        }
    },
    ...
}
```

Hierarchical source to tabular sink

When copying data from hierarchical source to tabular sink, copy activity supports the following capabilities:

- Extract data from objects and arrays.
- Cross apply multiple objects with the same pattern from an array, in which case to convert one JSON object into multiple records in tabular result.

For more advanced hierarchical-to-tabular transformation, you can use Data Flow.

For example, if you have source MongoDB document with the following content:

```
{
    "id": {
        "$oid": "592e07800000000000000000"
    },
    "number": "01",
    "date": "20170122",
    "orders": [
        {
            "prod": "p1",
            "price": 23
        },
        {
            "prod": "p2",
            "price": 13
        },
        {
            "prod": "p3",
            "price": 231
        }
    ],
    "city": [ { "name": "Seattle" } ]
}
```

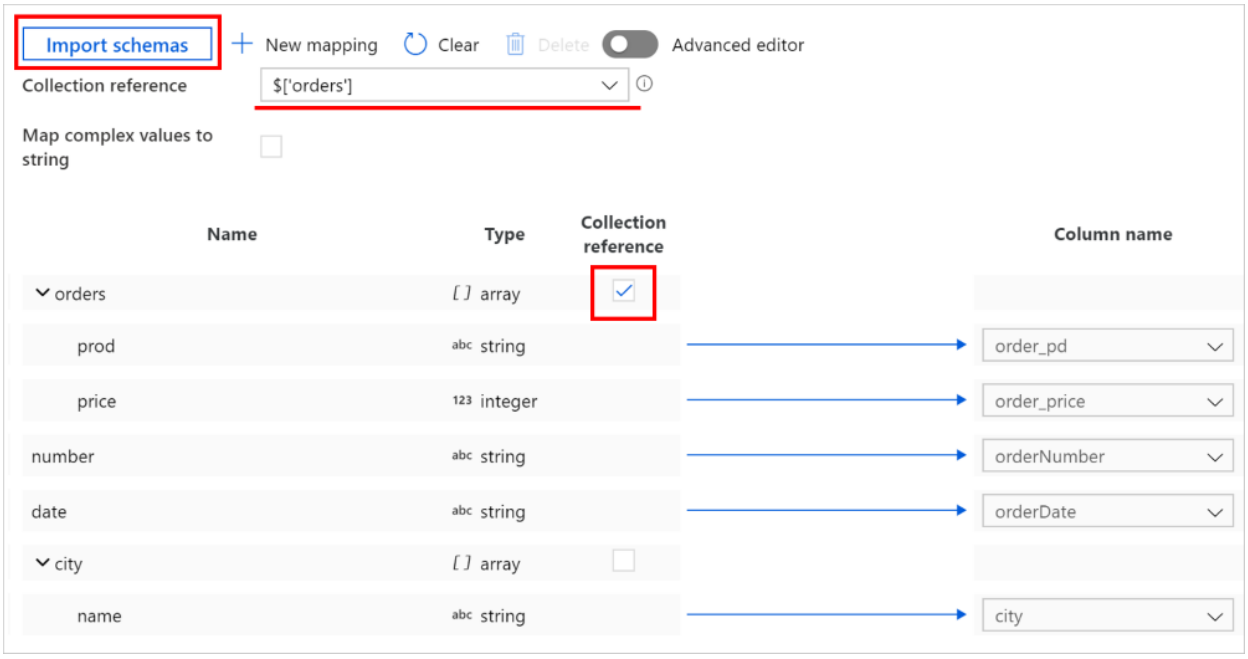
And you want to copy it into a text file in the following format with header line, by flattening the data inside the array *(order_pd and order_price)* and cross join with the common root info *(number, date, and city)*:

ORDERNUMBER	ORDERDATE	ORDER_PD	ORDER_PRICE	CITY
01	20170122	P1	23	Seattle
01	20170122	P2	13	Seattle
01	20170122	P3	231	Seattle

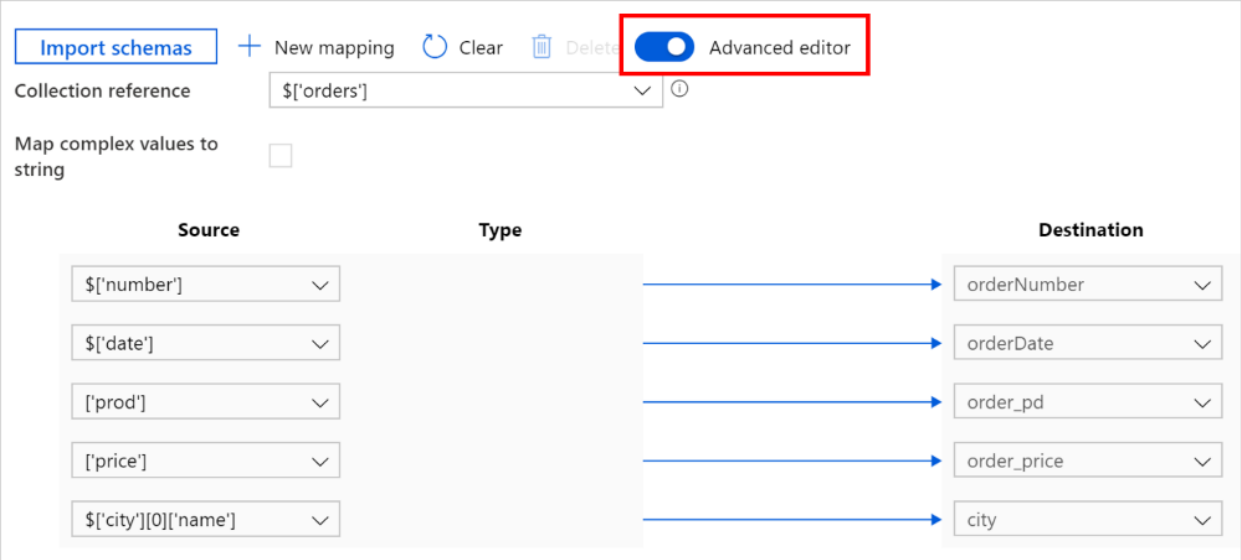
You can define such mapping on Data Factory authoring UI:

1. On copy activity -> mapping tab, click **Import schemas** button to import both source and sink schemas. As the service samples the top few objects when importing schema, if any field doesn't show up, you can add it to the correct layer in the hierarchy - hover on an existing field name and choose to add a node, an object, or an array.
2. Select the array from which you want to iterate and extract data. It will be auto populated as **Collection reference**. Note only single array is supported for such operation.
3. Map the needed fields to sink. The service automatically determines the corresponding JSON paths for the hierarchical side.

> **NOTE**
> For records where the array marked as collection reference is empty and the check box is selected, the entire record is skipped.



You can also switch to **Advanced editor**, in which case you can directly see and edit the fields' JSON paths. If you choose to add new mapping in this view, specify the JSON path.



The same mapping can be configured as the following in copy activity payload (see `translator`):

```
{
    "name": "CopyActivityHierarchicalToTabular",
    "type": "Copy",
    "typeProperties": {
        "source": { "type": "MongoDbV2Source" },
        "sink": { "type": "DelimitedTextSink" },
        "translator": {
            "type": "TabularTranslator",
            "mappings": [
                {
                    "source": { "path": "$['number']" },
                    "sink": { "name": "orderNumber" }
                },
                {
                    "source": { "path": "$['date']" },
                    "sink": { "name": "orderDate" }
                },
                {
                    "source": { "path": "['prod']" },
                    "sink": { "name": "order_pd" }
                },
                {
                    "source": { "path": "['price']" },
                    "sink": { "name": "order_price" }
                },
                {
                    "source": { "path": "$['city'][0]['name']" },
                    "sink": { "name": "city" }
                }
            ],
            "collectionReference": "$['orders']"
        }
    },
    ...
}
```

Tabular/Hierarchical source to hierarchical sink

The user experience flow is similar to Hierarchical source to tabular sink.

When copying data from tabular source to hierarchical sink, writing to array inside object is not supported.

When copying data from hierarchical source to hierarchical sink, you can additionally preserve entire layer's hierarchy, by selecting the object/array and map to sink without touching the inner fields.

For more advanced data reshape transformation, you can use Data Flow.

Parameterize mapping

If you want to create a templatized pipeline to copy large number of objects dynamically, determine whether you can leverage the default mapping or you need to define explicit mapping for respective objects.

If explicit mapping is needed, you can:

1. Define a parameter with object type at the pipeline level, for example, `mapping`.
2. Parameterize the mapping: on copy activity -> mapping tab, choose to add dynamic content and select the above parameter. The activity payload would be as the following:

```
{
    "name": "CopyActivityHierarchicalToTabular",
    "type": "Copy",
    "typeProperties": {
        "source": {...},
        "sink": {...},
        "translator": {
            "value": "@pipeline().parameters.mapping",
            "type": "Expression"
        },
        ...
    }
}
```

3. Construct the value to pass into the mapping parameter. It should be the entire object of `translator` definition, refer to the samples in explicit mapping section. For example, for tabular source to tabular sink copy, the value should be

```
{"type":"TabularTranslator","mappings":[{"source":{"name":"Id"},"sink":{"name":"CustomerID"}},
{"source":{"name":"Name"},"sink":{"name":"LastName"}},{"source":{"name":"LastModifiedDate"},"sink":
{"name":"ModifiedDate"}}]}
```

.

Data type mapping

Copy activity performs source types to sink types mapping with the following flow:

1. Convert from source native data types to interim data types used by Azure Data Factory and Synapse pipelines.
2. Automatically convert interim data type as needed to match corresponding sink types, applicable for both default mapping and explicit mapping.
3. Convert from interim data types to sink native data types.

Copy activity currently supports the following interim data types: Boolean, Byte, Byte array, Datetime, DatetimeOffset, Decimal, Double, GUID, Int16, Int32, Int64, SByte, Single, String, Timespan, UInt16, UInt32, and UInt64.

The following data type conversions are supported between the interim types from source to sink.

SOURCE \SINK	BOOLEAN	BYTE ARRAY	DECIMAL	DATE/TIME (1)	FLOAT-POINT (2)	GUID	INTEGER (3)	STRING	TIMESPAN
Boolean	✓		✓		✓		✓	✓	

SOURCE \SINK	BOOLEAN	BYTE ARRAY	DECIMAL	DATE/TIME (1)	FLOAT-POINT (2)	GUID	INTEGER (3)	STRING	TIMESPAN
Byte array		✓						✓	
Date/Time				✓				✓	
Decimal	✓		✓		✓		✓	✓	
Float-point	✓		✓		✓		✓	✓	
GUID						✓		✓	
Integer	✓		✓		✓		✓	✓	
String	✓	✓	✓	✓	✓	✓	✓	✓	✓
TimeSpan								✓	✓

(1) Date/Time includes DateTime and DateTimeOffset.

(2) Float-point includes Single and Double.

(3) Integer includes SByte, Byte, Int16, UInt16, Int32, UInt32, Int64, and UInt64.

> **NOTE**
>
> - Currently such data type conversion is supported when copying between tabular data. Hierarchical sources/sinks are not supported, which means there is no system-defined data type conversion between source and sink interim types.
> - This feature works with the latest dataset model. If you don't see this option from the UI, try creating a new dataset.

The following properties are supported in copy activity for data type conversion (under `translator` section for programmatical authoring):

PROPERTY	DESCRIPTION	REQUIRED

PROPERTY	DESCRIPTION	REQUIRED
typeConversion	Enable the new data type conversion experience. Default value is false due to backward compatibility. For new copy activities created via Data Factory authoring UI since late June 2020, this data type conversion is enabled by default for the best experience, and you can see the following type conversion settings on copy activity -> mapping tab for applicable scenarios. To create pipeline programmatically, you need to explicitly set `typeConversion` property to true to enable it. For existing copy activities created before this feature is released, you won't see type conversion options on the authoring UI for backward compatibility.	No
typeConversionSettings	A group of type conversion settings. Apply when `typeConversion` is set to `true`. The following properties are all under this group.	No
Under `typeConversionSettings`		
allowDataTruncation	Allow data truncation when converting source data to sink with different type during copy, for example, from decimal to integer, from DatetimeOffset to Datetime. Default value is true.	No
treatBooleanAsNumber	Treat booleans as numbers, for example, true as 1. Default value is false.	No
dateTimeFormat	Format string when converting between dates without time zone offset and strings, for example, `yyyy-MM-dd HH:mm:ss.fff`. Refer to Custom Date and Time Format Strings for detailed information.	No
dateTimeOffsetFormat	Format string when converting between dates with time zone offset and strings, for example, `yyyy-MM-dd HH:mm:ss.fff zzz`. Refer to Custom Date and Time Format Strings for detailed information.	No

PROPERTY	DESCRIPTION	REQUIRED
timeSpanFormat	Format string when converting between time periods and strings, for example, `dd\.hh\:mm`. Refer to Custom TimeSpan Format Strings for detailed information.	No
culture	Culture information to be used when convert types, for example, `en-us` or `fr-fr`.	No

Example:

```
{
    "name": "CopyActivity",
    "type": "Copy",
    "typeProperties": {
        "source": {
         "type": "ParquetSource"
        },
        "sink": {
            "type": "SqlSink"
        },
        "translator": {
            "type": "TabularTranslator",
            "typeConversion": true,
            "typeConversionSettings": {
                "allowDataTruncation": true,
                "treatBooleanAsNumber": true,
                "dateTimeFormat": "yyyy-MM-dd HH:mm:ss.fff",
                "dateTimeOffsetFormat": "yyyy-MM-dd HH:mm:ss.fff zzz",
                "timeSpanFormat": "dd\.hh\:mm",
                "culture": "en-gb"
            }
        }
 },
    ...
}
```

Legacy models

> **NOTE**
> The following models to map source columns/fields to sink are still supported as is for backward compatibility. We suggest that you use the new model mentioned in schema mapping. The authoring UI has switched to generating the new model.

Alternative column-mapping (legacy model)

You can specify copy activity -> `translator` -> `columnMappings` to map between tabular-shaped data. In this case, the "structure" section is required for both input and output datasets. Column mapping supports **mapping all or subset of columns in the source dataset "structure" to all columns in the sink dataset "structure"**. The following are error conditions that result in an exception:

- Source data store query result does not have a column name that is specified in the input dataset "structure" section.
- Sink data store (if with pre-defined schema) does not have a column name that is specified in the output dataset "structure" section.

- Either fewer columns or more columns in the "structure" of sink dataset than specified in the mapping.
- Duplicate mapping.

In the following example, the input dataset has a structure, and it points to a table in an on-premises Oracle database.

```
{
    "name": "OracleDataset",
    "properties": {
        "structure":
         [
            { "name": "UserId"},
            { "name": "Name"},
            { "name": "Group"}
         ],
        "type": "OracleTable",
        "linkedServiceName": {
            "referenceName": "OracleLinkedService",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "tableName": "SourceTable"
        }
    }
}
```

In this sample, the output dataset has a structure and it points to a table in Salesfoce.

```
{
    "name": "SalesforceDataset",
    "properties": {
        "structure":
        [
            { "name": "MyUserId"},
            { "name": "MyName" },
            { "name": "MyGroup"}
        ],
        "type": "SalesforceObject",
        "linkedServiceName": {
            "referenceName": "SalesforceLinkedService",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "tableName": "SinkTable"
        }
    }
}
```

The following JSON defines a copy activity in a pipeline. The columns from source mapped to columns in sink by using the **translator** -> **columnMappings** property.

```
{
    "name": "CopyActivity",
    "type": "Copy",
    "inputs": [
        {
            "referenceName": "OracleDataset",
            "type": "DatasetReference"
        }
    ],
    "outputs": [
        {
            "referenceName": "SalesforceDataset",
            "type": "DatasetReference"
        }
    ],
    "typeProperties":    {
        "source": { "type": "OracleSource" },
        "sink": { "type": "SalesforceSink" },
        "translator":
        {
            "type": "TabularTranslator",
            "columnMappings":
            {
                "UserId": "MyUserId",
                "Group": "MyGroup",
                "Name": "MyName"
            }
        }
    }
}
```

If you are using the syntax of `"columnMappings": "UserId: MyUserId, Group: MyGroup, Name: MyName"` to specify column mapping, it is still supported as-is.

Alternative schema-mapping (legacy model)

You can specify copy activity -> `translator` -> `schemaMapping` to map between hierarchical-shaped data and tabular-shaped data, for example, copy from MongoDB/REST to text file and copy from Oracle to Azure Cosmos DB's API for MongoDB. The following properties are supported in copy activity `translator` section:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity translator must be set to: **TabularTranslator**	Yes

PROPERTY	DESCRIPTION	REQUIRED
schemaMapping	A collection of key-value pairs, which represents the mapping relation **from source side to sink side**. - **Key:** represents source. For **tabular source**, specify the column name as defined in dataset structure; for **hierarchical source**, specify the JSON path expression for each field to extract and map. - **Value:** represents sink. For **tabular sink**, specify the column name as defined in dataset structure; for **hierarchical sink**, specify the JSON path expression for each field to extract and map. In the case of hierarchical data, for fields under root object, JSON path starts with root $; for fields inside the array chosen by `collectionReference` property, JSON path starts from the array element.	Yes
collectionReference	If you want to iterate and extract data from the objects **inside an array field** with the same pattern and convert to per row per object, specify the JSON path of that array to do cross-apply. This property is supported only when hierarchical data is source.	No

Example: copy from MongoDB to Oracle:

For example, if you have MongoDB document with the following content:

```
{
    "id": {
        "$oid": "592e07800000000000000000"
    },
    "number": "01",
    "date": "20170122",
    "orders": [
        {
            "prod": "p1",
            "price": 23
        },
        {
            "prod": "p2",
            "price": 13
        },
        {
            "prod": "p3",
            "price": 231
        }
    ],
    "city": [ { "name": "Seattle" } ]
}
```

and you want to copy it into an Azure SQL table in the following format, by flattening the data inside the array *(order_pd and order_price)* and cross join with the common root info *(number, date, and city)*:

ORDERNUMBER	ORDERDATE	ORDER_PD	ORDER_PRICE	CITY
01	20170122	P1	23	Seattle
01	20170122	P2	13	Seattle
01	20170122	P3	231	Seattle

Configure the schema-mapping rule as the following copy activity JSON sample:

```
{
    "name": "CopyFromMongoDBToOracle",
    "type": "Copy",
    "typeProperties": {
        "source": {
            "type": "MongoDbV2Source"
        },
        "sink": {
            "type": "OracleSink"
        },
        "translator": {
            "type": "TabularTranslator",
            "schemaMapping": {
                "$.number": "orderNumber",
                "$.date": "orderDate",
                "prod": "order_pd",
                "price": "order_price",
                "$.city[0].name": "city"
            },
            "collectionReference":  "$.orders"
        }
    }
}
```

Next steps

See the other Copy Activity articles:

- Copy activity overview

Fault tolerance of copy activity in Azure Data Factory and Synapse Analytics pipelines

8/19/2022 • 12 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

When you copy data from source to destination store, the copy activity provides certain level of fault tolerances to prevent interruption from failures in the middle of data movement. For example, you are copying millions of rows from source to destination store, where a primary key has been created in the destination database, but source database does not have any primary keys defined. When you happen to copy duplicated rows from source to the destination, you will hit the PK violation failure on the destination database. At this moment, copy activity offers you two ways to handle such errors:

- You can abort the copy activity once any failure is encountered.
- You can continue to copy the rest by enabling fault tolerance to skip the incompatible data. For example, skip the duplicated row in this case. In addition, you can log the skipped data by enabling session log within copy activity. You can refer to session log in copy activity for more details.

Copying binary files

The service supports the following fault tolerance scenarios when copying binary files. You can choose to abort the copy activity or continue to copy the rest in the following scenarios:

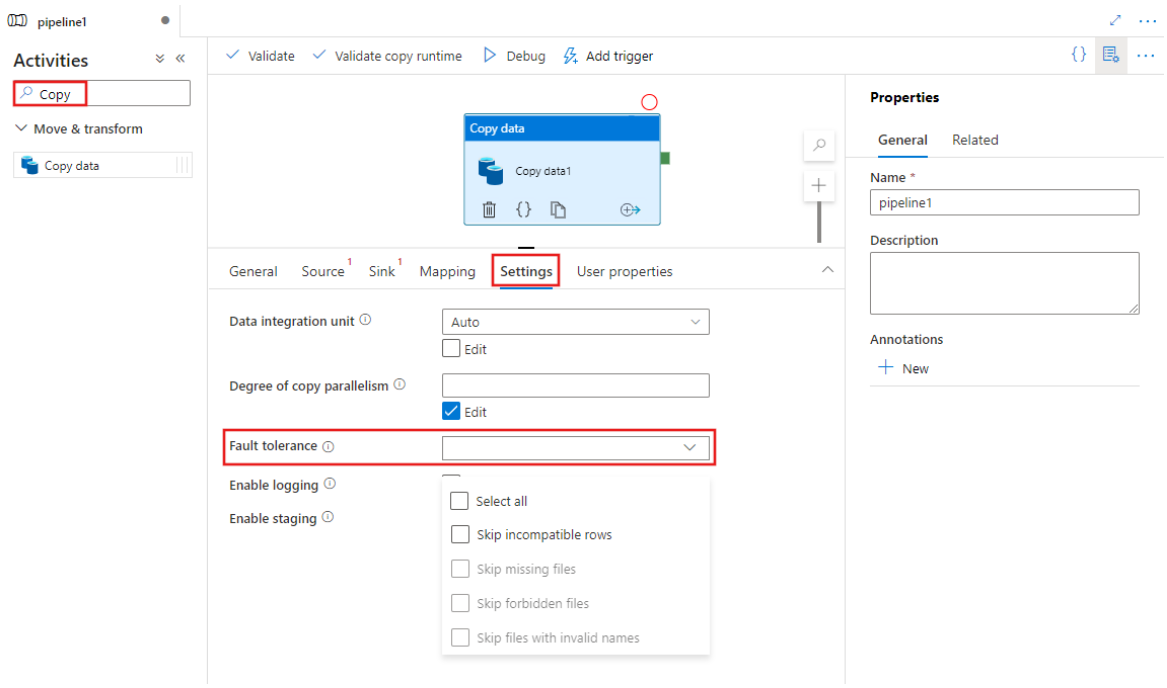
1. The files to be copied by the service are being deleted by other applications at the same time.
2. Some particular folders or files do not allow the service access because ACLs of those files or folders require higher permission level than the configured connection information.
3. One or more files are not verified to be consistent between source and destination store if you enable data consistency verification setting.

Enable fault tolerance with UI

To configure fault tolerance in a Copy activity in a pipeline with UI, complete the following steps:

1. If you did not create a Copy activity for your pipeline already, search for *Copy* in the pipeline Activities pane, and drag a Copy Data activity to the pipeline canvas.

2. Select the new Copy Data activity on the canvas if it is not already selected, and its **Settings** tab, to configure fault tolerance.



Configuration

When you copy binary files between storage stores, you can enable fault tolerance as followings:

```
"typeProperties": {
    "source": {
        "type": "BinarySource",
        "storeSettings": {
            "type": "AzureDataLakeStoreReadSettings",
            "recursive": true
            }
    },
    "sink": {
        "type": "BinarySink",
        "storeSettings": {
            "type": "AzureDataLakeStoreWriteSettings"
        }
    },
    "skipErrorFile": {
        "fileMissing": true,
        "fileForbidden": true,
        "dataInconsistency": true,
        "invalidFileName": true
    },
    "validateDataConsistency": true,
    "logSettings": {
        "enableCopyActivityLog": true,
        "copyActivityLogSettings": {
            "logLevel": "Warning",
            "enableReliableLogging": false
        },
        "logLocationSettings": {
            "linkedServiceName": {
               "referenceName": "ADLSGen2",
               "type": "LinkedServiceReference"
            },
            "path": "sessionlog/"
        }
    }
}
```

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
skipErrorFile	A group of properties to specify the types of failures you want to skip during the data movement.		No
fileMissing	One of the key-value pairs within skipErrorFile property bag to determine if you want to skip files that are being deleted by other applications at the time the service is performing the copy operation. -True: you want to copy the rest by skipping the files being deleted by other applications. - False: you want to abort the copy activity once any files are being deleted from source store in the middle of data movement. Be aware this property is set to true as default.	True(default) False	No
fileForbidden	One of the key-value pairs within skipErrorFile property bag to determine if you want to skip the particular files, when the ACLs of those files or folders require higher permission level than the configured connection. -True: you want to copy the rest by skipping the files. - False: you want to abort the copy activity once getting the permission issue on folders or files.	True False(default)	No
dataInconsistency	One of the key-value pairs within skipErrorFile property bag to determine if you want to skip the inconsistent data between source and destination store. -True: you want to copy the rest by skipping inconsistent data. - False: you want to abort the copy activity once inconsistent data found. Be aware this property is only valid when you set validateDataConsistency as True.	True False(default)	No

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
invalidFileName	One of the key-value pairs within skipErrorFile property bag to determine if you want to skip the particular files, when the file names are invalid for the destination store. -True: you want to copy the rest by skipping the files having invalid file names. - False: you want to abort the copy activity once any files have invalid file names. Be aware this property works when copying binary files from any storage store to ADLS Gen2 or copying binary files from AWS S3 to any storage store only.	True False(default)	No
logSettings	A group of properties that can be specified when you want to log the skipped object names.		No
linkedServiceName	The linked service of Azure Blob Storage or Azure Data Lake Storage Gen2 to store the session log files.	The names of an `AzureBlobStorage` or `AzureBlobFS` type linked service, which refers to the instance that you use to store the log file.	No
path	The path of the log files.	Specify the path that you use to store the log files. If you do not provide a path, the service creates a container for you.	No

NOTE

The followings are the prerequisites of enabling fault tolerance in copy activity when copying binary files. For skipping particular files when they are being deleted from source store:

- The source dataset and sink dataset have to be binary format, and the compression type cannot be specified.
- The supported data store types are Azure Blob storage, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure Files, File System, FTP, SFTP, Amazon S3, Google Cloud Storage and HDFS.
- Only if when you specify multiple files in source dataset, which can be a folder, wildcard or a list of files, copy activity can skip the particular error files. If a single file is specified in source dataset to be copied to the destination, copy activity will fail if any error occurred.

For skipping particular files when their access are forbidden from source store:

- The source dataset and sink dataset have to be binary format, and the compression type cannot be specified.
- The supported data store types are Azure Blob storage, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure Files, SFTP, Amazon S3 and HDFS.
- Only if when you specify multiple files in source dataset, which can be a folder, wildcard or a list of files, copy activity can skip the particular error files. If a single file is specified in source dataset to be copied to the destination, copy activity will fail if any error occurred.

For skipping particular files when they are verified to be inconsistent between source and destination store:

- You can get more details from data consistency doc here.

Monitoring

Output from copy activity

You can get the number of files being read, written, and skipped via the output of each copy activity run.

```
"output": {
            "dataRead": 695,
            "dataWritten": 186,
            "filesRead": 3,
            "filesWritten": 1,
            "filesSkipped": 2,
            "throughput": 297,
            "logFilePath": "myfolder/a84bf8d4-233f-4216-8cb5-45962831cd1b/",
            "dataConsistencyVerification":
           {
                "VerificationResult": "Verified",
                "InconsistentData": "Skipped"
           }
       }
```

Session log from copy activity

If you configure to log the skipped file names, you can find the log file from this path:

```
https://[your-blob-account].blob.core.windows.net/[path-if-configured]/copyactivity-logs/[copy-activity-name]/[copy-activity-run-id]/[auto-generated-GUID].csv
```

.

The log files have to be the csv files. The schema of the log file is as following:

COLUMN	DESCRIPTION
Timestamp	The timestamp when the file was skipped.

COLUMN	DESCRIPTION
Level	The log level of this item. It will be in 'Warning' level for the item showing file skipping.
OperationName	Copy activity operational behavior on each file. It will be 'FileSkip' to specify the file to be skipped.
OperationItem	The file names to be skipped.
Message	More information to illustrate why the file being skipped.

The example of a log file is as following:

```
Timestamp,Level,OperationName,OperationItem,Message
2020-03-24 05:35:41.0209942,Warning,FileSkip,"bigfile.csv","File is skipped after read 322961408 bytes:
ErrorCode=UserErrorSourceBlobNotExist,'Type=Microsoft.DataTransfer.Common.Shared.HybridDeliveryException,Mes
sage=The required Blob is missing. ContainerName:
https://transferserviceonebox.blob.core.windows.net/skipfaultyfile, path:
bigfile.csv.,Source=Microsoft.DataTransfer.ClientLibrary,'."
2020-03-24 05:38:41.2595989,Warning,FileSkip,"3_nopermission.txt","File is skipped after read 0 bytes:
ErrorCode=AdlsGen2OperationFailed,'Type=Microsoft.DataTransfer.Common.Shared.HybridDeliveryException,Message
=ADLS Gen2 operation failed for: Operation returned an invalid status code 'Forbidden'. Account:
'adlsgen2perfsource'. FileSystem: 'skipfaultyfilesforbidden'. Path: '3_nopermission.txt'. ErrorCode:
'AuthorizationPermissionMismatch'. Message: 'This request is not authorized to perform this operation using
this permission.'. RequestId: '35089f5d-101f-008c-489e-
01cce4000000'..,Source=Microsoft.DataTransfer.ClientLibrary,''Type=Microsoft.DataTransfer.Common.Shared.Hybr
idDeliveryException,Message=Operation returned an invalid status code
'Forbidden',Source=,''Type=Microsoft.Azure.Storage.Data.Models.ErrorSchemaException,Message='Type=Microsoft.
Azure.Storage.Data.Models.ErrorSchemaException,Message=Operation returned an invalid status code
'Forbidden',Source=Microsoft.DataTransfer.ClientLibrary,',Source=Microsoft.DataTransfer.ClientLibrary,'."
```

From the log above, you can see bigfile.csv has been skipped due to another application deleted this file when the service was copying it. And 3_nopermission.txt has been skipped because the service is not allowed to access it due to permission issue.

Copying tabular data

Supported scenarios

Copy activity supports three scenarios for detecting, skipping, and logging incompatible tabular data:

- **Incompatibility between the source data type and the sink native type**.

  For example: Copy data from a CSV file in Blob storage to a SQL database with a schema definition that contains three INT type columns. The CSV file rows that contain numeric data, such as 123,456,789 are copied successfully to the sink store. However, the rows that contain non-numeric values, such as 123,456, abc are detected as incompatible and are skipped.

- **Mismatch in the number of columns between the source and the sink**.

  For example: Copy data from a CSV file in Blob storage to a SQL database with a schema definition that contains six columns. The CSV file rows that contain six columns are copied successfully to the sink store. The CSV file rows that contain more than six columns are detected as incompatible and are skipped.

- **Primary key violation when writing to SQL Server/Azure SQL Database/Azure Cosmos DB**.

  For example: Copy data from a SQL server to a SQL database. A primary key is defined in the sink SQL database, but no such primary key is defined in the source SQL server. The duplicated rows that exist in

the source cannot be copied to the sink. Copy activity copies only the first row of the source data into the sink. The subsequent source rows that contain the duplicated primary key value are detected as incompatible and are skipped.

NOTE

- To load data into Azure Synapse Analytics using PolyBase, configure PolyBase's native fault tolerance settings by specifying reject policies via "polyBaseSettings" in copy activity. You can still enable redirecting PolyBase incompatible rows to Blob or ADLS as normal as shown below.
- This feature doesn't apply when copy activity is configured to invoke Amazon Redshift Unload.
- This feature doesn't apply when copy activity is configured to invoke a stored procedure from a SQL sink.

Configuration

The following example provides a JSON definition to configure skipping the incompatible rows in copy activity:

```
"typeProperties": {
    "source": {
        "type": "AzureSqlSource"
    },
    "sink": {
        "type": "AzureSqlSink"
    },
    "enableSkipIncompatibleRow": true,
    "logSettings": {
        "enableCopyActivityLog": true,
        "copyActivityLogSettings": {
            "logLevel": "Warning",
            "enableReliableLogging": false
        },
        "logLocationSettings": {
            "linkedServiceName": {
               "referenceName": "ADLSGen2",
               "type": "LinkedServiceReference"
            },
            "path": "sessionlog/"
        }
    }
},
```

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
enableSkipIncompatibleRow	Specifies whether to skip incompatible rows during copy or not.	True False (default)	No
logSettings	A group of properties that can be specified when you want to log the incompatible rows.		No
linkedServiceName	The linked service of Azure Blob Storage or Azure Data Lake Storage Gen2 to store the log that contains the skipped rows.	The names of an `AzureBlobStorage` or `AzureBlobFS` type linked service, which refers to the instance that you use to store the log file.	No

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
path	The path of the log files that contains the skipped rows.	Specify the path that you want to use to log the incompatible data. If you do not provide a path, the service creates a container for you.	No

Monitor skipped rows

After the copy activity run completes, you can see the number of skipped rows in the output of the copy activity:

```
"output": {
            "dataRead": 95,
            "dataWritten": 186,
            "rowsCopied": 9,
            "rowsSkipped": 2,
            "copyDuration": 16,
            "throughput": 0.01,
            "logFilePath": "myfolder/a84bf8d4-233f-4216-8cb5-45962831cd1b/",
            "errors": []
        },
```

If you configure to log the incompatible rows, you can find the log file from this path:

```
https://[your-blob-account].blob.core.windows.net/[path-if-configured]/copyactivity-logs/[copy-activity-name]/[copy-activity-run-id]/[auto-generated-GUID].csv
```

.

The log files will be the csv files. The schema of the log file is as following:

COLUMN	DESCRIPTION
Timestamp	The timestamp when the incompatible rows were skipped
Level	The log level of this item. It will be in 'Warning' level if this item shows the skipped rows
OperationName	Copy activity operational behavior on each row. It will be 'TabularRowSkip' to specify that the particular incompatible row has been skipped
OperationItem	The skipped rows from the source data store.
Message	More information to illustrate why the incompatibility of this particular row.

An example of the log file content is as follows:

```
Timestamp, Level, OperationName, OperationItem, Message
2020-02-26 06:22:32.2586581, Warning, TabularRowSkip, """data1"", ""data2"", ""data3"","" "Column 'Prop_2' contains an invalid value 'data3'. Cannot convert 'data3' to type 'DateTime'."
2020-02-26 06:22:33.2586351, Warning, TabularRowSkip, """data4"", ""data5"", ""data6"","", "Violation of PRIMARY KEY constraint 'PK_tblintstrdatetimewithpk'. Cannot insert duplicate key in object 'dbo.tblintstrdatetimewithpk'. The duplicate key value is (data4)."
```

From the sample log file above, you can see one row "data1, data2, data3" has been skipped due to type

conversion issue from source to destination store. Another row "data4, data5, data6" has been skipped due to PK violation issue from source to destination store.

Copying tabular data (legacy):

The following approach is the legacy way to enable fault tolerance for copying tabular data only. If you are creating new pipeline or activity, you are encouraged to start from here instead.

Configuration

The following example provides a JSON definition to configure skipping the incompatible rows in copy activity:

```
"typeProperties": {
    "source": {
        "type": "BlobSource"
    },
    "sink": {
        "type": "SqlSink",
    },
    "enableSkipIncompatibleRow": true,
    "redirectIncompatibleRowSettings": {
         "linkedServiceName": {
              "referenceName": "<Azure Storage or Data Lake Store linked service>",
              "type": "LinkedServiceReference"
            },
            "path": "redirectcontainer/erroroutput"
     }
}
```

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
enableSkipIncompatibleRow	Specifies whether to skip incompatible rows during copy or not.	True False (default)	No
redirectIncompatibleRowSettings	A group of properties that can be specified when you want to log the incompatible rows.		No
linkedServiceName	The linked service of Azure Storage or Azure Data Lake Store to store the log that contains the skipped rows.	The names of an `AzureStorage` or `AzureDataLakeStore` type linked service, which refers to the instance that you want to use to store the log file.	No
path	The path of the log file that contains the skipped rows.	Specify the path that you want to use to log the incompatible data. If you do not provide a path, the service creates a container for you.	No

Monitor skipped rows

After the copy activity run completes, you can see the number of skipped rows in the output of the copy activity:

```
"output": {
            "dataRead": 95,
            "dataWritten": 186,
            "rowsCopied": 9,
            "rowsSkipped": 2,
            "copyDuration": 16,
            "throughput": 0.01,
            "redirectRowPath": "https://myblobstorage.blob.core.windows.net//myfolder/a84bf8d4-233f-4216-
8cb5-45962831cd1b/",
            "errors": []
        },
```

If you configure to log the incompatible rows, you can find the log file at this path:

```
https://[your-blob-account].blob.core.windows.net/[path-if-configured]/[copy-activity-run-id]/[auto-
generated-GUID].csv
```

.

The log files can only be the csv files. The original data being skipped will be logged with comma as column delimiter if needed. We add two more columns "ErrorCode" and "ErrorMessage" in additional to the original source data in log file, where you can see the root cause of the incompatibility. The ErrorCode and ErrorMessage will be quoted by double quotes.

An example of the log file content is as follows:

```
data1, data2, data3, "UserErrorInvalidDataValue", "Column 'Prop_2' contains an invalid value 'data3'. Cannot
convert 'data3' to type 'DateTime'."
data4, data5, data6, "2627", "Violation of PRIMARY KEY constraint 'PK_tblintstrdatetimewithpk'. Cannot
insert duplicate key in object 'dbo.tblintstrdatetimewithpk'. The duplicate key value is (data4)."
```

Next steps

See the other copy activity articles:

- Copy activity overview
- Copy activity performance

Data consistency verification in copy activity

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

When you move data from source to destination store, the copy activity provides an option for you to do additional data consistency verification to ensure the data is not only successfully copied from source to destination store, but also verified to be consistent between source and destination store. Once inconsistent files have been found during the data movement, you can either abort the copy activity or continue to copy the rest by enabling fault tolerance setting to skip inconsistent files. You can get the skipped file names by enabling session log setting in copy activity. You can refer to session log in copy activity for more details.

Supported data stores and scenarios

- Data consistency verification is supported by all the connectors except FTP, SFTP, HTTP, Snowflake, Office 365 and Azure Databricks Delta Lake.
- Data consistency verification is not supported in staging copy scenario.
- When copying binary files, data consistency verification is only available when 'PreserveHierarchy' behavior is set in copy activity.
- When copying multiple binary files in single copy activity with data consistency verification enabled, you have an option to either abort the copy activity or continue to copy the rest by enabling fault tolerance setting to skip inconsistent files.
- When copying a table in single copy activity with data consistency verification enabled, copy activity fails if the number of rows read from the source is different from the number of rows copied to the destination plus the number of incompatible rows that were skipped.

Configuration

The following example provides a JSON definition to enable data consistency verification in Copy Activity:

```
"typeProperties": {
"source": {
        "type": "BinarySource",
        "storeSettings": {
            "type": "AzureDataLakeStoreReadSettings",
            "recursive": true
        }
    },
    "sink": {
        "type": "BinarySink",
        "storeSettings": {
            "type": "AzureDataLakeStoreWriteSettings"
        }
},
    "validateDataConsistency": true,
    "skipErrorFile": {
        "dataInconsistency": true
    },
    "logSettings": {
        "enableCopyActivityLog": true,
        "copyActivityLogSettings": {
            "logLevel": "Warning",
            "enableReliableLogging": false
        },
        "logLocationSettings": {
            "linkedServiceName": {
               "referenceName": "ADLSGen2",
               "type": "LinkedServiceReference"
            },
            "path": "sessionlog/"
        }
    }
}
```

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
validateDataConsistency	If you set true for this property, when copying binary files, copy activity will check file size, lastModifiedDate, and MD5 checksum for each binary file copied from source to destination store to ensure the data consistency between source and destination store. When copying tabular data, copy activity will check the total row count after job completes to ensure the total number of rows read from the source is same as the number of rows copied to the destination plus the number of incompatible rows that were skipped. Be aware the copy performance will be affected by enabling this option.	True False (default)	No

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
dataInconsistency	One of the key-value pairs within skipErrorFile property bag to determine if you want to skip the inconsistent files. -True: you want to copy the rest by skipping inconsistent files. - False: you want to abort the copy activity once inconsistent file found. Be aware this property is only valid when you are copying binary files and set validateDataConsistency as True.	True False (default)	No
logSettings	A group of properties that can be specified to enable session log to log skipped files.		No
linkedServiceName	The linked service of Azure Blob Storage or Azure Data Lake Storage Gen2 to store the session log files.	The names of an `AzureBlobStorage` or `AzureBlobFS` types linked service, which refers to the instance that you use to store the log files.	No
path	The path of the log files.	Specify the path that you want to store the log files. If you do not provide a path, the service creates a container for you.	No

NOTE

- When copying binary files from, or to Azure Blob or Azure Data Lake Storage Gen2, the service does block level MD5 checksum verification leveraging Azure Blob API and Azure Data Lake Storage Gen2 API. If ContentMD5 on files exist on Azure Blob or Azure Data Lake Storage Gen2 as data sources, the service does file level MD5 checksum verification after reading the files as well. After copying files to Azure Blob or Azure Data Lake Storage Gen2 as data destination, the service writes ContentMD5 to Azure Blob or Azure Data Lake Storage Gen2 which can be further consumed by downstream applications for data consistency verification.
- The service does file size verification when copying binary files between any storage stores.

Monitoring

Output from copy activity

After the copy activity runs completely, you can see the result of data consistency verification from the output of each copy activity run:

```
"output": {
            "dataRead": 695,
            "dataWritten": 186,
            "filesRead": 3,
            "filesWritten": 1,
            "filesSkipped": 2,
            "throughput": 297,
            "logFilePath": "myfolder/a84bf8d4-233f-4216-8cb5-45962831cd1b/",
            "dataConsistencyVerification":
           {
                "VerificationResult": "Verified",
                "InconsistentData": "Skipped"
           }
        }
```

You can see the details of data consistency verification from "dataConsistencyVerification property".

Value of **VerificationResult**:

- **Verified**: Your copied data has been verified to be consistent between source and destination store.
- **NotVerified**: Your copied data has not been verified to be consistent because you have not enabled the validateDataConsistency in copy activity.
- **Unsupported**: Your copied data has not been verified to be consistent because data consistency verification is not supported for this particular copy pair.

Value of **InconsistentData**:

- **Found**: The copy activity has found inconsistent data.
- **Skipped**: The copy activity has found and skipped inconsistent data.
- **None**: The copy activity has not found any inconsistent data. It can be either because your data has been verified to be consistent between source and destination store or because you disabled validateDataConsistency in copy activity.

Session log from copy activity

If you configure to log the inconsistent file, you can find the log file from this path: `https://[your-blob-account].blob.core.windows.net/[path-if-configured]/copyactivity-logs/[copy-activity-name]/[copy-activity-run-id]/[auto-generated-GUID].csv`. The log files will be the csv files.

The schema of a log file is as following:

COLUMN	DESCRIPTION
Timestamp	The timestamp when the service skips the inconsistent files.
Level	The log level of this item. It will be in 'Warning' level for the item showing file skipping.
OperationName	The copy activity operational behavior on each file. It will be 'FileSkip' to specify the file to be skipped.
OperationItem	The file name to be skipped.
Message	More information to illustrate why files being skipped.

The example of a log file is as following:

```
Timestamp, Level, OperationName, OperationItem, Message
2020-02-26 06:22:56.3190846, Warning, FileSkip, "sample1.csv", "File is skipped after read 548000000 bytes:
ErrorCode=DataConsistencySourceDataChanged,'Type=Microsoft.DataTransfer.Common.Shared.HybridDeliveryExceptio
n,Message=Source file 'sample1.csv' is changed by other clients during the copy activity run.,Source=,'."
```

From the log file above, you can see sample1.csv has been skipped because it failed to be verified to be consistent between source and destination store. You can get more details about why sample1.csv becomes inconsistent is because it was being changed by other applications when the copy activity is copying at the same time.

Next steps

See the other Copy Activity articles:

- Copy activity overview
- Copy activity fault tolerance

Supported file formats and compression codecs in Azure Data Factory and Synapse Analytics (legacy)

8/19/2022 • 18 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article applies to the following connectors: Amazon S3, Azure Blob, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure Files, File System, FTP, Google Cloud Storage, HDFS, HTTP, and SFTP.

IMPORTANT

The service introduced new format-based dataset model, see corresponding format article with details:

- Avro format
- Binary format
- Delimited text format
- JSON format
- ORC format
- Parquet format

The rest configurations mentioned in this article are still supported as-is for backward compabitility. You are suggested to use the new model going forward.

Text format (legacy)

NOTE

Learn the new model from Delimited text format article. The following configurations on file-based data store dataset is still supported as-is for backward compabitility. You are suggested to use the new model going forward.

If you want to read from a text file or write to a text file, set the `type` property in the `format` section of the dataset to **TextFormat**. You can also specify the following **optional** properties in the `format` section. See TextFormat example section on how to configure.

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
columnDelimiter	The character used to separate columns in a file. You can consider to use a rare unprintable character that may not exist in your data. For example, specify "\u0001", which represents Start of Heading (SOH).	Only one character is allowed. The **default** value is **comma (',')**. To use a Unicode character, refer to Unicode Characters to get the corresponding code for it.	No
rowDelimiter	The character used to separate rows in a file.	Only one character is allowed. The **default** value is any of the following values on read: **["\r\n", "\r", "\n"]** and **"\r\n"** on write.	No

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
escapeChar	The special character used to escape a column delimiter in the content of input file. You cannot specify both escapeChar and quoteChar for a table.	Only one character is allowed. No default value. Example: if you have comma (',') as the column delimiter but you want to have the comma character in the text (example: "Hello, world"), you can define '$' as the escape character and use string "Hello$, world" in the source.	No
quoteChar	The character used to quote a string value. The column and row delimiters inside the quote characters would be treated as part of the string value. This property is applicable to both input and output datasets. You cannot specify both escapeChar and quoteChar for a table.	Only one character is allowed. No default value. For example, if you have comma (',') as the column delimiter but you want to have comma character in the text (example: <Hello, world>), you can define " (double quote) as the quote character and use the string "Hello, world" in the source.	No
nullValue	One or more characters used to represent a null value.	One or more characters. The **default** values are **"\N" and "NULL"** on read and **"\N"** on write.	No
encodingName	Specify the encoding name.	A valid encoding name. see Encoding.EncodingName Property. Example: windows-1250 or shift_jis. The **default** value is **UTF-8**.	No
firstRowAsHeader	Specifies whether to consider the first row as a header. For an input dataset, the service reads first row as a header. For an output dataset, the service writes first row as a header. See Scenarios for using `firstRowAsHeader` and `skipLineCount` for sample scenarios.	True **False (default)**	No

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
skipLineCount	Indicates the number of **non-empty** rows to skip when reading data from input files. If both skipLineCount and firstRowAsHeader are specified, the lines are skipped first and then the header information is read from the input file. See Scenarios for using `firstRowAsHeader` and `skipLineCount` for sample scenarios.	Integer	No
treatEmptyAsNull	Specifies whether to treat null or empty string as a null value when reading data from an input file.	**True (default)** False	No

TextFormat example

In the following JSON definition for a dataset, some of the optional properties are specified.

```
"typeProperties":
{
    "folderPath": "mycontainer/myfolder",
    "fileName": "myblobname",
    "format":
    {
        "type": "TextFormat",
        "columnDelimiter": ",",
        "rowDelimiter": ";",
        "quoteChar": "\"",
        "NullValue": "NaN",
        "firstRowAsHeader": true,
        "skipLineCount": 0,
        "treatEmptyAsNull": true
    }
},
```

To use an `escapeChar` instead of `quoteChar`, replace the line with `quoteChar` with the following escapeChar:

```
"escapeChar": "$",
```

Scenarios for using firstRowAsHeader and skipLineCount

- You are copying from a non-file source to a text file and would like to add a header line containing the schema metadata (for example: SQL schema). Specify `firstRowAsHeader` as true in the output dataset for this scenario.
- You are copying from a text file containing a header line to a non-file sink and would like to drop that line. Specify `firstRowAsHeader` as true in the input dataset.
- You are copying from a text file and want to skip a few lines at the beginning that contain no data or header information. Specify `skipLineCount` to indicate the number of lines to be skipped. If the rest of the file contains a header line, you can also specify `firstRowAsHeader`. If both `skipLineCount` and `firstRowAsHeader` are specified, the lines are skipped first and then the header information is read from the input file

JSON format (legacy)

> **NOTE**
>
> Learn the new model from JSON format article. The following configurations on file-based data store dataset is still supported as-is for backward compatibility. You are suggested to use the new model going forward.

To **import/export a JSON file as-is into/from Azure Cosmos DB**, see Import/export JSON documents section in Move data to/from Azure Cosmos DB article.

If you want to parse the JSON files or write the data in JSON format, set the `type` property in the `format` section to **JsonFormat**. You can also specify the following **optional** properties in the `format` section. See JsonFormat example section on how to configure.

PROPERTY	DESCRIPTION	REQUIRED
filePattern	Indicate the pattern of data stored in each JSON file. Allowed values are: **setOfObjects** and **arrayOfObjects**. The **default** value is **setOfObjects**. See JSON file patterns section for details about these patterns.	No
jsonNodeReference	If you want to iterate and extract data from the objects inside an array field with the same pattern, specify the JSON path of that array. This property is supported only when copying data **from** JSON files.	No
jsonPathDefinition	Specify the JSON path expression for each column mapping with a customized column name (start with lowercase). This property is supported only when copying data **from** JSON files, and you can extract data from object or array. For fields under root object, start with root $; for fields inside the array chosen by `jsonNodeReference` property, start from the array element. See JsonFormat example section on how to configure.	No
encodingName	Specify the encoding name. For the list of valid encoding names, see: Encoding.EncodingName Property. For example: windows-1250 or shift_jis. The **default** value is: **UTF-8**.	No
nestingSeparator	Character that is used to separate nesting levels. The default value is '.' (dot).	No

> **NOTE**
>
> For the case of cross-apply data in array into multiple rows (case 1 -> sample 2 in JsonFormat examples), you can only choose to expand single array using property `jsonNodeReference`.

JSON file patterns

Copy activity can parse the following patterns of JSON files:

- Type I: setOfObjects

  Each file contains single object, or line-delimited/concatenated multiple objects. When this option is chosen in an output dataset, copy activity produces a single JSON file with each object per line (line-delimited).

  - single object JSON example

    ```
    {
        "time": "2015-04-29T07:12:20.9100000Z",
        "callingimsi": "466920403025604",
        "callingnum1": "678948008",
        "callingnum2": "567834760",
        "switch1": "China",
        "switch2": "Germany"
    }
    ```

  - line-delimited JSON example

    ```
    {"time":"2015-04-
    29T07:12:20.9100000Z","callingimsi":"466920403025604","callingnum1":"678948008","callingnum2":
    "567834760","switch1":"China","switch2":"Germany"}
    {"time":"2015-04-
    29T07:13:21.0220000Z","callingimsi":"466922202613463","callingnum1":"123436380","callingnum2":
    "789037573","switch1":"US","switch2":"UK"}
    {"time":"2015-04-
    29T07:13:21.4370000Z","callingimsi":"466923101048691","callingnum1":"678901578","callingnum2":
    "345626404","switch1":"Germany","switch2":"UK"}
    ```

  - concatenated JSON example

```
{
    "time": "2015-04-29T07:12:20.9100000Z",
    "callingimsi": "466920403025604",
    "callingnum1": "678948008",
    "callingnum2": "567834760",
    "switch1": "China",
    "switch2": "Germany"
}
{
    "time": "2015-04-29T07:13:21.0220000Z",
    "callingimsi": "466922202613463",
    "callingnum1": "123436380",
    "callingnum2": "789037573",
    "switch1": "US",
    "switch2": "UK"
}
{
    "time": "2015-04-29T07:13:21.4370000Z",
    "callingimsi": "466923101048691",
    "callingnum1": "678901578",
    "callingnum2": "345626404",
    "switch1": "Germany",
    "switch2": "UK"
}
```

- Type II: arrayOfObjects

  Each file contains an array of objects.

```
[
    {
        "time": "2015-04-29T07:12:20.9100000Z",
        "callingimsi": "466920403025604",
        "callingnum1": "678948008",
        "callingnum2": "567834760",
        "switch1": "China",
        "switch2": "Germany"
    },
    {
        "time": "2015-04-29T07:13:21.0220000Z",
        "callingimsi": "466922202613463",
        "callingnum1": "123436380",
        "callingnum2": "789037573",
        "switch1": "US",
        "switch2": "UK"
    },
    {
        "time": "2015-04-29T07:13:21.4370000Z",
        "callingimsi": "466923101048691",
        "callingnum1": "678901578",
        "callingnum2": "345626404",
        "switch1": "Germany",
        "switch2": "UK"
    }
]
```

JsonFormat example

Case 1: Copying data from JSON files

Sample 1: extract data from object and array

In this sample, you expect one root JSON object maps to single record in tabular result. If you have a JSON file with the following content:

```
{
    "id": "ed0e4960-d9c5-11e6-85dc-d7996816aad3",
    "context": {
        "device": {
            "type": "PC"
        },
        "custom": {
            "dimensions": [
                {
                    "TargetResourceType": "Microsoft.Compute/virtualMachines"
                },
                {
                    "ResourceManagementProcessRunId": "827f8aaa-ab72-437c-ba48-d8917a7336a3"
                },
                {
                    "OccurrenceTime": "1/13/2017 11:24:37 AM"
                }
            ]
        }
    }
}
```

and you want to copy it into an Azure SQL table in the following format, by extracting data from both objects and array:

ID	DEVICETYPE	TARGETRESOURCETYPE	RESOURCEMANAGEMENTPROCESSRUNID	OCCURRENCETIME
ed0e4960-d9c5-11e6-85dc-d7996816aad3	PC	Microsoft.Compute/virtualMachines	827f8aaa-ab72-437c-ba48-d8917a7336a3	1/13/2017 11:24:37 AM

The input dataset with **JsonFormat** type is defined as follows: (partial definition with only the relevant parts). More specifically:

- `structure` section defines the customized column names and the corresponding data type while converting to tabular data. This section is **optional** unless you need to do column mapping. For more information, see Map source dataset columns to destination dataset columns.
- `jsonPathDefinition` specifies the JSON path for each column indicating where to extract the data from. To copy data from array, you can use `array[x].property` to extract value of the given property from the `xth` object, or you can use `array[*].property` to find the value from any object containing such property.

```
"properties": {
    "structure": [
        {
            "name": "id",
            "type": "String"
        },
        {
            "name": "deviceType",
            "type": "String"
        },
        {
            "name": "targetResourceType",
            "type": "String"
        },
        {
            "name": "resourceManagementProcessRunId",
            "type": "String"
        },
        {
            "name": "occurrenceTime",
            "type": "DateTime"
        }
    ],
    "typeProperties": {
        "folderPath": "mycontainer/myfolder",
        "format": {
            "type": "JsonFormat",
            "filePattern": "setOfObjects",
            "jsonPathDefinition": {"id": "$.id", "deviceType": "$.context.device.type",
"targetResourceType": "$.context.custom.dimensions[0].TargetResourceType", "resourceManagementProcessRunId":
"$.context.custom.dimensions[1].ResourceManagementProcessRunId", "occurrenceTime": "
$.context.custom.dimensions[2].OccurrenceTime"}
        }
    }
}
```

Sample 2: cross apply multiple objects with the same pattern from array

In this sample, you expect to transform one root JSON object into multiple records in tabular result. If you have a JSON file with the following content:

```
{
    "ordernumber": "01",
    "orderdate": "20170122",
    "orderlines": [
        {
            "prod": "p1",
            "price": 23
        },
        {
            "prod": "p2",
            "price": 13
        },
        {
            "prod": "p3",
            "price": 231
        }
    ],
    "city": [ { "sanmateo": "No 1" } ]
}
```

and you want to copy it into an Azure SQL table in the following format, by flattening the data inside the array and cross join with the common root info:

ORDERNUMBER	ORDERDATE	ORDER_PD	ORDER_PRICE	CITY
01	20170122	P1	23	`[{"sanmateo":"No 1"}]`
01	20170122	P2	13	`[{"sanmateo":"No 1"}]`
01	20170122	P3	231	`[{"sanmateo":"No 1"}]`

The input dataset with **JsonFormat** type is defined as follows: (partial definition with only the relevant parts). More specifically:

- `structure` section defines the customized column names and the corresponding data type while converting to tabular data. This section is **optional** unless you need to do column mapping. For more information, see Map source dataset columns to destination dataset columns.
- `jsonNodeReference` indicates to iterate and extract data from the objects with the same pattern under **array** `orderlines`.
- `jsonPathDefinition` specifies the JSON path for each column indicating where to extract the data from. In this example, `ordernumber`, `orderdate`, and `city` are under root object with JSON path starting with `$.`, while `order_pd` and `order_price` are defined with path derived from the array element without `$.`.

```
"properties": {
    "structure": [
        {
            "name": "ordernumber",
            "type": "String"
        },
        {
            "name": "orderdate",
            "type": "String"
        },
        {
            "name": "order_pd",
            "type": "String"
        },
        {
            "name": "order_price",
            "type": "Int64"
        },
        {
            "name": "city",
            "type": "String"
        }
    ],
    "typeProperties": {
        "folderPath": "mycontainer/myfolder",
        "format": {
            "type": "JsonFormat",
            "filePattern": "setOfObjects",
            "jsonNodeReference": "$.orderlines",
            "jsonPathDefinition": {"ordernumber": "$.ordernumber", "orderdate": "$.orderdate", "order_pd":
"prod", "order_price": "price", "city": " $.city"}
        }
    }
}
```

Note the following points:

- If the `structure` and `jsonPathDefinition` are not defined in the dataset, the Copy Activity detects the schema from the first object and flatten the whole object.
- If the JSON input has an array, by default the Copy Activity converts the entire array value into a string. You can choose to extract data from it using `jsonNodeReference` and/or `jsonPathDefinition`, or skip it by not specifying it in `jsonPathDefinition`.
- If there are duplicate names at the same level, the Copy Activity picks the last one.
- Property names are case-sensitive. Two properties with same name but different casings are treated as two separate properties.

Case 2: Writing data to JSON file

If you have the following table in SQL Database:

ID	ORDER_DATE	ORDER_PRICE	ORDER_BY
1	20170119	2000	David
2	20170120	3500	Patrick
3	20170121	4000	Jason

and for each record, you expect to write to a JSON object in the following format:

```
{
    "id": "1",
    "order": {
        "date": "20170119",
        "price": 2000,
        "customer": "David"
    }
}
```

The output dataset with **JsonFormat** type is defined as follows: (partial definition with only the relevant parts). More specifically, `structure` section defines the customized property names in destination file, `nestingSeparator` (default is ".") are used to identify the nest layer from the name. This section is **optional** unless you want to change the property name comparing with source column name, or nest some of the properties.

```
"properties": {
    "structure": [
        {
            "name": "id",
            "type": "String"
        },
        {
            "name": "order.date",
            "type": "String"
        },
        {
            "name": "order.price",
            "type": "Int64"
        },
        {
            "name": "order.customer",
            "type": "String"
        }
    ],
    "typeProperties": {
        "folderPath": "mycontainer/myfolder",
        "format": {
            "type": "JsonFormat"
        }
    }
}
```

Parquet format (legacy)

> **NOTE**
>
> Learn the new model from Parquet format article. The following configurations on file-based data store dataset is still supported as-is for backward compabitility. You are suggested to use the new model going forward.

If you want to parse the Parquet files or write the data in Parquet format, set the `format` `type` property to **ParquetFormat**. You do not need to specify any properties in the Format section within the typeProperties section. Example:

```
"format":
{
    "type": "ParquetFormat"
}
```

Note the following points:

- Complex data types are not supported (MAP, LIST).
- White space in column name is not supported.
- Parquet file has the following compression-related options: NONE, SNAPPY, GZIP, and LZO. The service supports reading data from Parquet file in any of these compressed formats except LZO - it uses the compression codec in the metadata to read the data. However, when writing to a Parquet file, the service chooses SNAPPY, which is the default for Parquet format. Currently, there is no option to override this behavior.

IMPORTANT

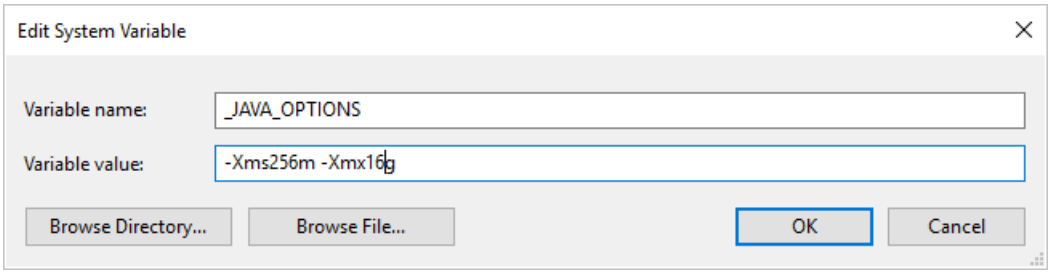
For copy empowered by Self-hosted Integration Runtime e.g. between on-premises and cloud data stores, if you are not copying Parquet files **as-is**, you need to install the **64-bit JRE 8 (Java Runtime Environment) or OpenJDK** on your IR machine. See the following paragraph with more details.

For copy running on Self-hosted IR with Parquet file serialization/deserialization, the service locates the Java runtime by firstly checking the registry `(SOFTWARE\JavaSoft\Java Runtime Environment\{Current Version}\JavaHome)` for JRE, if not found, secondly checking system variable `JAVA_HOME` for OpenJDK.

- **To use JRE**: The 64-bit IR requires 64-bit JRE. You can find it from here.
- **To use OpenJDK**: it's supported since IR version 3.13. Package the jvm.dll with all other required assemblies of OpenJDK into Self-hosted IR machine, and set system environment variable JAVA_HOME accordingly.

TIP

If you copy data to/from Parquet format using Self-hosted Integration Runtime and hit error saying "An error occurred when invoking java, message: **java.lang.OutOfMemoryError:Java heap space**", you can add an environment variable `_JAVA_OPTIONS` in the machine that hosts the Self-hosted IR to adjust the min/max heap size for JVM to empower such copy, then rerun the pipeline.



Example: set variable `_JAVA_OPTIONS` with value `-Xms256m -Xmx16g`. The flag `Xms` specifies the initial memory allocation pool for a Java Virtual Machine (JVM), while `Xmx` specifies the maximum memory allocation pool. This means that JVM will be started with `Xms` amount of memory and will be able to use a maximum of `Xmx` amount of memory. By default, the service uses min 64MB and max 1G.

Data type mapping for Parquet files

INTERIM SERVICE DATA TYPE	PARQUET PRIMITIVE TYPE	PARQUET ORIGINAL TYPE (DESERIALIZE)	PARQUET ORIGINAL TYPE (SERIALIZE)
Boolean	Boolean	N/A	N/A
SByte	Int32	Int8	Int8
Byte	Int32	UInt8	Int16
Int16	Int32	Int16	Int16
UInt16	Int32	UInt16	Int32
Int32	Int32	Int32	Int32
UInt32	Int64	UInt32	Int64

INTERIM SERVICE DATA TYPE	PARQUET PRIMITIVE TYPE	PARQUET ORIGINAL TYPE (DESERIALIZE)	PARQUET ORIGINAL TYPE (SERIALIZE)
Int64	Int64	Int64	Int64
UInt64	Int64/Binary	UInt64	Decimal
Single	Float	N/A	N/A
Double	Double	N/A	N/A
Decimal	Binary	Decimal	Decimal
String	Binary	Utf8	Utf8
DateTime	Int96	N/A	N/A
TimeSpan	Int96	N/A	N/A
DateTimeOffset	Int96	N/A	N/A
ByteArray	Binary	N/A	N/A
Guid	Binary	Utf8	Utf8
Char	Binary	Utf8	Utf8
CharArray	Not supported	N/A	N/A

ORC format (legacy)

> **NOTE**
> Learn the new model from ORC format article. The following configurations on file-based data store dataset is still supported as-is for backward compatibility. You are suggested to use the new model going forward.

If you want to parse the ORC files or write the data in ORC format, set the `format` `type` property to **OrcFormat**. You do not need to specify any properties in the Format section within the typeProperties section. Example:

```
"format":
{
    "type": "OrcFormat"
}
```

Note the following points:

- Complex data types are not supported (STRUCT, MAP, LIST, UNION).
- White space in column name is not supported.
- ORC file has three compression-related options: NONE, ZLIB, SNAPPY. The service supports reading data from ORC file in any of these compressed formats. It uses the compression codec is in the metadata to read the data. However, when writing to an ORC file, the service chooses ZLIB, which is the default for ORC.

Currently, there is no option to override this behavior.

> **IMPORTANT**
> For copy empowered by Self-hosted Integration Runtime e.g. between on-premises and cloud data stores, if you are not copying ORC files **as-is**, you need to install the **64-bit JRE 8 (Java Runtime Environment) or OpenJDK** on your IR machine. See the following paragraph with more details.

For copy running on Self-hosted IR with ORC file serialization/deserialization, the service locates the Java runtime by firstly checking the registry `(SOFTWARE\JavaSoft\Java Runtime Environment\{Current Version}\JavaHome)` for JRE, if not found, secondly checking system variable `JAVA_HOME` for OpenJDK.

- **To use JRE**: The 64-bit IR requires 64-bit JRE. You can find it from here.
- **To use OpenJDK**: it's supported since IR version 3.13. Package the jvm.dll with all other required assemblies of OpenJDK into Self-hosted IR machine, and set system environment variable JAVA_HOME accordingly.

Data type mapping for ORC files

INTERIM SERVICE DATA TYPE	ORC TYPES
Boolean	Boolean
SByte	Byte
Byte	Short
Int16	Short
UInt16	Int
Int32	Int
UInt32	Long
Int64	Long
UInt64	String
Single	Float
Double	Double
Decimal	Decimal
String	String
DateTime	Timestamp
DateTimeOffset	Timestamp
TimeSpan	Timestamp

INTERIM SERVICE DATA TYPE	ORC TYPES
ByteArray	Binary
Guid	String
Char	Char(1)

AVRO format (legacy)

> **NOTE**
> Learn the new model from Avro format article. The following configurations on file-based data store dataset is still supported as-is for backward compatibility. You are suggested to use the new model going forward.

If you want to parse the Avro files or write the data in Avro format, set the `format` `type` property to **AvroFormat**. You do not need to specify any properties in the Format section within the typeProperties section. Example:

```
"format":
{
    "type": "AvroFormat",
}
```

To use Avro format in a Hive table, you can refer to Apache Hive's tutorial.

Note the following points:

- Complex data types are not supported (records, enums, arrays, maps, unions, and fixed).

Compression support (legacy)

The service supports compress/decompress data during copy. When you specify `compression` property in an input dataset, the copy activity read the compressed data from the source and decompress it; and when you specify the property in an output dataset, the copy activity compress then write data to the sink. Here are a few sample scenarios:

- Read GZIP compressed data from an Azure blob, decompress it, and write result data to Azure SQL Database. You define the input Azure Blob dataset with the `compression` `type` property as GZIP.
- Read data from a plain-text file from on-premises File System, compress it using GZip format, and write the compressed data to an Azure blob. You define an output Azure Blob dataset with the `compression` `type` property as GZip.
- Read .zip file from FTP server, decompress it to get the files inside, and land those files in Azure Data Lake Store. You define an input FTP dataset with the `compression` `type` property as ZipDeflate.
- Read a GZIP-compressed data from an Azure blob, decompress it, compress it using BZIP2, and write result data to an Azure blob. You define the input Azure Blob dataset with `compression` `type` set to GZIP and the output dataset with `compression` `type` set to BZIP2.

To specify compression for a dataset, use the **compression** property in the dataset JSON as in the following example:

```
{
    "name": "AzureBlobDataSet",
    "properties": {
        "type": "AzureBlob",
        "linkedServiceName": {
            "referenceName": "StorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "fileName": "pagecounts.csv.gz",
            "folderPath": "compression/file/",
            "format": {
                "type": "TextFormat"
            },
            "compression": {
                "type": "GZip",
                "level": "Optimal"
            }
        }
    }
}
```

The **compression** section has two properties:

- **Type:** the compression codec, which can be **GZIP**, **Deflate**, **BZIP2**, or **ZipDeflate**. Note when using copy activity to decompress ZipDeflate file(s) and write to file-based sink data store, files will be extracted to the folder: `<path specified in dataset>/<folder named as source zip file>/`.
- **Level:** the compression ratio, which can be **Optimal** or **Fastest**.
  - **Fastest:** The compression operation should complete as quickly as possible, even if the resulting file is not optimally compressed.
  - **Optimal**: The compression operation should be optimally compressed, even if the operation takes a longer time to complete.

    For more information, see Compression Level topic.

> **NOTE**
>
> Compression settings are not supported for data in the **AvroFormat**, **OrcFormat**, or **ParquetFormat**. When reading files in these formats, the service detects and uses the compression codec in the metadata. When writing to files in these formats, the service chooses the default compression codec for that format. For example, ZLIB for OrcFormat and SNAPPY for ParquetFormat.

Unsupported file types and compression formats

You can use the extensibility features to transform files that aren't supported. Two options include Azure Functions and custom tasks by using Azure Batch.

You can see a sample that uses an Azure function to extract the contents of a tar file. For more information, see Azure Functions activity.

You can also build this functionality using a custom dotnet activity. Further information is available here

Next steps

Learn the latest supported file formats and compressions from Supported file formats and compressions.

Transform data in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

IMPORTANT

Support for Machine Learning Studio (classic) will end on 31 August 2024. We recommend you transition to Azure Machine Learning by that date.

Beginning 1 December 2021, you will not be able to create new Machine Learning Studio (classic) resources (workspace and web service plan). Through 31 August 2024, you can continue to use the existing Machine Learning Studio (classic) experiments and web services.

- See information on moving machine learning projects from ML Studio (classic) to Azure Machine Learning.
- Learn more about Azure Machine Learning

ML Studio (classic) documentation is being retired and may not be updated in the future.

Overview

This article explains data transformation activities in Azure Data Factory and Synapse pipelines that you can use to transform and process your raw data into predictions and insights at scale. A transformation activity executes in a computing environment such as Azure Databricks or Azure HDInsight. It provides links to articles with detailed information on each transformation activity.

The service supports the following data transformation activities that can be added to pipelines either individually or chained with another activity.

Transform natively in Azure Data Factory and Azure Synapse Analytics with data flows

Mapping data flows

Mapping data flows are visually designed data transformations in Azure Data Factory and Azure Synapse. Data flows allow data engineers to develop graphical data transformation logic without writing code. The resulting data flows are executed as activities within pipelines that use scaled-out Spark clusters. Data flow activities can be operationalized via existing scheduling, control, flow, and monitoring capabilities within the service. For more information, see mapping data flows.

Data wrangling

Power Query in Azure Data Factory enables cloud-scale data wrangling, which allows you to do code-free data preparation at cloud scale iteratively. Data wrangling integrates with Power Query Online and makes Power Query M functions available for data wrangling at cloud scale via spark execution. For more information, see data wrangling in Azure Data Factory.

> **NOTE**
>
> Power Query is currently only supported in Azure Data Factory, and not in Azure Synapse. For a list of specific features supported in each service, see Available features in Azure Data Factory & Azure Synapse Analytics pipelines.

External transformations

Optionally, you can hand-code transformations and manage the external compute environment yourself.

HDInsight Hive activity

The HDInsight Hive activity in a pipeline executes Hive queries on your own or on-demand Windows/Linux-based HDInsight cluster. See Hive activity article for details about this activity.

HDInsight Pig activity

The HDInsight Pig activity in a pipeline executes Pig queries on your own or on-demand Windows/Linux-based HDInsight cluster. See Pig activity article for details about this activity.

HDInsight MapReduce activity

The HDInsight MapReduce activity in a pipeline executes MapReduce programs on your own or on-demand Windows/Linux-based HDInsight cluster. See MapReduce activity article for details about this activity.

HDInsight Streaming activity

The HDInsight Streaming activity in a pipeline executes Hadoop Streaming programs on your own or on-demand Windows/Linux-based HDInsight cluster. See HDInsight Streaming activity for details about this activity.

HDInsight Spark activity

The HDInsight Spark activity in a pipeline executes Spark programs on your own HDInsight cluster. For details, see Invoke Spark programs with Azure Data Factory or Azure Synapse Analytics.

ML Studio (classic) activities

> **IMPORTANT**
>
> Support for Machine Learning Studio (classic) will end on 31 August 2024. We recommend you transition to Azure Machine Learning by that date.
>
> Beginning 1 December 2021, you will not be able to create new Machine Learning Studio (classic) resources (workspace and web service plan). Through 31 August 2024, you can continue to use the existing Machine Learning Studio (classic) experiments and web services.
>
> - See information on moving machine learning projects from ML Studio (classic) to Azure Machine Learning.
> - Learn more about Azure Machine Learning
>
> ML Studio (classic) documentation is being retired and may not be updated in the future.

The service enables you to easily create pipelines that use a published ML Studio (classic) web service for predictive analytics. Using the Batch Execution activity in a pipeline, you can invoke a Studio (classic) web service to make predictions on the data in batch.

Over time, the predictive models in the Studio (classic) scoring experiments need to be retrained using new input datasets. After you are done with retraining, you want to update the scoring web service with the retrained machine learning model. You can use the Update Resource activity to update the web service with the newly trained model.

See Use ML Studio (classic) activities for details about these Studio (classic) activities.

Stored procedure activity

You can use the SQL Server Stored Procedure activity in a Data Factory pipeline to invoke a stored procedure in one of the following data stores: Azure SQL Database, Azure Synapse Analytics, SQL Server Database in your enterprise or an Azure VM. See Stored Procedure activity article for details.

Data Lake Analytics U-SQL activity

Data Lake Analytics U-SQL activity runs a U-SQL script on an Azure Data Lake Analytics cluster. See Data Analytics U-SQL activity article for details.

Azure Synapse Notebook activity

The Azure Synapse Notebook Activity in a Synapse pipeline runs a Synapse notebook in your Azure Synapse workspace. See Transform data by running an Azure Synapse notebook.

Databricks Notebook activity

The Azure Databricks Notebook Activity in a pipeline runs a Databricks notebook in your Azure Databricks workspace. Azure Databricks is a managed platform for running Apache Spark. See Transform data by running a Databricks notebook.

Databricks Jar activity

The Azure Databricks Jar Activity in a pipeline runs a Spark Jar in your Azure Databricks cluster. Azure Databricks is a managed platform for running Apache Spark. See Transform data by running a Jar activity in Azure Databricks.

Databricks Python activity

The Azure Databricks Python Activity in a pipeline runs a Python file in your Azure Databricks cluster. Azure Databricks is a managed platform for running Apache Spark. See Transform data by running a Python activity in Azure Databricks.

Custom activity

If you need to transform data in a way that is not supported by Data Factory, you can create a custom activity with your own data processing logic and use the activity in the pipeline. You can configure the custom .NET activity to run using either an Azure Batch service or an Azure HDInsight cluster. See Use custom activities article for details.

You can create a custom activity to run R scripts on your HDInsight cluster with R installed. See Run R Script using Azure Data Factory and Synapse pipelines.

Compute environments

You create a linked service for the compute environment and then use the linked service when defining a transformation activity. There are two supported types of compute environments.

- **On-Demand**: In this case, the computing environment is fully managed by the service. It is automatically created by the service before a job is submitted to process data and removed when the job is completed. You can configure and control granular settings of the on-demand compute environment for job execution, cluster management, and bootstrapping actions.
- **Bring Your Own**: In this case, you can register your own computing environment (for example HDInsight cluster) as a linked service. The computing environment is managed by you and the service uses it to execute the activities.

See Compute Linked Services article to learn about supported compute services.

Next steps

See the following tutorial for an example of using a transformation activity: Tutorial: transform data using Spark

Data Flow activity in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

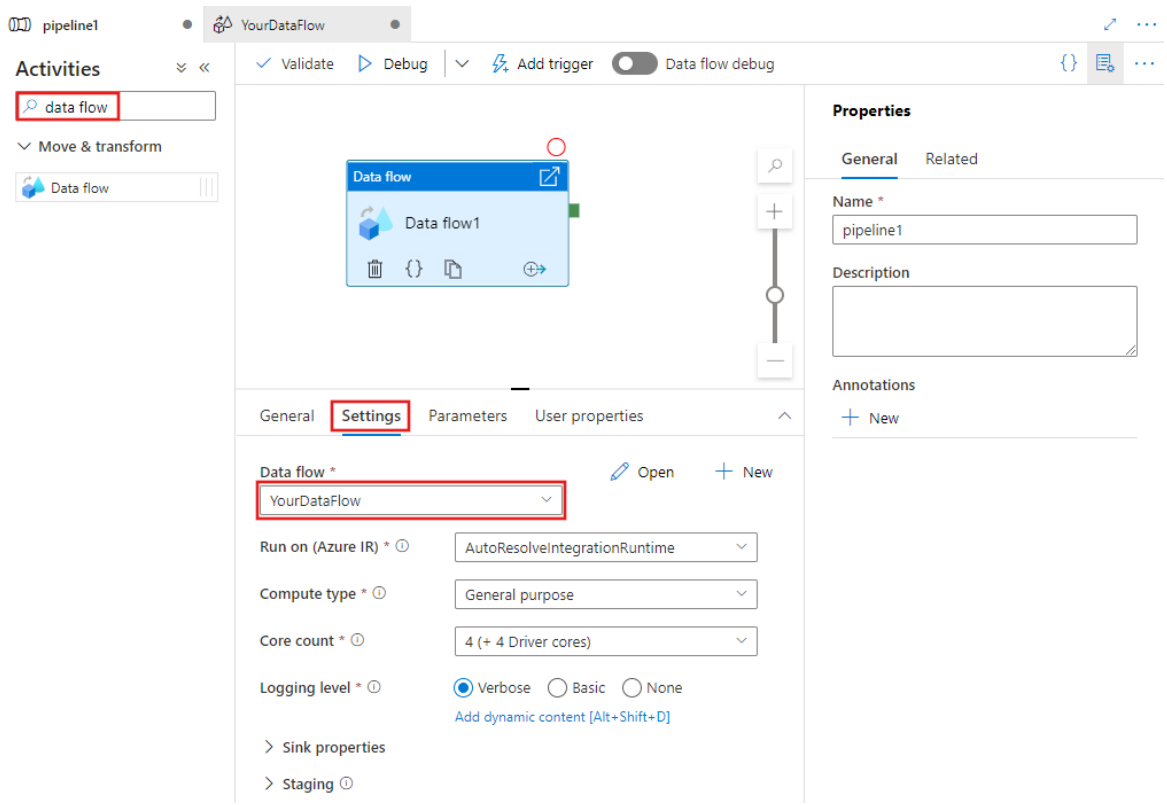
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Use the Data Flow activity to transform and move data via mapping data flows. If you're new to data flows, see Mapping Data Flow overview

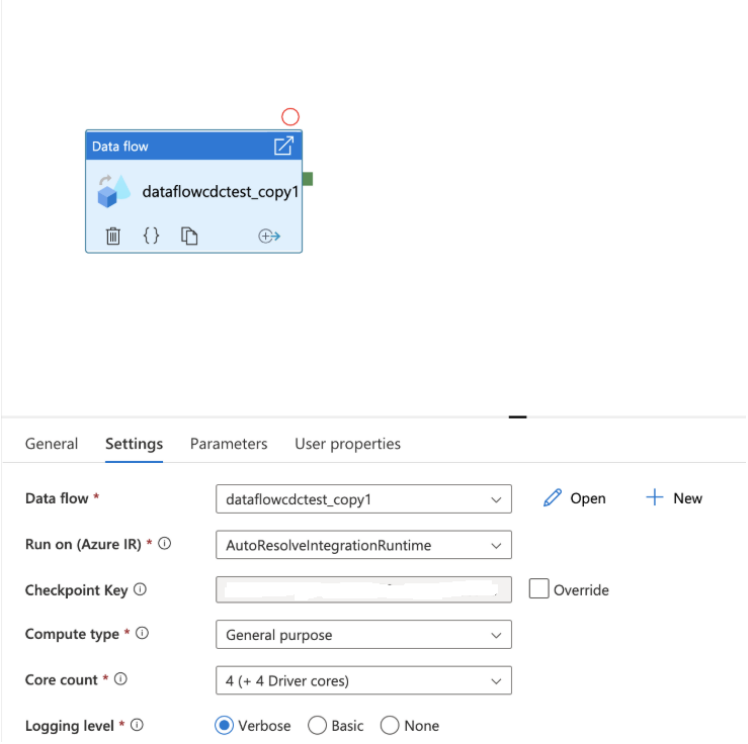
Create a Data Flow activity with UI

To use a Data Flow activity in a pipeline, complete the following steps:

1. Search for *Data Flow* in the pipeline Activities pane, and drag a Data Flow activity to the pipeline canvas.
2. Select the new Data Flow activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



3. Checkpoint key is used to set the checkpoint when data flow is used for changed data capture. You can overwrite it. Data flow activities use a guid value as checkpoint key instead of "pipelinename + activityname" so that it can always keep tracking customer's change data capture state even there's any renaming actions. All existing data flow activity will use the old pattern key for backward compatibility. Checkpoint key option after publishing a new data flow activity with change data capture enabled data flow resource is shown as below.



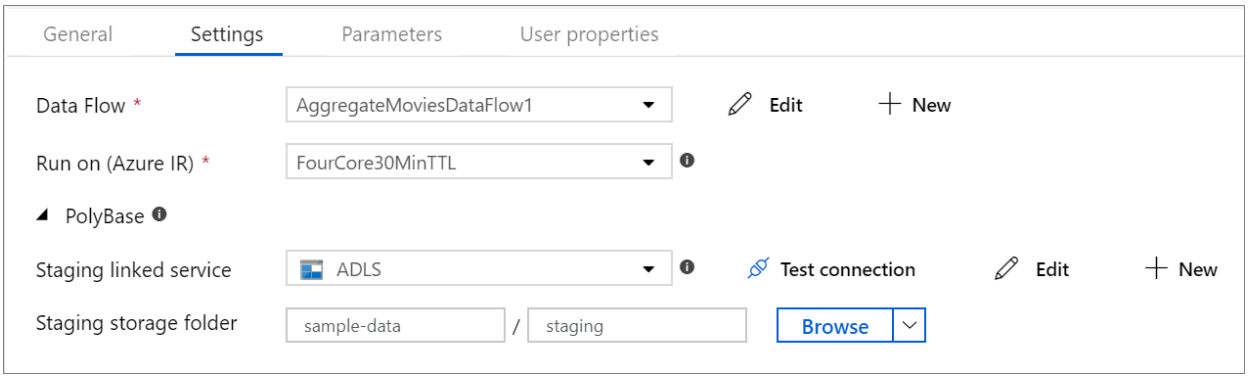
4. Select an existing data flow or create a new one using the New button. Select other options as required to complete your configuration.

Syntax

```
{
    "name": "MyDataFlowActivity",
    "type": "ExecuteDataFlow",
    "typeProperties": {
      "dataflow": {
         "referenceName": "MyDataFlow",
         "type": "DataFlowReference"
      },
      "compute": {
         "coreCount": 8,
         "computeType": "General"
      },
      "traceLevel": "Fine",
      "runConcurrently": true,
      "continueOnError": true,
      "staging": {
          "linkedService": {
              "referenceName": "MyStagingLinkedService",
              "type": "LinkedServiceReference"
          },
          "folderPath": "my-container/my-folder"
      },
      "integrationRuntime": {
          "referenceName": "MyDataFlowIntegrationRuntime",
          "type": "IntegrationRuntimeReference"
      }
}
```

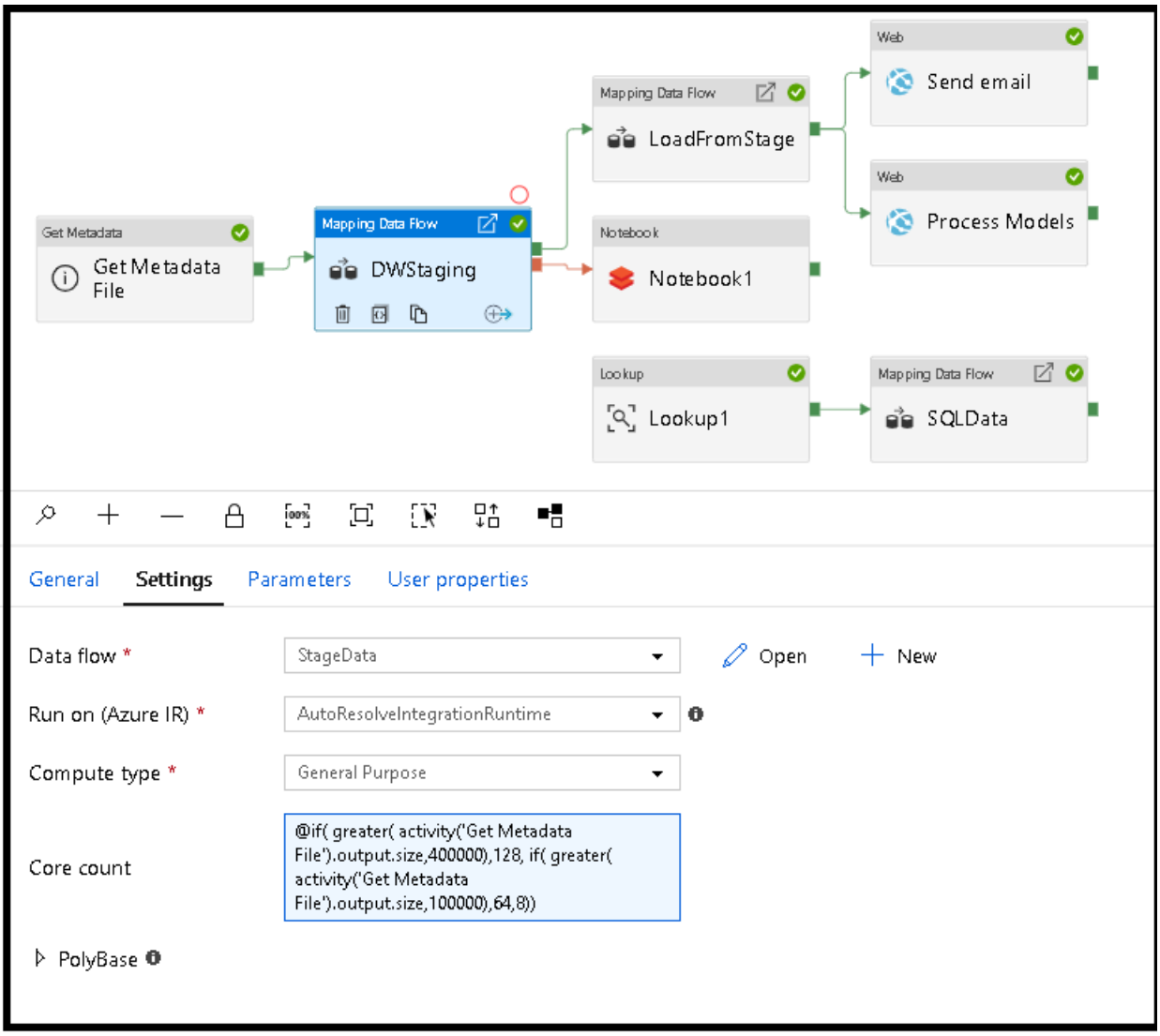
Type properties

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
dataflow	The reference to the Data Flow being executed	DataFlowReference	Yes
integrationRuntime	The compute environment the data flow runs on. If not specified, the auto-resolve Azure integration runtime will be used.	IntegrationRuntimeReference	No
compute.coreCount	The number of cores used in the spark cluster. Can only be specified if the auto-resolve Azure Integration runtime is used	8, 16, 32, 48, 80, 144, 272	No
compute.computeType	The type of compute used in the spark cluster. Can only be specified if the auto-resolve Azure Integration runtime is used	"General", "MemoryOptimized"	No
staging.linkedService	If you're using an Azure Synapse Analytics source or sink, specify the storage account used for PolyBase staging. If your Azure Storage is configured with VNet service endpoint, you must use managed identity authentication with "allow trusted Microsoft service" enabled on storage account, refer to Impact of using VNet Service Endpoints with Azure storage. Also learn the needed configurations for Azure Blob and Azure Data Lake Storage Gen2 respectively.	LinkedServiceReference	Only if the data flow reads or writes to an Azure Synapse Analytics
staging.folderPath	If you're using an Azure Synapse Analytics source or sink, the folder path in blob storage account used for PolyBase staging	String	Only if the data flow reads or writes to Azure Synapse Analytics
traceLevel	Set logging level of your data flow activity execution	Fine, Coarse, None	No



Dynamically size data flow compute at runtime

The Core Count and Compute Type properties can be set dynamically to adjust to the size of your incoming source data at runtime. Use pipeline activities like Lookup or Get Metadata in order to find the size of the source dataset data. Then, use Add Dynamic Content in the Data Flow activity properties. You can choose small, medium, or large compute sizes. Optionally, pick "Custom" and configure the compute types and number of cores manually.



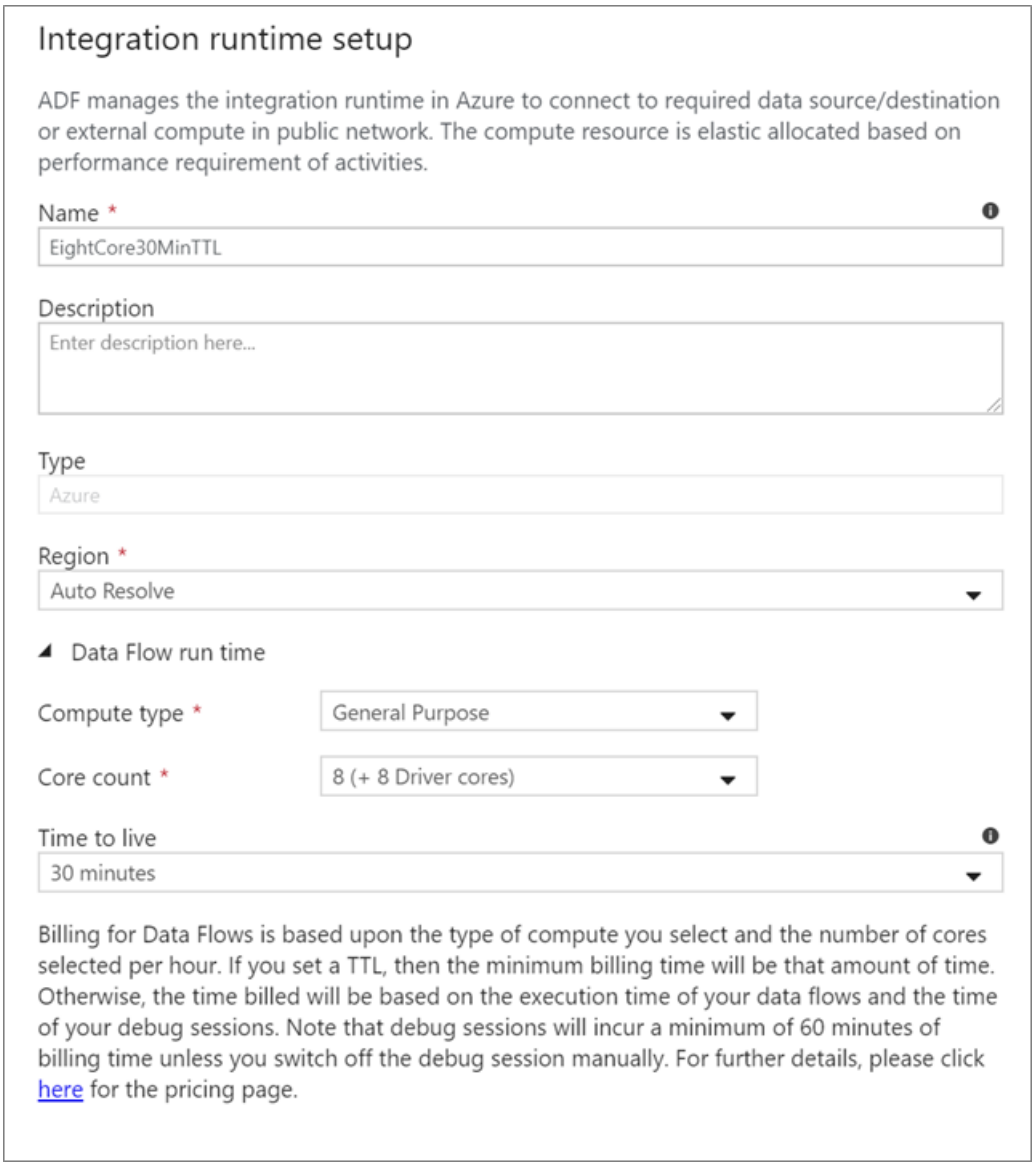
Here is a brief video tutorial explaining this technique

Data Flow integration runtime

Choose which Integration Runtime to use for your Data Flow activity execution. By default, the service will use the auto-resolve Azure Integration runtime with four worker cores. This IR has a general purpose compute type and runs in the same region as your service instance. For operationalized pipelines, it is highly recommended that you create your own Azure Integration Runtimes that define specific regions, compute type, core counts,

and TTL for your data flow activity execution.

A minimum compute type of General Purpose with an 8+8 (16 total v-cores) configuration and a 10-minute Time to live (TTL) is the minimum recommendation for most production workloads. By setting a small TTL, the Azure IR can maintain a warm cluster that will not incur the several minutes of start time for a cold cluster. For more information, see Azure integration runtime.



IMPORTANT

The Integration Runtime selection in the Data Flow activity only applies to *triggered executions* of your pipeline. Debugging your pipeline with data flows runs on the cluster specified in the debug session.

PolyBase

If you're using an Azure Synapse Analytics as a sink or source, you must choose a staging location for your PolyBase batch load. PolyBase allows for batch loading in bulk instead of loading the data row-by-row. PolyBase drastically reduces the load time into Azure Synapse Analytics.

Checkpoint key

When using the change capture option for data flow sources, ADF will maintain and manage the checkpoint for you automatically. The default checkpoint key is a hash of the data flow name and the pipeline name. If you are using a dynamic pattern for your source tables or folders, you may wish to override this hash and set your own checkpoint key value here.

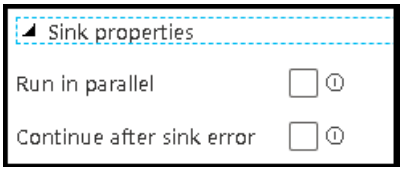
Logging level

If you do not require every pipeline execution of your data flow activities to fully log all verbose telemetry logs, you can optionally set your logging level to "Basic" or "None". When executing your data flows in "Verbose" mode (default), you are requesting the service to fully log activity at each individual partition level during your data transformation. This can be an expensive operation, so only enabling verbose when troubleshooting can improve your overall data flow and pipeline performance. "Basic" mode will only log transformation durations while "None" will only provide a summary of durations.

Sink properties

The grouping feature in data flows allow you to both set the order of execution of your sinks as well as to group sinks together using the same group number. To help manage groups, you can ask the service to run sinks, in the same group, in parallel. You can also set the sink group to continue even after one of the sinks encounters an error.

The default behavior of data flow sinks is to execute each sink sequentially, in a serial manner, and to fail the data flow when an error is encountered in the sink. Additionally, all sinks are defaulted to the same group unless you go into the data flow properties and set different priorities for the sinks.



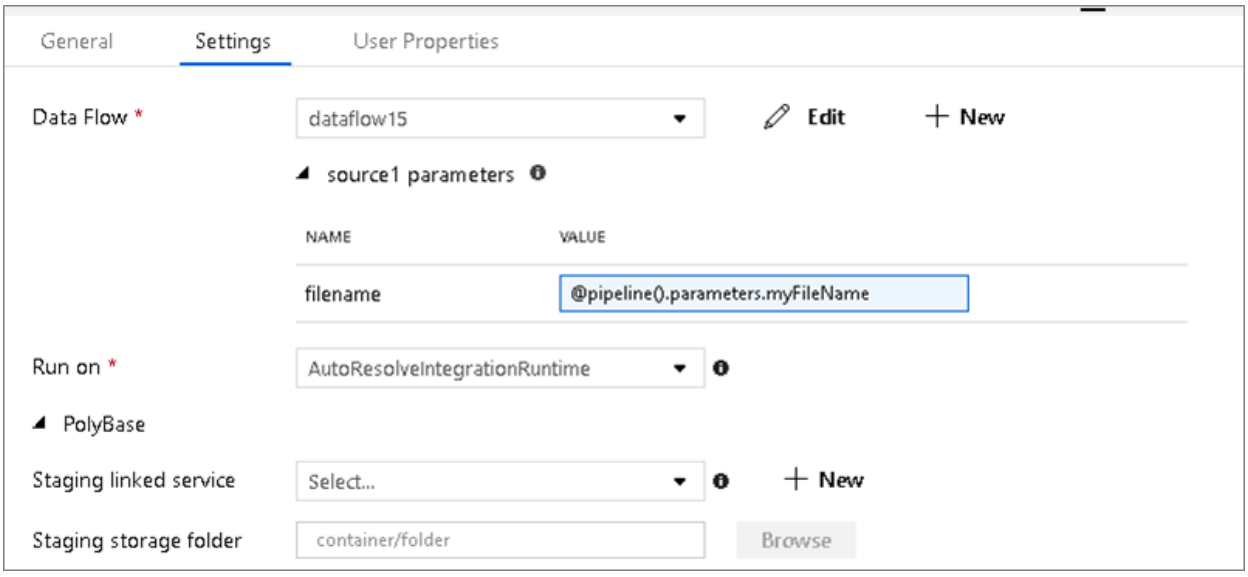
First row only

This option is only available for data flows that have cache sinks enabled for "Output to activity". The output from the data flow that is injected directly into your pipeline is limited to 2MB. Setting "first row only" helps you to limit the data output from data flow when injecting the data flow activity output directly to your pipeline.

Parameterizing Data Flows

Parameterized datasets

If your data flow uses parameterized datasets, set the parameter values in the **Settings** tab.

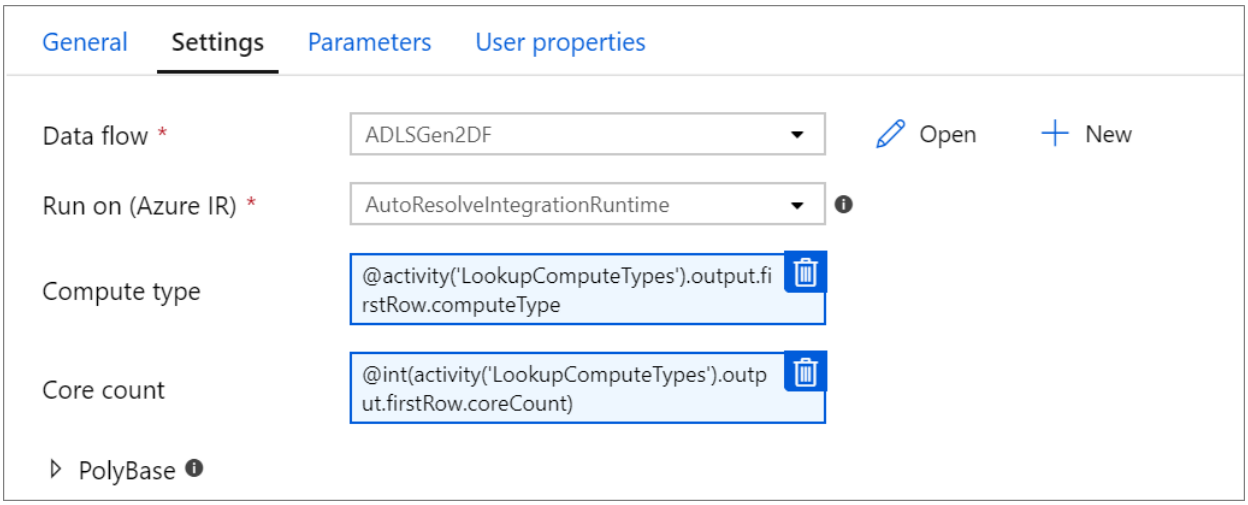


Parameterized data flows

If your data flow is parameterized, set the dynamic values of the data flow parameters in the **Parameters** tab. You can use either the pipeline expression language or the data flow expression language to assign dynamic or literal parameter values. For more information, see Data Flow Parameters.

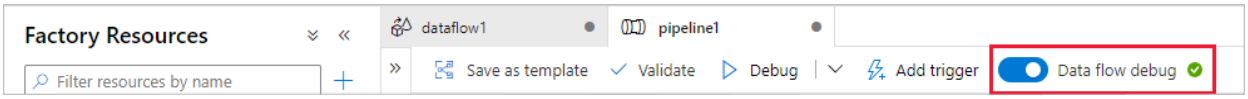
Parameterized compute properties.

You can parameterize the core count or compute type if you use the auto-resolve Azure Integration runtime and specify values for compute.coreCount and compute.computeType.



Pipeline debug of Data Flow activity

To execute a debug pipeline run with a Data Flow activity, you must switch on data flow debug mode via the **Data Flow Debug** slider on the top bar. Debug mode lets you run the data flow against an active Spark cluster. For more information, see Debug Mode.



The debug pipeline runs against the active debug cluster, not the integration runtime environment specified in the Data Flow activity settings. You can choose the debug compute environment when starting up debug mode.

Monitoring the Data Flow activity

The Data Flow activity has a special monitoring experience where you can view partitioning, stage time, and data lineage information. Open the monitoring pane via the eyeglasses icon under **Actions**. For more information, see Monitoring Data Flows.

Use Data Flow activity results in a subsequent activity

The data flow activity outputs metrics regarding the number of rows written to each sink and rows read from each source. These results are returned in the `output` section of the activity run result. The metrics returned are in the format of the below json.

```
{
    "runStatus": {
        "metrics": {
            "<your sink name1>": {
                "rowsWritten": <number of rows written>,
                "sinkProcessingTime": <sink processing time in ms>,
                "sources": {
                    "<your source name1>": {
                        "rowsRead": <number of rows read>
                    },
                    "<your source name2>": {
                        "rowsRead": <number of rows read>
                    },
                    ...
                }
            },
            "<your sink name2>": {
                ...
            },
            ...
        }
    }
}
```

For example, to get to number of rows written to a sink named 'sink1' in an activity named 'dataflowActivity', use `@activity('dataflowActivity').output.runStatus.metrics.sink1.rowsWritten`.

To get the number of rows read from a source named 'source1' that was used in that sink, use `@activity('dataflowActivity').output.runStatus.metrics.sink1.sources.source1.rowsRead`.

> **NOTE**
> If a sink has zero rows written, it will not show up in metrics. Existence can be verified using the `contains` function. For example, `contains(activity('dataflowActivity').output.runStatus.metrics, 'sink1')` will check whether any rows were written to sink1.

Next steps

See supported control flow activities:

- If Condition Activity
- Execute Pipeline Activity
- For Each Activity
- Get Metadata Activity
- Lookup Activity
- Web Activity
- Until Activity

Transform data by running a Synapse notebook

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

The Azure Synapse notebook activity in a Synapse pipeline runs a Synapse notebook in your Azure Synapse workspace. This article builds on the data transformation activities article, which presents a general overview of data transformation and the supported transformation activities.

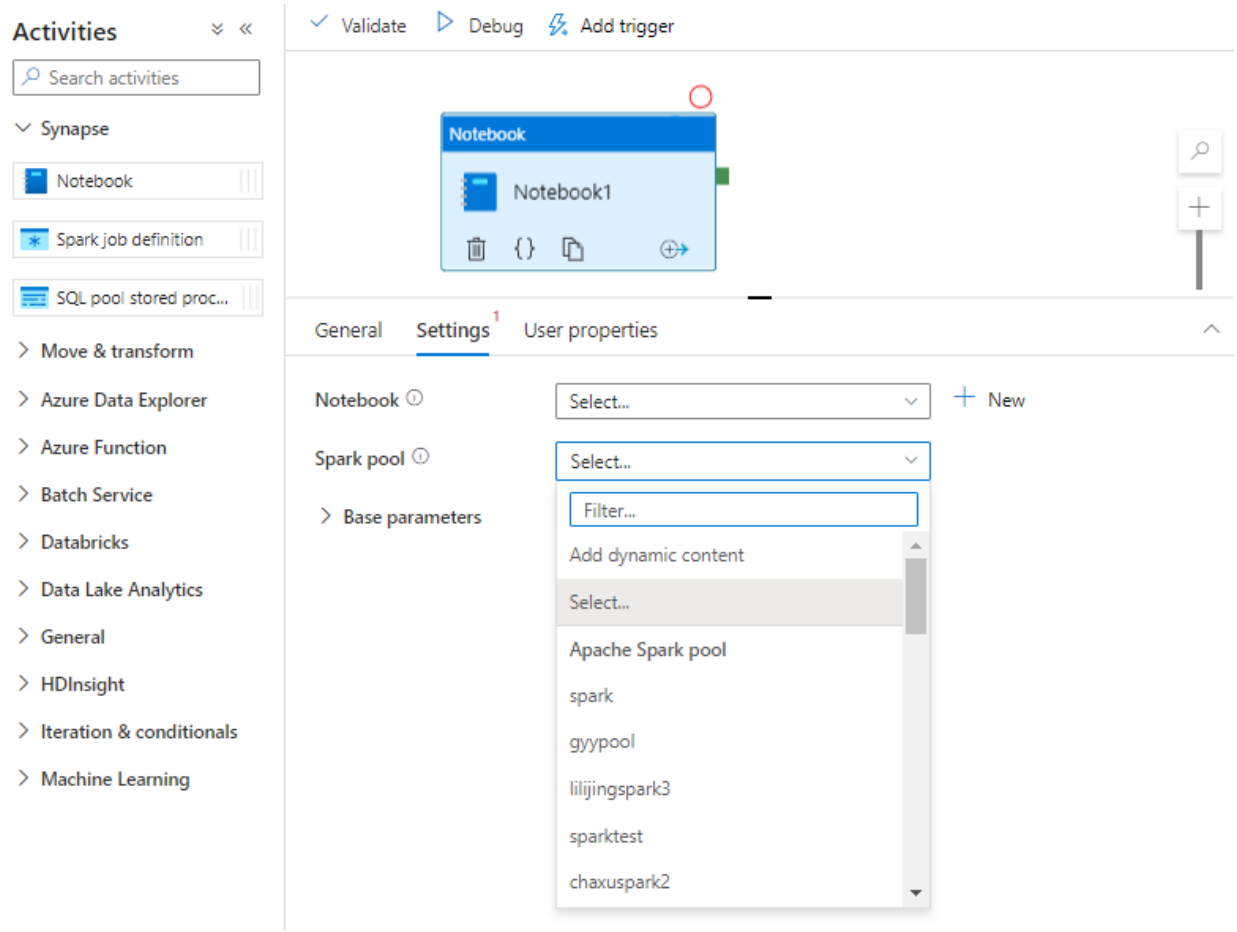
Create a Synapse notebook activity

You can create a Synapse notebook activity directly from the Synapse pipeline canvas or from the notebook editor. The Synapse notebook activity runs on the Spark pool that gets chosen in the Synapse notebook.

Add a Synapse notebook activity from pipeline canvas

Drag and drop **Synapse notebook** under **Activities** onto the Synapse pipeline canvas. Select on the Synapse notebook activity box and config the notebook content for current activity in the **settings**. You can select an existing notebook from the current workspace or add a new one.

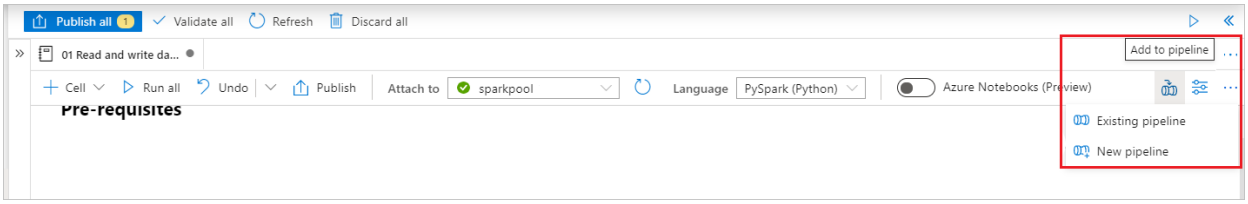
You can also select an Apache Spark pool in the settings. It should be noted that the Apache spark pool set here will replace the Apache spark pool used in the notebook. If Apache spark pool is not selected in the settings of notebook content for current activity, the Apache spark pool selected in that notebook will be used to run.



> **NOTE**
>
> The execution of parallel Spark Notebooks in Azure Synapse pipelines be queued and executed in a FIFO manner, jobs order in the queue is according to the time sequence, the expire time of a job in the queue is 3 days, please notice that queue for notebook only work in synapse pipeline.

Add a notebook to Synapse pipeline

Select the **Add to pipeline** button on the upper right corner to add a notebook to an existing pipeline or create a new pipeline.

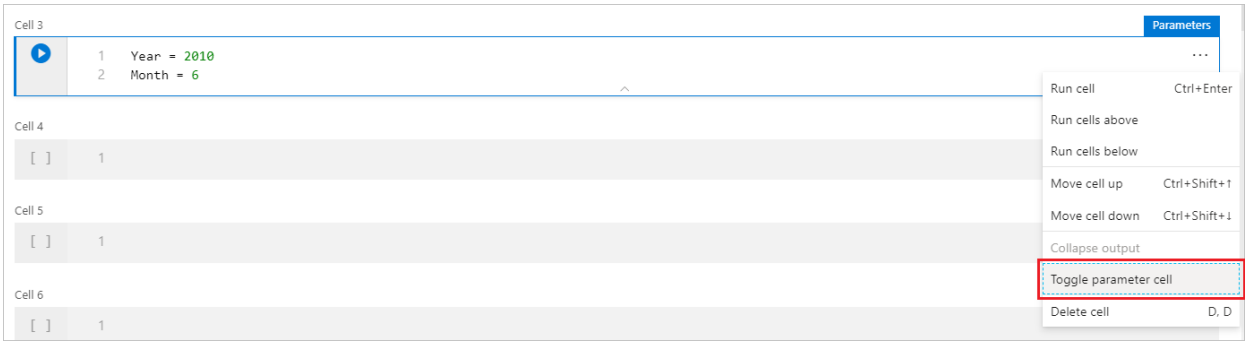


Passing parameters

Designate a parameters cell

- Classic notebook
- Preview notebook

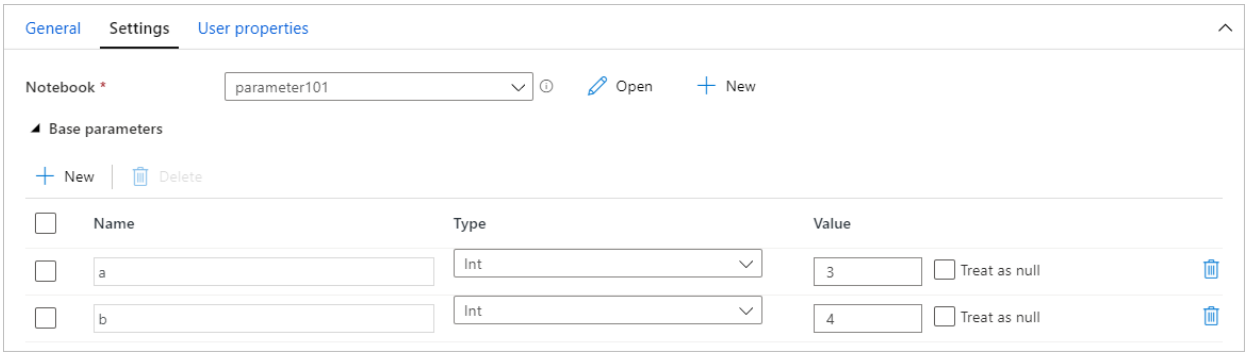
To parameterize your notebook, select the ellipses (...) to access the other cell actions menu at the far right. Then select **Toggle parameter cell** to designate the cell as the parameters cell.



Azure Data Factory looks for the parameters cell and uses the values as defaults for the parameters passed in at execution time. The execution engine will add a new cell beneath the parameters cell with input parameters to overwrite the default values.

Assign parameters values from a pipeline

Once you've created a notebook with parameters, you can execute it from a pipeline with the Synapse notebook activity. After you add the activity to your pipeline canvas, you'll be able to set the parameters values under **Base parameters** section on the **Settings** tab.



When assigning parameter values, you can use the pipeline expression language or system variables.

Read Synapse notebook cell output value

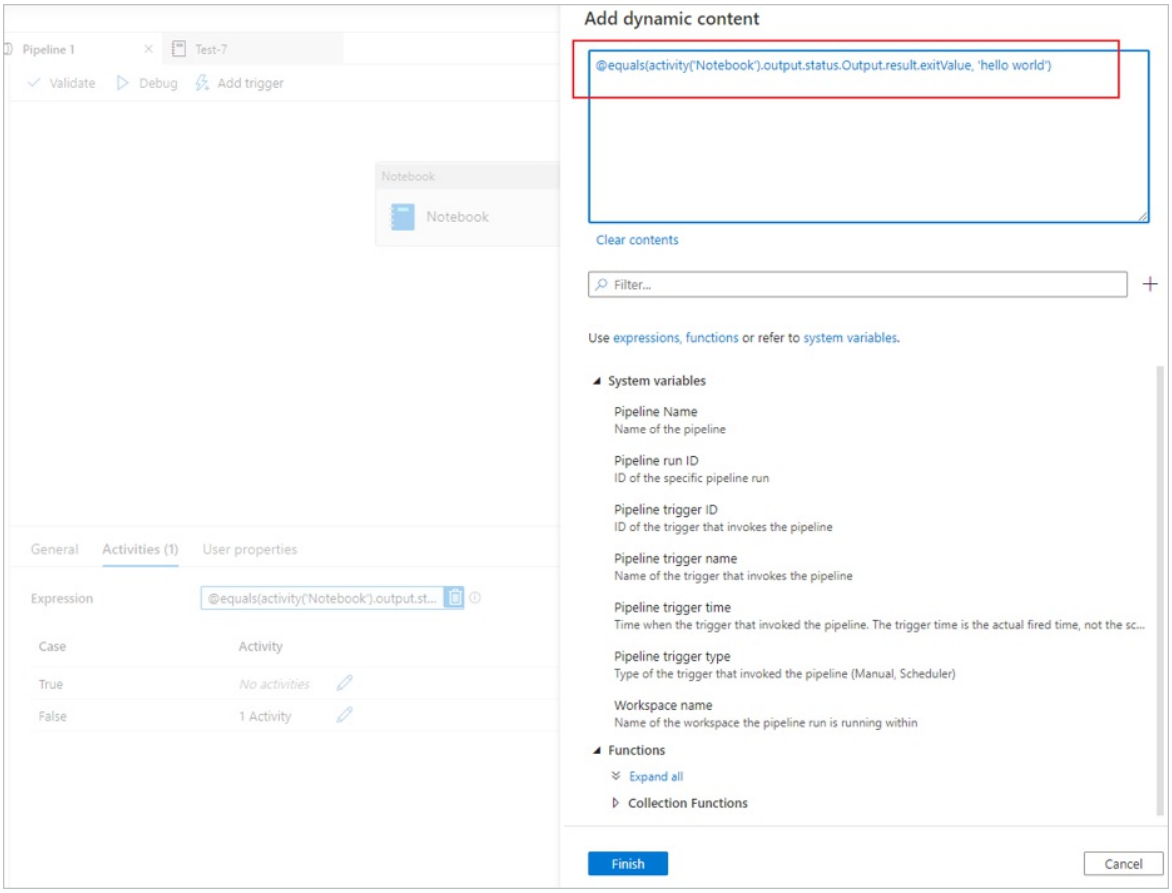
You can read the notebook cell output value in the next activities by following the steps below:

1. Call mssparkutils.notebook.exit API in your Synapse notebook activity to return the value that you want to show in activity output, for example:

   ```
   mssparkutils.notebook.exit("hello world")
   ```

   Saving the notebook content and retrigger the pipeline, the notebook activity output will contain the exitValue that can be consumed for the following activities in step 2.

2. Read exitValue property from notebook activity output. Here is a sample expression that is used to check whether the exitValue fetched from the notebook activity output equals "hello world":



Run another Synapse notebook

You can reference other notebooks in a Synapse notebook activity via calling %run magic or mssparkutils notebook utilities. Both support nesting function calls. The key differences of these two methods that you should consider based on your scenario are:

- %run magic copies all cells from the referenced notebook to the %run cell and shares the variable context. When notebook1 references notebook2 via `%run notebook2` and notebook2 calls a mssparkutils.notebook.exit function, the cell execution in notebook1 will be stopped. We recommend you use %run magic when you want to "include" a notebook file.
- mssparkutils notebook utilities calls the referenced notebook as a method or a function. The variable context isn't shared. When notebook1 references notebook2 via `mssparkutils.notebook.run("notebook2")` and notebook2 calls a mssparkutils.notebook.exit function, the cell execution in notebook1 will continue. We

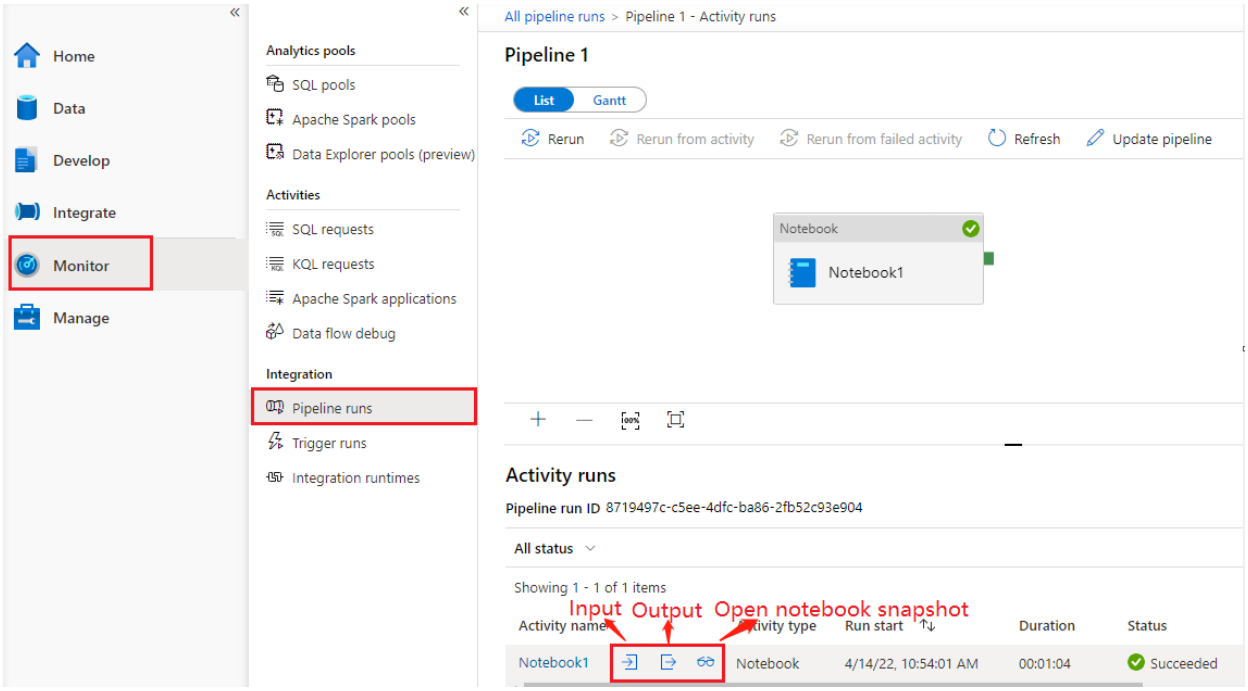
recommend you use mssparkutils notebook utilities when you want to "import" a notebook.

> **NOTE**
> Run another Synapse notebook from a Synapse pipeline will only work for a notebook with Preview enabled.

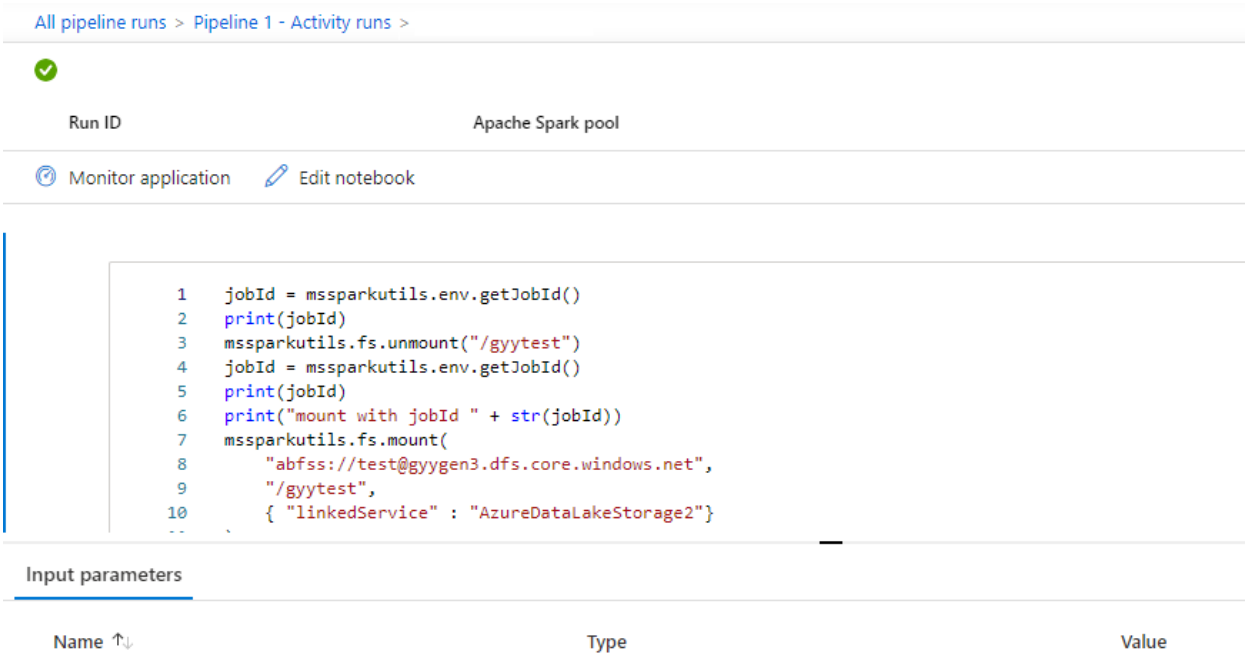
See notebook activity run history

Go to **Pipeline runs** under the **Monitor** tab, you'll see the pipeline you have triggered. Open the pipeline that contains notebook activity to see the run history.

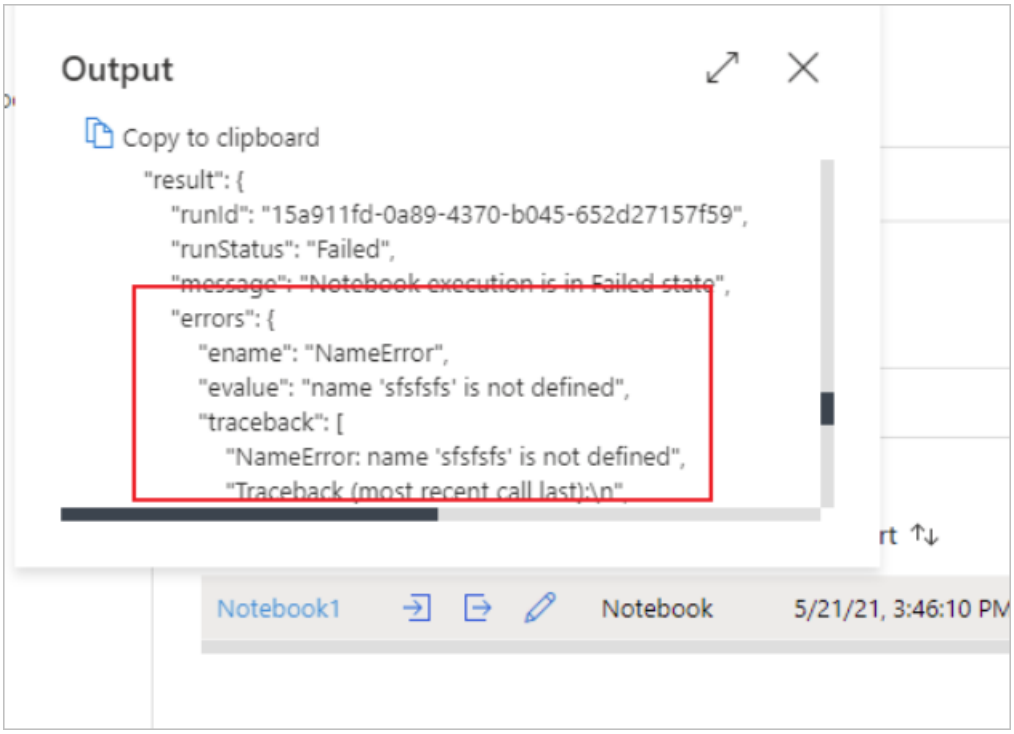
You can see the latest notebook run snapshot including both cells input and output by selecting the **open notebook** button.



Open notebook snapshot:



You can see the notebook activity input or output by selecting the **input** or **Output** button. If your pipeline failed with a user error, select the **output** to check the **result** field to see the detailed user error traceback.



Synapse notebook activity definition

Here's the sample JSON definition of a Synapse notebook activity:

```
{
    "name": "parameter_test",
    "type": "SynapseNotebook",
    "dependsOn": [],
    "policy": {
        "timeout": "7.00:00:00",
        "retry": 0,
        "retryIntervalInSeconds": 30,
        "secureOutput": false,
        "secureInput": false
    },
    "userProperties": [],
    "typeProperties": {
        "notebook": {
            "referenceName": "parameter_test",
            "type": "NotebookReference"
        },
        "parameters": {
            "input": {
                "value": {
                    "value": "@pipeline().parameters.input",
                    "type": "Expression"
                }
            }
        }
    }
}
```

Synapse notebook activity output

Here's the sample JSON of a Synapse notebook activity output:

```
{
    "status": {
        "Status": 1,
        "Output": {
            "status": <livySessionInfo>
            },
            "result": {
                "runId": "<GUID>",
                "runStatus": "Succeed",
                "message": "Notebook execution is in Succeeded state",
                "lastCheckedOn": "2021-03-23T00:40:10.6033333Z",
                "errors": {
                    "ename": "",
                    "evalue": ""
                },
                "sessionId": 4,
                "sparkpool": "sparkpool",
                "snapshotUrl": "https://myworkspace.dev.azuresynapse.net/notebooksnapshot/{guid}",
                "exitCode": "abc" // return value from user notebook via mssparkutils.notebook.exit("abc")
            }
        },
        "Error": null,
        "ExecutionDetails": {}
    },

    "effectiveIntegrationRuntime": "DefaultIntegrationRuntime (West US 2)",
    "executionDuration": 234,
    "durationInQueue": {
        "integrationRuntimeQueue": 0
    },
    "billingReference": {
        "activityType": "ExternalActivity",
        "billableDuration": [
            {
                "meterType": "AzureIR",
                "duration": 0.06666666666666667,
                "unit": "Hours"
            }
        ]
    }
}
```

Transform data by using SQL pool stored procedure activity in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

You use data transformation activities in a pipeline to transform and process raw data into predictions and insights. This article builds on the transform data article, which presents a general overview of data transformation and the supported transformation activities.

In Azure Synapse Analytics, you can use the SQL pool Stored Procedure Activity to invoke a stored procedure in a dedicated SQL pool.

Syntax details

The following settings are supported in SQL pool stored procedure activity:

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity	Yes
description	Text describing what the activity is used for	No
type	For SQL pool stored procedure activity, the activity type is **SqlPoolStoredProcedure**	Yes
sqlPool	Reference to a dedicated SQL pool in the current Azure Synapse workspace.	Yes
storedProcedureName	Specify the name of the stored procedure to invoke.	Yes
storedProcedureParameters	Specify the values for stored procedure parameters. Use `"param1": { "value": "param1Value","type":"param1Type" }` to pass parameter values and their type supported by the data source. If you need to pass null for a parameter, use `"param1": { "value": null }` (all lower case).	No

Example:

```
{
    "name": "SQLPoolStoredProcedureActivity",
    "description":"Description",
    "type": "SqlPoolStoredProcedure",
    "sqlPool": {
        "referenceName": "DedicatedSQLPool",
        "type": "SqlPoolReference"
    },
    "typeProperties": {
        "storedProcedureName": "usp_sample",
        "storedProcedureParameters": {
            "identifier": { "value": "1", "type": "Int" },
            "stringData": { "value": "str1" }

        }
    }
}
```

Next steps

- Pipeline and activity
- Dedicated SQL Pool

Azure Function activity in Azure Data Factory

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

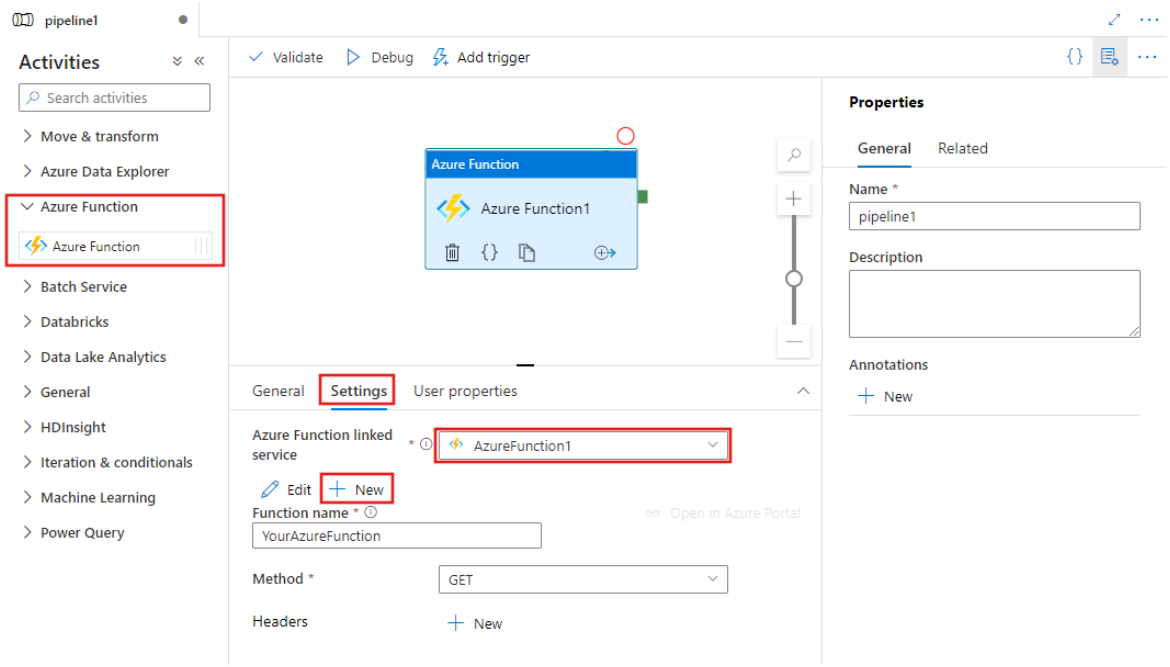
The Azure Function activity allows you to run Azure Functions in an Azure Data Factory or Synapse pipeline. To run an Azure Function, you must create a linked service connection. Then you can use the linked service with an activity that specifies the Azure Function that you plan to execute.

Create an Azure Function activity with UI

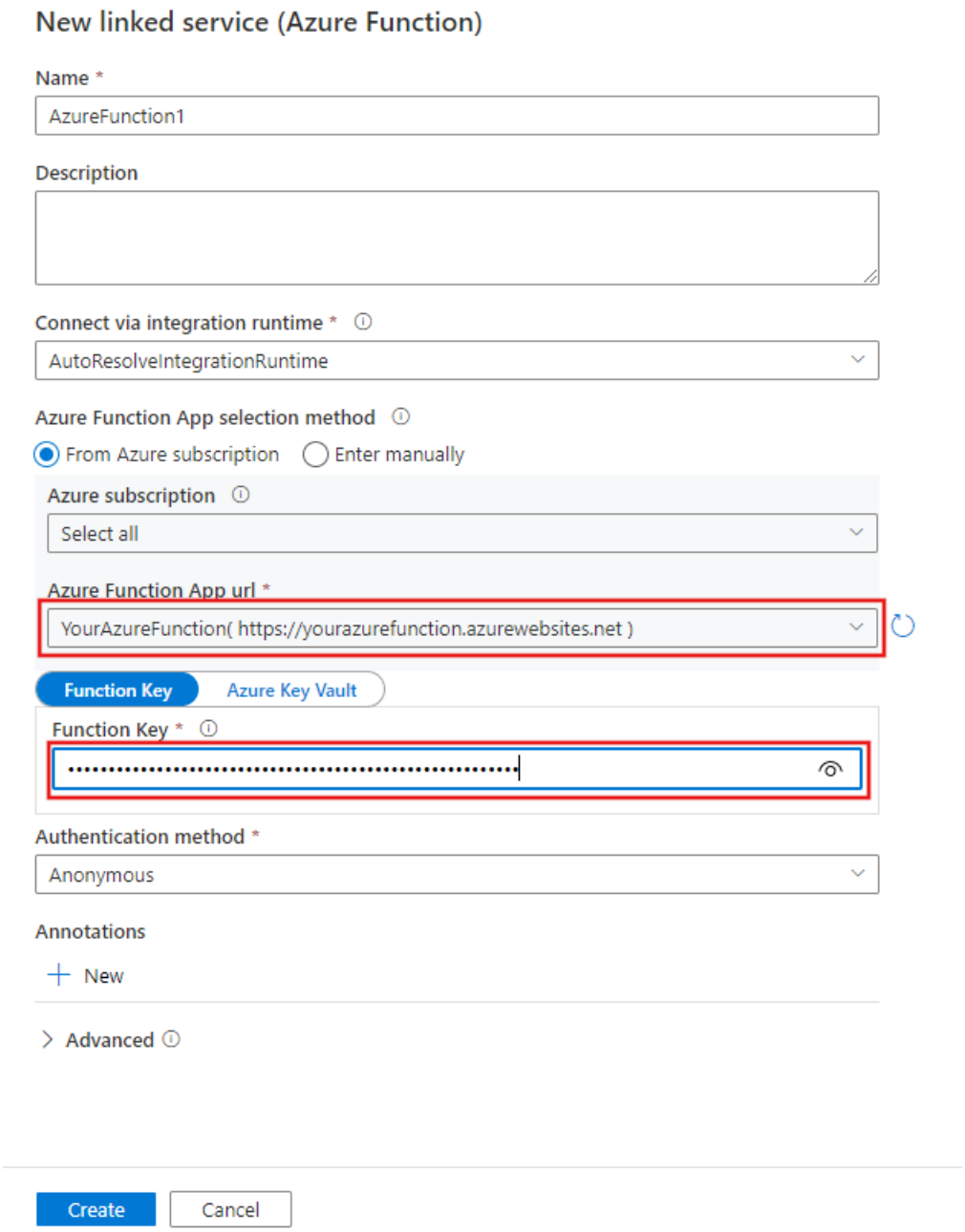
To use an Azure Function activity in a pipeline, complete the following steps:

1. Expand the Azure Function section of the pipeline Activities pane, and drag an Azure Function activity to the pipeline canvas.

2. Select the new Azure Function activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



3. If you do not already have an Azure Function linked service defined, select New to create a new one. In the new Azure Function linked service pane, choose your existing Azure Function App url and provide a Function Key.



4. After selecting the Azure Function linked service, provide the function name and other details to complete the configuration.

Azure Function linked service

The return type of the Azure function has to be a valid `JObject`. (Keep in mind that JArray is *not* a `JObject`.) Any return type other than `JObject` fails and raises the user error *Response Content is not a valid JObject*.

Function Key provides secure access to function name with each one having separate unique keys or master key within a function app. Managed identity provides secure access to the entire function app. User needs to provide key to access function name. For more information, see the function documentation for more details about Function access key

PROPERTY	DESCRIPTION	REQUIRED
Type	The type property must be set to: **AzureFunction**	Yes

PROPERTY	DESCRIPTION	REQUIRED
Function app url	URL for the Azure Function App. Format is `https://<accountname>.azurewebsites.net`. This URL is the value under **URL** section when viewing your Function App in the Azure portal	Yes
Function key	Access key for the Azure Function. Click on the **Manage** section for the respective function, and copy either the **Function Key** or the **Host key**. Find out more here: Azure Functions HTTP triggers and bindings	Yes

Azure Function activity

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
Name	Name of the activity in the pipeline	String	Yes
Type	Type of activity is 'AzureFunctionActivity'	String	Yes
Linked service	The Azure Function linked service for the corresponding Azure Function App	Linked service reference	Yes
Function name	Name of the function in the Azure Function App that this activity calls	String	Yes
Method	REST API method for the function call	String Supported Types: "GET", "POST", "PUT"	Yes
Header	Headers that are sent to the request. For example, to set the language and type on a request: "headers": { "Accept-Language": "en-us", "Content-Type": "application/json" }	String (or expression with resultType of string)	No
Body	Body that is sent along with the request to the function api method	String (or expression with resultType of string) or object.	Required for PUT/POST methods

See the schema of the request payload in Request payload schema section.

Routing and queries

The Azure Function Activity supports **routing**. For example, if your Azure Function has the endpoint `https://functionAPP.azurewebsites.net/api/<functionName>/<value>?code=<secret>`, then the `functionName` to use in the Azure Function Activity is `<functionName>/<value>`. You can parameterize this function to provide the desired `functionName` at runtime.

> **NOTE**
> The `functionName` for Durable Functions should be taken from the **route** property of the function's binding in its JSON definition, to include its routing information. Simply using the `functionName` without the route detail included will result in a failure because the Function App cannot be found.

The Azure Function Activity also supports **queries**. A query must be included as part of the `functionName`. For example, when the function name is `HttpTriggerCSharp` and the query that you want to include is `name=hello`, then you can construct the `functionName` in the Azure Function Activity as `HttpTriggerCSharp?name=hello`. This function can be parameterized so the value can be determined at runtime.

Timeout and long running functions

Azure Functions times out after 230 seconds regardless of the `functionTimeout` setting you've configured in the settings. For more information, see this article. To work around this behavior, follow an async pattern or use Durable Functions. The benefit of Durable Functions is that they offer their own state-tracking mechanism, so you don't need to implement your own state-tracking.

Learn more about Durable Functions in this article. You can set up an Azure Function Activity to call the Durable Function, which will return a response with a different URI, such as this example. Because `statusQueryGetUri` returns HTTP Status 202 while the function is running, you can poll the status of the function by using a Web Activity. Set up a Web Activity with the `url` field set to `@activity('<AzureFunctionActivityName>').output.statusQueryGetUri`. When the Durable Function completes, the output of the function will be the output of the Web Activity.

Sample

You can find a sample that uses an Azure Function to extract the content of a tar file here.

Next steps

Learn more about supported activities in Pipelines and activities.

Transform data by running a Jar activity in Azure Databricks

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

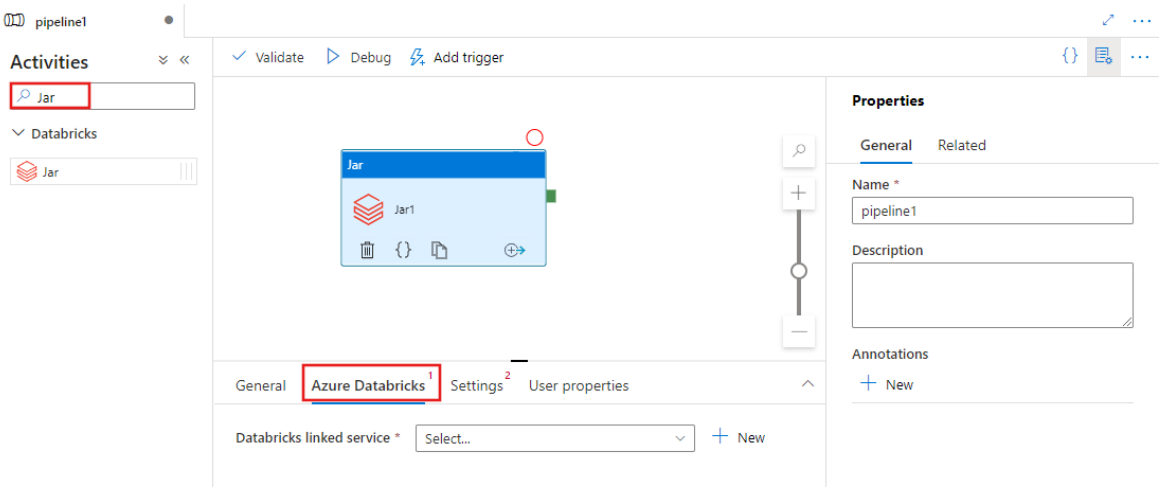
The Azure Databricks Jar Activity in a pipeline runs a Spark Jar in your Azure Databricks cluster. This article builds on the data transformation activities article, which presents a general overview of data transformation and the supported transformation activities. Azure Databricks is a managed platform for running Apache Spark.

For an eleven-minute introduction and demonstration of this feature, watch the following video:

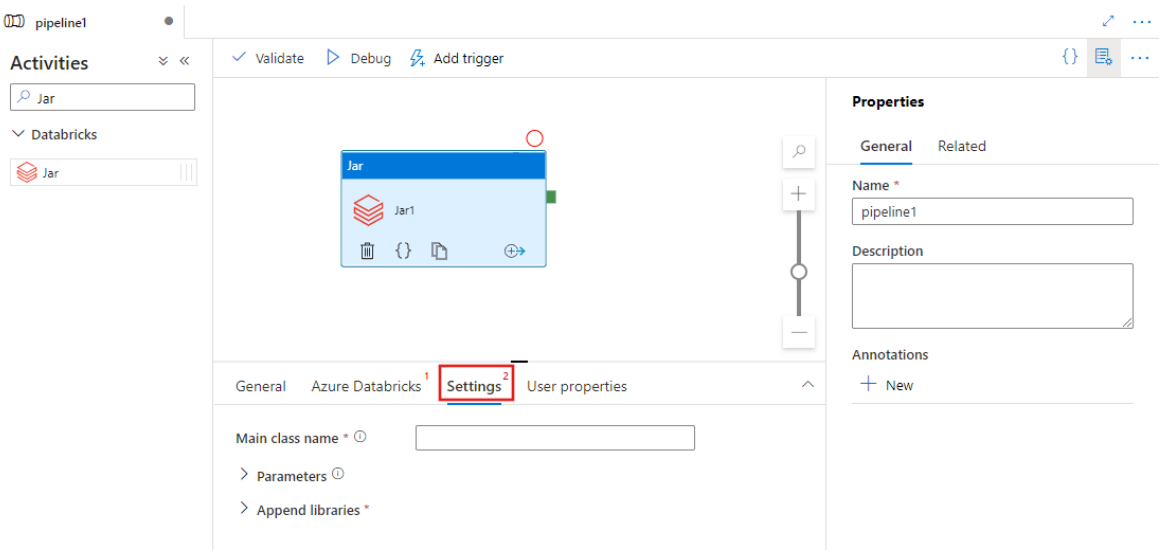
Add a Jar activity for Azure Databricks to a pipeline with UI

To use a Jar activity for Azure Databricks in a pipeline, complete the following steps:

1. Search for *Jar* in the pipeline Activities pane, and drag a Jar activity to the pipeline canvas.
2. Select the new Jar activity on the canvas if it is not already selected.
3. Select the **Azure Databricks** tab to select or create a new Azure Databricks linked service that will execute the Jar activity.



4. Select the **Settings** tab and specify a class name to be executed on Azure Databricks, optional parameters to be passed to the Jar, and libraries to be installed on the cluster to execute the job.



Databricks Jar activity definition

Here's the sample JSON definition of a Databricks Jar Activity:

```
{
    "name": "SparkJarActivity",
    "type": "DatabricksSparkJar",
    "linkedServiceName": {
        "referenceName": "AzureDatabricks",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "mainClassName": "org.apache.spark.examples.SparkPi",
        "parameters": [ "10" ],
        "libraries": [
            {
                "jar": "dbfs:/docs/sparkpi.jar"
            }
        ]
    }
}
```

Databricks Jar activity properties

The following table describes the JSON properties used in the JSON definition:

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity in the pipeline.	Yes
description	Text describing what the activity does.	No
type	For Databricks Jar Activity, the activity type is DatabricksSparkJar.	Yes
linkedServiceName	Name of the Databricks Linked Service on which the Jar activity runs. To learn about this linked service, see Compute linked services article.	Yes

PROPERTY	DESCRIPTION	REQUIRED
mainClassName	The full name of the class containing the main method to be executed. This class must be contained in a JAR provided as a library. A JAR file can contain multiple classes. Each of the classes can contain a main method.	Yes
parameters	Parameters that will be passed to the main method. This property is an array of strings.	No
libraries	A list of libraries to be installed on the cluster that will execute the job. It can be an array of <string, object>	Yes (at least one containing the mainClassName method)

NOTE

Known Issue - When using the same Interactive cluster for running concurrent Databricks Jar activities (without cluster restart), there is a known issue in Databricks where in parameters of the 1st activity will be used by following activities as well. Hence resulting to incorrect parameters being passed to the subsequent jobs. To mitigate this use a Job cluster instead.

Supported libraries for databricks activities

In the previous Databricks activity definition, you specified these library types: `jar`, `egg`, `maven`, `pypi`, `cran`.

```
{
    "libraries": [
        {
            "jar": "dbfs:/mnt/libraries/library.jar"
        },
        {
            "egg": "dbfs:/mnt/libraries/library.egg"
        },
        {
            "maven": {
                "coordinates": "org.jsoup:jsoup:1.7.2",
                "exclusions": [ "slf4j:slf4j" ]
            }
        },
        {
            "pypi": {
                "package": "simplejson",
                "repo": "http://my-pypi-mirror.com"
            }
        },
        {
            "cran": {
                "package": "ada",
                "repo": "https://cran.us.r-project.org"
            }
        }
    ]
}
```

For more information, see the Databricks documentation for library types.

How to upload a library in Databricks

You can use the Workspace UI:

1. Use the Databricks workspace UI
2. To obtain the dbfs path of the library added using UI, you can use Databricks CLI.

   Typically the Jar libraries are stored under dbfs:/FileStore/jars while using the UI. You can list all through the CLI: *databricks fs ls dbfs:/FileStore/job-jars*

Or you can use the Databricks CLI:

1. Follow Copy the library using Databricks CLI
2. Use Databricks CLI (installation steps)

   As an example, to copy a JAR to dbfs: `dbfs cp SparkPi-assembly-0.1.jar dbfs:/docs/sparkpi.jar`

Next steps

For an eleven-minute introduction and demonstration of this feature, watch the video.

Transform data by running a Databricks notebook

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

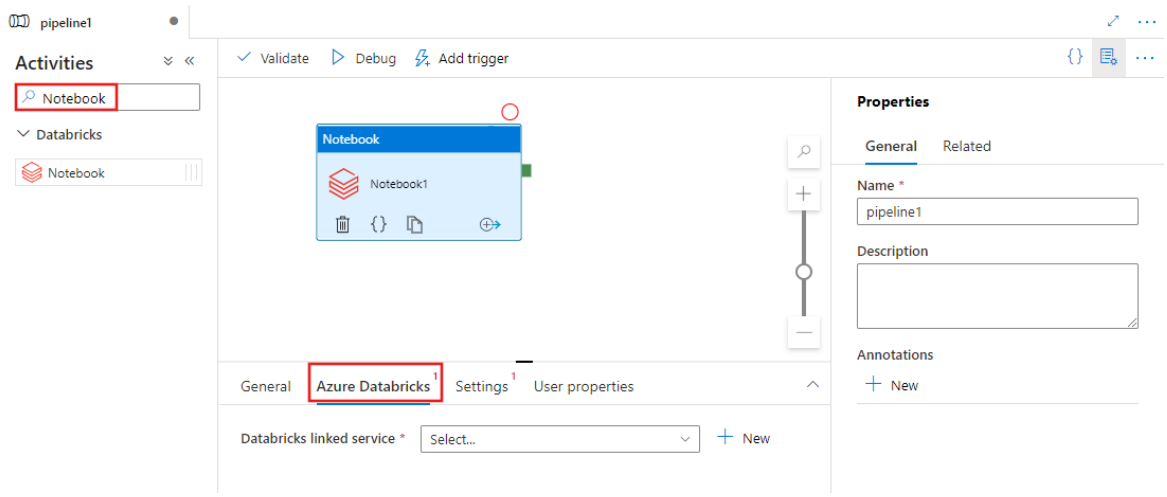
The Azure Databricks Notebook Activity in a pipeline runs a Databricks notebook in your Azure Databricks workspace. This article builds on the data transformation activities article, which presents a general overview of data transformation and the supported transformation activities. Azure Databricks is a managed platform for running Apache Spark.

You can create a Databricks notebook with an ARM template using JSON, or directly through the Azure Data Factory Studio user interface. For a step-by-step walkthrough of how to create a Databricks notebook activity using the user interface, reference the tutorial Run a Databricks notebook with the Databricks Notebook Activity in Azure Data Factory.

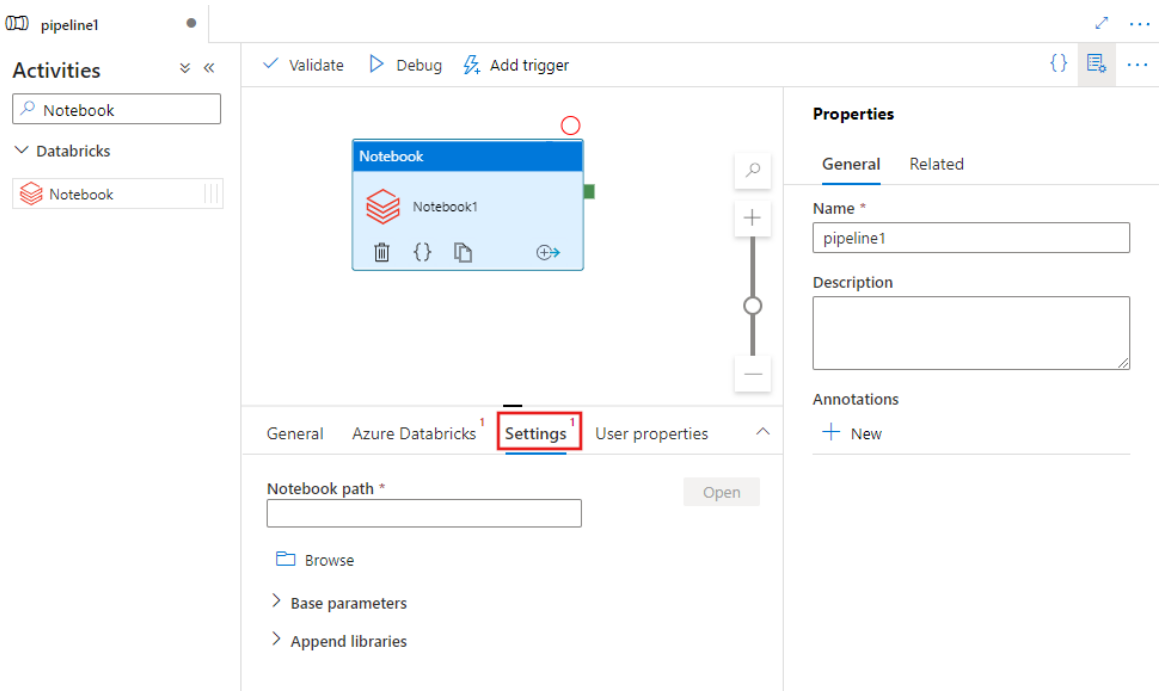
Add a Notebook activity for Azure Databricks to a pipeline with UI

To use a Notebook activity for Azure Databricks in a pipeline, complete the following steps:

1. Search for *Notebook* in the pipeline Activities pane, and drag a Notebook activity to the pipeline canvas.
2. Select the new Notebook activity on the canvas if it is not already selected.
3. Select the **Azure Databricks** tab to select or create a new Azure Databricks linked service that will execute the Notebook activity.



4. Select the **Settings** tab and specify the notebook path to be executed on Azure Databricks, optional base parameters to be passed to the notebook, and any additional libraries to be installed on the cluster to execute the job.



Databricks Notebook activity definition

Here is the sample JSON definition of a Databricks Notebook Activity:

```
{
    "activity": {
        "name": "MyActivity",
        "description": "MyActivity description",
        "type": "DatabricksNotebook",
        "linkedServiceName": {
            "referenceName": "MyDatabricksLinkedservice",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "notebookPath": "/Users/user@example.com/ScalaExampleNotebook",
            "baseParameters": {
                "inputpath": "input/folder1/",
                "outputpath": "output/"
            },
            "libraries": [
                {
                "jar": "dbfs:/docs/library.jar"
                }
            ]
        }
    }
}
```

Databricks Notebook activity properties

The following table describes the JSON properties used in the JSON definition:

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity in the pipeline.	Yes
description	Text describing what the activity does.	No

PROPERTY	DESCRIPTION	REQUIRED
type	For Databricks Notebook Activity, the activity type is DatabricksNotebook.	Yes
linkedServiceName	Name of the Databricks Linked Service on which the Databricks notebook runs. To learn about this linked service, see Compute linked services article.	Yes
notebookPath	The absolute path of the notebook to be run in the Databricks Workspace. This path must begin with a slash.	Yes
baseParameters	An array of Key-Value pairs. Base parameters can be used for each activity run. If the notebook takes a parameter that is not specified, the default value from the notebook will be used. Find more on parameters in Databricks Notebooks.	No
libraries	A list of libraries to be installed on the cluster that will execute the job. It can be an array of <string, object>.	No

Supported libraries for Databricks activities

In the above Databricks activity definition, you specify these library types: *jar*, *egg*, *whl*, *maven*, *pypi*, *cran*.

```
{
    "libraries": [
        {
            "jar": "dbfs:/mnt/libraries/library.jar"
        },
        {
            "egg": "dbfs:/mnt/libraries/library.egg"
        },
        {
            "whl": "dbfs:/mnt/libraries/mlflow-0.0.1.dev0-py2-none-any.whl"
        },
        {
            "whl": "dbfs:/mnt/libraries/wheel-libraries.wheelhouse.zip"
        },
        {
            "maven": {
                "coordinates": "org.jsoup:jsoup:1.7.2",
                "exclusions": [ "slf4j:slf4j" ]
            }
        },
        {
            "pypi": {
                "package": "simplejson",
                "repo": "http://my-pypi-mirror.com"
            }
        },
        {
            "cran": {
                "package": "ada",
                "repo": "https://cran.us.r-project.org"
            }
        }
    ]
}
```

For more information, see the Databricks documentation for library types.

Passing parameters between notebooks and pipelines

You can pass parameters to notebooks using *baseParameters* property in databricks activity.

In certain cases, you might require to pass back certain values from notebook back to the service, which can be used for control flow (conditional checks) in the service or be consumed by downstream activities (size limit is 2 MB).

1. In your notebook, you may call dbutils.notebook.exit("returnValue") and corresponding "returnValue" will be returned to the service.

2. You can consume the output in the service by using expression such as `@{activity('databricks notebook activity name').output.runOutput}`.

> **IMPORTANT**
>
> If you are passing JSON object you can retrieve values by appending property names. Example:
> `@{activity('databricks notebook activity name').output.runOutput.PropertyName}`

How to upload a library in Databricks

You can use the Workspace UI:

1. Use the Databricks workspace UI
2. To obtain the dbfs path of the library added using UI, you can use Databricks CLI.

   Typically the Jar libraries are stored under dbfs:/FileStore/jars while using the UI. You can list all through the CLI: *databricks fs ls dbfs:/FileStore/job-jars*

Or you can use the Databricks CLI:

1. Follow Copy the library using Databricks CLI
2. Use Databricks CLI (installation steps)

   As an example, to copy a JAR to dbfs: `dbfs cp SparkPi-assembly-0.1.jar dbfs:/docs/sparkpi.jar`

Transform data by running a Python activity in Azure Databricks

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

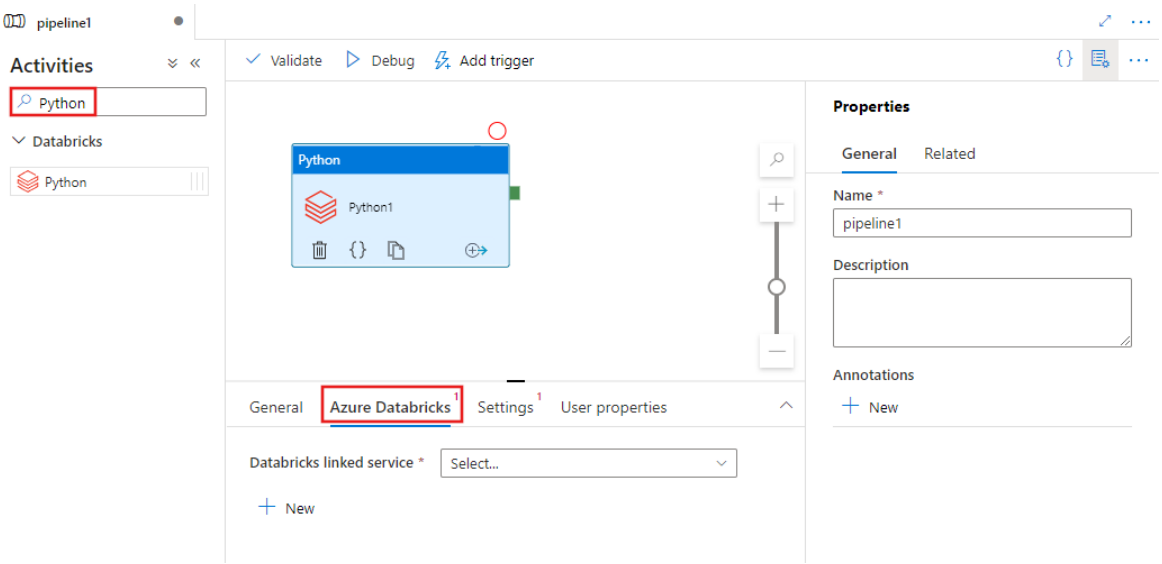
The Azure Databricks Python Activity in a pipeline runs a Python file in your Azure Databricks cluster. This article builds on the data transformation activities article, which presents a general overview of data transformation and the supported transformation activities. Azure Databricks is a managed platform for running Apache Spark.

For an eleven-minute introduction and demonstration of this feature, watch the following video:

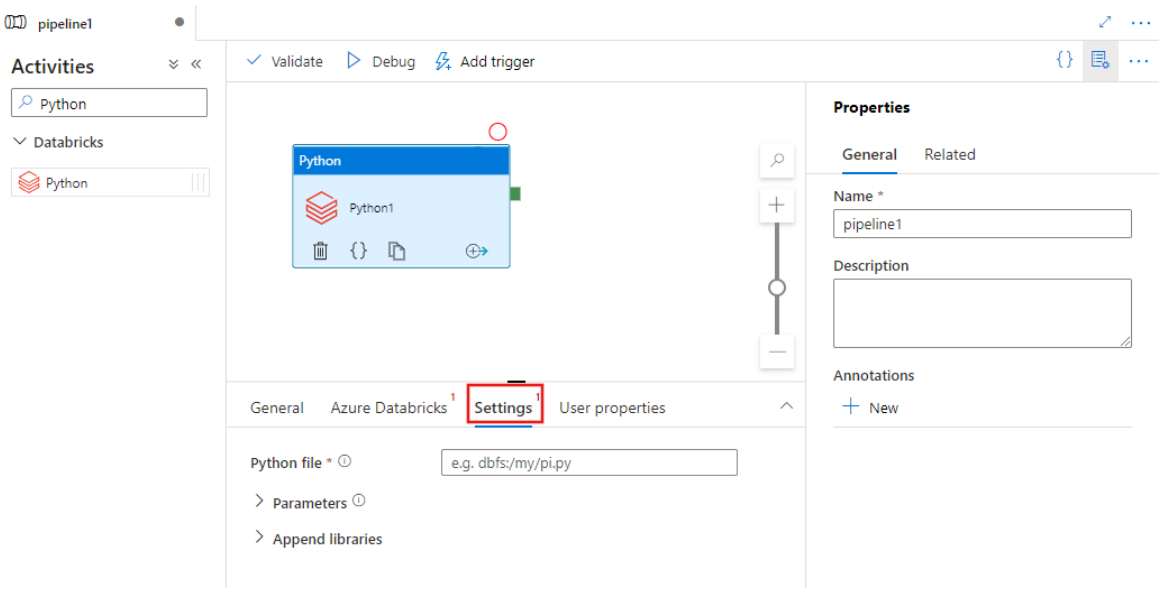
Add a Python activity for Azure Databricks to a pipeline with UI

To use a Python activity for Azure Databricks in a pipeline, complete the following steps:

1. Search for *Python* in the pipeline Activities pane, and drag a Python activity to the pipeline canvas.
2. Select the new Python activity on the canvas if it is not already selected.
3. Select the **Azure Databricks** tab to select or create a new Azure Databricks linked service that will execute the Python activity.



4. Select the **Settings** tab and specify the path within Azure Databricks to a Python file to be executed, optional parameters to be passed, and any additional libraries to be installed on the cluster to execute the job.



Databricks Python activity definition

Here is the sample JSON definition of a Databricks Python Activity:

```
{
    "activity": {
        "name": "MyActivity",
        "description": "MyActivity description",
        "type": "DatabricksSparkPython",
        "linkedServiceName": {
            "referenceName": "MyDatabricksLinkedService",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "pythonFile": "dbfs:/docs/pi.py",
            "parameters": [
                "10"
            ],
            "libraries": [
                {
                    "pypi": {
                        "package": "tensorflow"
                    }
                }
            ]
        }
    }
}
```

Databricks Python activity properties

The following table describes the JSON properties used in the JSON definition:

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity in the pipeline.	Yes
description	Text describing what the activity does.	No
type	For Databricks Python Activity, the activity type is DatabricksSparkPython.	Yes

PROPERTY	DESCRIPTION	REQUIRED
linkedServiceName	Name of the Databricks Linked Service on which the Python activity runs. To learn about this linked service, see Compute linked services article.	Yes
pythonFile	The URI of the Python file to be executed. Only DBFS paths are supported.	Yes
parameters	Command line parameters that will be passed to the Python file. This is an array of strings.	No
libraries	A list of libraries to be installed on the cluster that will execute the job. It can be an array of <string, object>	No

Supported libraries for databricks activities

In the above Databricks activity definition you specify these library types: *jar*, *egg*, *maven*, *pypi*, *cran*.

```
{
    "libraries": [
        {
            "jar": "dbfs:/mnt/libraries/library.jar"
        },
        {
            "egg": "dbfs:/mnt/libraries/library.egg"
        },
        {
            "maven": {
                "coordinates": "org.jsoup:jsoup:1.7.2",
                "exclusions": [ "slf4j:slf4j" ]
            }
        },
        {
            "pypi": {
                "package": "simplejson",
                "repo": "http://my-pypi-mirror.com"
            }
        },
        {
            "cran": {
                "package": "ada",
                "repo": "https://cran.us.r-project.org"
            }
        }
    ]
}
```

For more details refer Databricks documentation for library types.

How to upload a library in Databricks

You can use the Workspace UI:

1. Use the Databricks workspace UI

2. To obtain the dbfs path of the library added using UI, you can use Databricks CLI.

   Typically the Jar libraries are stored under dbfs:/FileStore/jars while using the UI. You can list all through the CLI: *databricks fs ls dbfs:/FileStore/job-jars*

Or you can use the Databricks CLI:

1. Follow Copy the library using Databricks CLI
2. Use Databricks CLI (installation steps)

   As an example, to copy a JAR to dbfs: `dbfs cp SparkPi-assembly-0.1.jar dbfs:/docs/sparkpi.jar`

Process data by running U-SQL scripts on Azure Data Lake Analytics with Azure Data Factory and Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

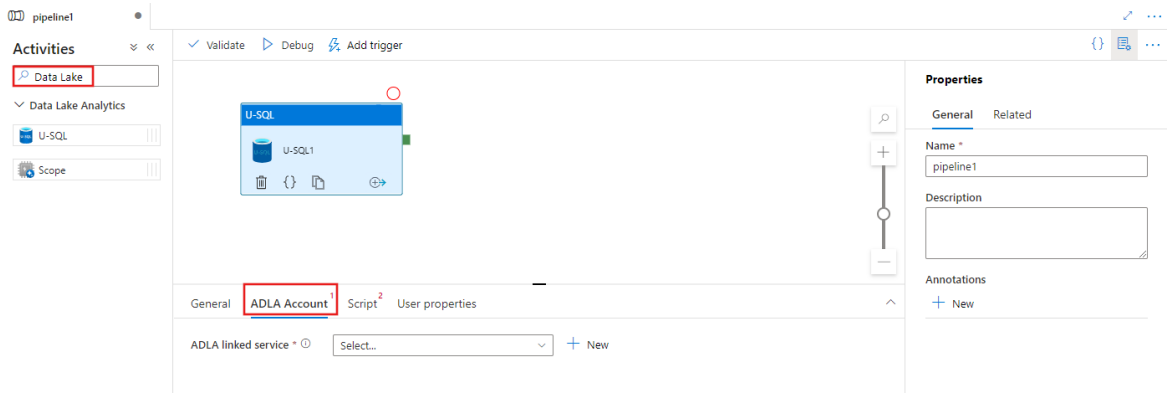
A pipeline in an Azure Data Factory or Synapse Analytics workspace processes data in linked storage services by using linked compute services. It contains a sequence of activities where each activity performs a specific processing operation. This article describes the **Data Lake Analytics U-SQL Activity** that runs a **U-SQL** script on an **Azure Data Lake Analytics** compute linked service.

Create an Azure Data Lake Analytics account before creating a pipeline with a Data Lake Analytics U-SQL Activity. To learn about Azure Data Lake Analytics, see Get started with Azure Data Lake Analytics.

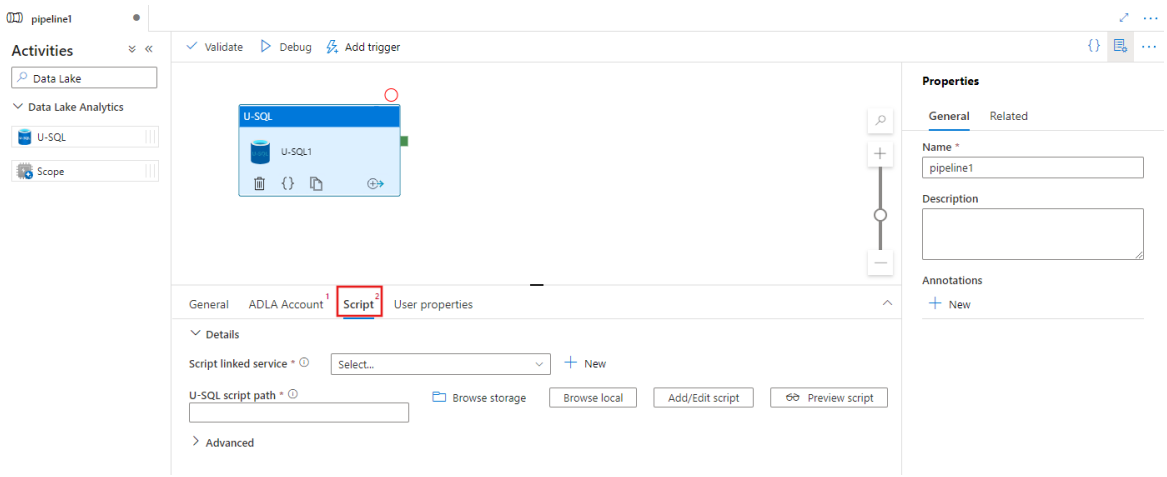
Add a U-SQL activity for Azure Data Lake Analytics to a pipeline with UI

To use a U-SQL activity for Azure Data Lake Analytics in a pipeline, complete the following steps:

1. Search for *Data Lake* in the pipeline Activities pane, and drag a U-SQL activity to the pipeline canvas.
2. Select the new U-SQL activity on the canvas if it is not already selected.
3. Select the **ADLA Account** tab to select or create a new Azure Data Lake Analytics linked service that will be used to execute the U-SQL activity.



4. Select the **Script** tab to select or create a new storage linked service, and a path within the storage location, which will host the script.



Azure Data Lake Analytics linked service

You create an **Azure Data Lake Analytics** linked service to link an Azure Data Lake Analytics compute service to an Azure Data Factory or Synapse Analytics workspace. The Data Lake Analytics U-SQL activity in the pipeline refers to this linked service.

The following table provides descriptions for the generic properties used in the JSON definition.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property should be set to: **AzureDataLakeAnalytics**.	Yes
accountName	Azure Data Lake Analytics Account Name.	Yes
dataLakeAnalyticsUri	Azure Data Lake Analytics URI.	No
subscriptionId	Azure subscription ID	No
resourceGroupName	Azure resource group name	No

Service principal authentication

The Azure Data Lake Analytics linked service requires a service principal authentication to connect to the Azure Data Lake Analytics service. To use service principal authentication, register an application entity in Azure Active Directory (Azure AD) and grant it the access to both the Data Lake Analytics and the Data Lake Store it uses. For detailed steps, see Service-to-service authentication. Make note of the following values, which you use to define the linked service:

- Application ID
- Application key
- Tenant ID

Grant service principal permission to your Azure Data Lake Analytics using the Add User Wizard.

Use service principal authentication by specifying the following properties:

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalId	Specify the application's client ID.	Yes

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalKey	Specify the application's key.	Yes
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. You can retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Yes

Example: Service principal authentication

```
{
    "name": "AzureDataLakeAnalyticsLinkedService",
    "properties": {
        "type": "AzureDataLakeAnalytics",
        "typeProperties": {
            "accountName": "<account name>",
            "dataLakeAnalyticsUri": "<azure data lake analytics URI>",
            "servicePrincipalId": "<service principal id>",
            "servicePrincipalKey": {
                "value": "<service principal key>",
                "type": "SecureString"
            },
            "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>",
            "subscriptionId": "<optional, subscription id of ADLA>",
            "resourceGroupName": "<optional, resource group name of ADLA>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

To learn more about the linked service, see Compute linked services.

Data Lake Analytics U-SQL Activity

The following JSON snippet defines a pipeline with a Data Lake Analytics U-SQL Activity. The activity definition has a reference to the Azure Data Lake Analytics linked service you created earlier. To execute a Data Lake Analytics U-SQL script, the service submits the script you specified to the Data Lake Analytics, and the required inputs and outputs is defined in the script for Data Lake Analytics to fetch and output.

```
{
    "name": "ADLA U-SQL Activity",
    "description": "description",
    "type": "DataLakeAnalyticsU-SQL",
    "linkedServiceName": {
        "referenceName": "<linked service name of Azure Data Lake Analytics>",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "scriptLinkedService": {
            "referenceName": "<linked service name of Azure Data Lake Store or Azure Storage which contains
the U-SQL script>",
            "type": "LinkedServiceReference"
        },
        "scriptPath": "scripts\\kona\\SearchLogProcessing.txt",
        "degreeOfParallelism": 3,
        "priority": 100,
        "parameters": {
            "in": "/datalake/input/SearchLog.tsv",
            "out": "/datalake/output/Result.tsv"
        }
    }
}
```

The following table describes names and descriptions of properties that are specific to this activity.

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity in the pipeline	Yes
description	Text describing what the activity does.	No
type	For Data Lake Analytics U-SQL activity, the activity type is **DataLakeAnalyticsU-SQL**.	Yes
linkedServiceName	Linked Service to Azure Data Lake Analytics. To learn about this linked service, see Compute linked services article.	Yes
scriptPath	Path to folder that contains the U-SQL script. Name of the file is case-sensitive.	Yes
scriptLinkedService	Linked service that links the **Azure Data Lake Store** or **Azure Storage** that contains the script	Yes
degreeOfParallelism	The maximum number of nodes simultaneously used to run the job.	No
priority	Determines which jobs out of all that are queued should be selected to run first. The lower the number, the higher the priority.	No
parameters	Parameters to pass into the U-SQL script.	No

PROPERTY	DESCRIPTION	REQUIRED
runtimeVersion	Runtime version of the U-SQL engine to use.	No
compilationMode	Compilation mode of U-SQL. Must be one of these values: **Semantic**: Only perform semantic checks and necessary sanity checks, **Full**: Perform the full compilation, including syntax check, optimization, code generation, etc., **SingleBox**: Perform the full compilation, with TargetType setting to SingleBox. If you don't specify a value for this property, the server determines the optimal compilation mode.	No

See SearchLogProcessing.txt for the script definition.

Sample U-SQL script

```
@searchlog =
    EXTRACT UserId          int,
            Start           DateTime,
            Region          string,
            Query           string,
            Duration        int,
            Urls            string,
            ClickedUrls     string
    FROM @in
    USING Extractors.Tsv(nullEscape:"#NULL#");

@rs1 =
    SELECT Start, Region, Duration
    FROM @searchlog
WHERE Region == "en-gb";

@rs1 =
    SELECT Start, Region, Duration
    FROM @rs1
    WHERE Start <= DateTime.Parse("2012/02/19");

OUTPUT @rs1
    TO @out
      USING Outputters.Tsv(quoting:false, dateTimeFormat:null);
```

In above script example, the input and output to the script is defined in **@in** and **@out** parameters. The values for **@in** and **@out** parameters in the U-SQL script are passed dynamically by the service using the 'parameters' section.

You can specify other properties such as degreeOfParallelism and priority as well in your pipeline definition for the jobs that run on the Azure Data Lake Analytics service.

Dynamic parameters

In the sample pipeline definition, in and out parameters are assigned with hard-coded values.

```
"parameters": {
    "in": "/datalake/input/SearchLog.tsv",
    "out": "/datalake/output/Result.tsv"
}
```

It is possible to use dynamic parameters instead. For example:

```
"parameters": {
    "in": "/datalake/input/@{formatDateTime(pipeline().parameters.WindowStart,'yyyy/MM/dd')}/data.tsv",
    "out": "/datalake/output/@{formatDateTime(pipeline().parameters.WindowStart,'yyyy/MM/dd')}/result.tsv"
}
```

In this case, input files are still picked up from the /datalake/input folder and output files are generated in the /datalake/output folder. The file names are dynamic based on the window start time being passed in when pipeline gets triggered.

Next steps

See the following articles that explain how to transform data in other ways:

- Hive activity
- Pig activity
- MapReduce activity
- Hadoop Streaming activity
- Spark activity
- .NET custom activity
- Stored procedure activity

Transform data using Hadoop Hive activity in Azure Data Factory or Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

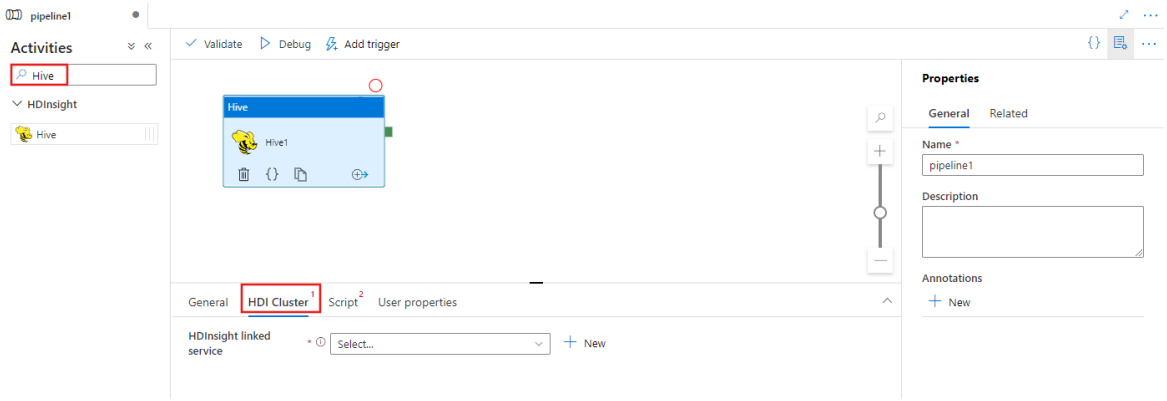
The HDInsight Hive activity in an Azure Data Factory or Synapse Analytics pipeline executes Hive queries on your own or on-demand HDInsight cluster. This article builds on the data transformation activities article, which presents a general overview of data transformation and the supported transformation activities.

If you are new to Azure Data Factory and Synapse Analytics, read through the introduction articles for Azure Data Factory or Synapse Analytics, and do the Tutorial: transform data before reading this article.

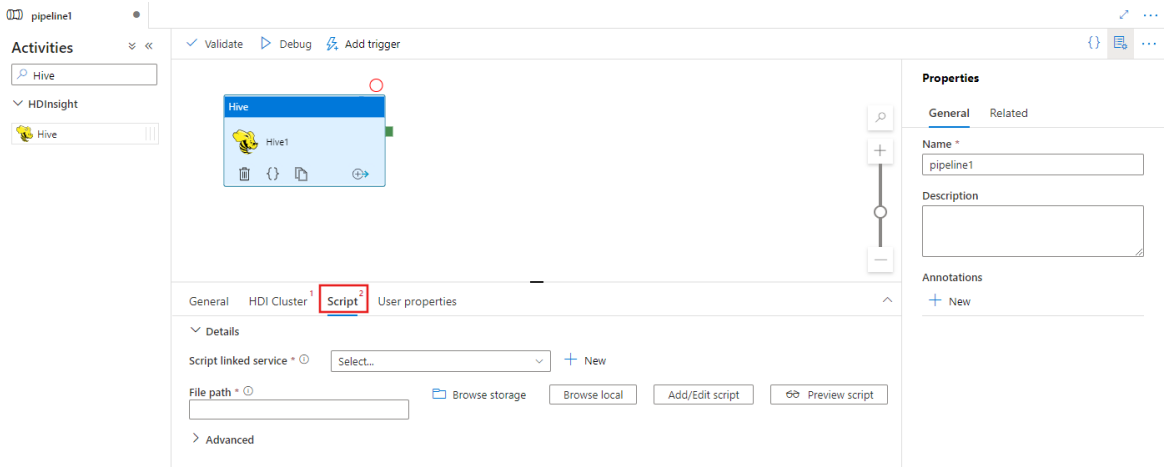
Add an HDInsight Hive activity to a pipeline with UI

To use an HDInsight Hive activity for Azure Data Lake Analytics in a pipeline, complete the following steps:

1. Search for *Hive* in the pipeline Activities pane, and drag a Hive activity to the pipeline canvas.
2. Select the new Hive activity on the canvas if it is not already selected.
3. Select the **HDI Cluster** tab to select or create a new linked service to an HDInsight cluster that will be used to execute the Hive activity.



4. Select the **Script** tab to select or create a new storage linked service, and a path within the storage location, which will host the script.



Syntax

```
{
    "name": "Hive Activity",
    "description": "description",
    "type": "HDInsightHive",
    "linkedServiceName": {
        "referenceName": "MyHDInsightLinkedService",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "scriptLinkedService": {
            "referenceName": "MyAzureStorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "scriptPath": "MyAzureStorage\\HiveScripts\\MyHiveSript.hql",
        "getDebugInfo": "Failure",
        "arguments": [
            "SampleHadoopJobArgument1"
        ],
        "defines": {
            "param1": "param1Value"
        }
    }
}
```

Syntax details

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity	Yes
description	Text describing what the activity is used for	No
type	For Hive Activity, the activity type is HDInsightHive	Yes
linkedServiceName	Reference to the HDInsight cluster registered as a linked service. To learn about this linked service, see Compute linked services article.	Yes
scriptLinkedService	Reference to an Azure Storage Linked Service used to store the Hive script to be executed. Only **Azure Blob Storage** and **ADLS Gen2** linked services are supported here. If you don't specify this Linked Service, the Azure Storage Linked Service defined in the HDInsight Linked Service is used.	No
scriptPath	Provide the path to the script file stored in the Azure Storage referred by scriptLinkedService. The file name is case-sensitive.	Yes

PROPERTY	DESCRIPTION	REQUIRED
getDebugInfo	Specifies when the log files are copied to the Azure Storage used by HDInsight cluster (or) specified by scriptLinkedService. Allowed values: None, Always, or Failure. Default value: None.	No
arguments	Specifies an array of arguments for a Hadoop job. The arguments are passed as command-line arguments to each task.	No
defines	Specify parameters as key/value pairs for referencing within the Hive script.	No
queryTimeout	Query timeout value (in minutes). Applicable when the HDInsight cluster is with Enterprise Security Package enabled.	No

> **NOTE**
> The default value for queryTimeout is 120 minutes.

Next steps

See the following articles that explain how to transform data in other ways:

- U-SQL activity
- Pig activity
- MapReduce activity
- Hadoop Streaming activity
- Spark activity
- .NET custom activity
- Stored procedure activity

Transform data using Hadoop MapReduce activity in Azure Data Factory or Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

The HDInsight MapReduce activity in an Azure Data Factory or Synapse Analytics pipeline invokes MapReduce program on your own or on-demand HDInsight cluster. This article builds on the data transformation activities article, which presents a general overview of data transformation and the supported transformation activities.

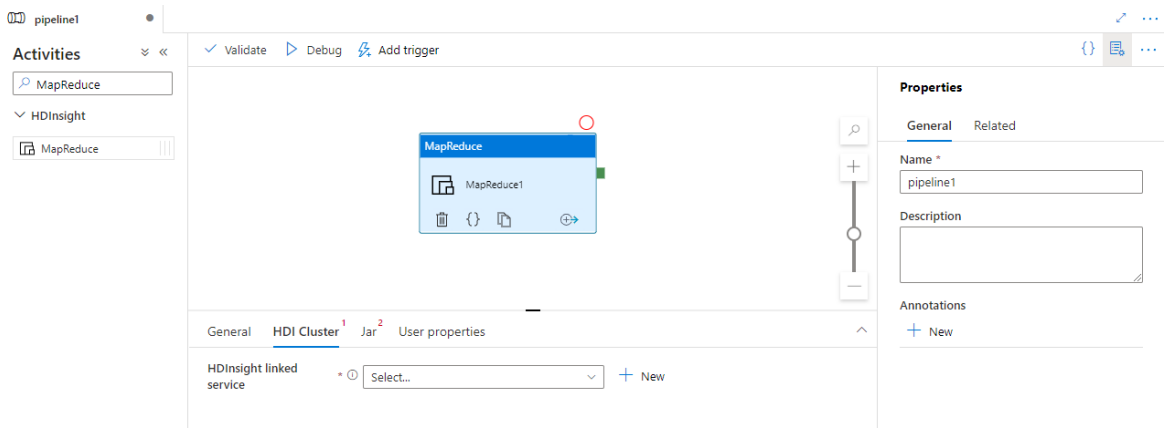
To learn more, read through the introduction articles for Azure Data Factory and Synapse Analytics, and do the tutorial: Tutorial: transform data before reading this article.

See Pig and Hive for details about running Pig/Hive scripts on a HDInsight cluster from a pipeline by using HDInsight Pig and Hive activities.

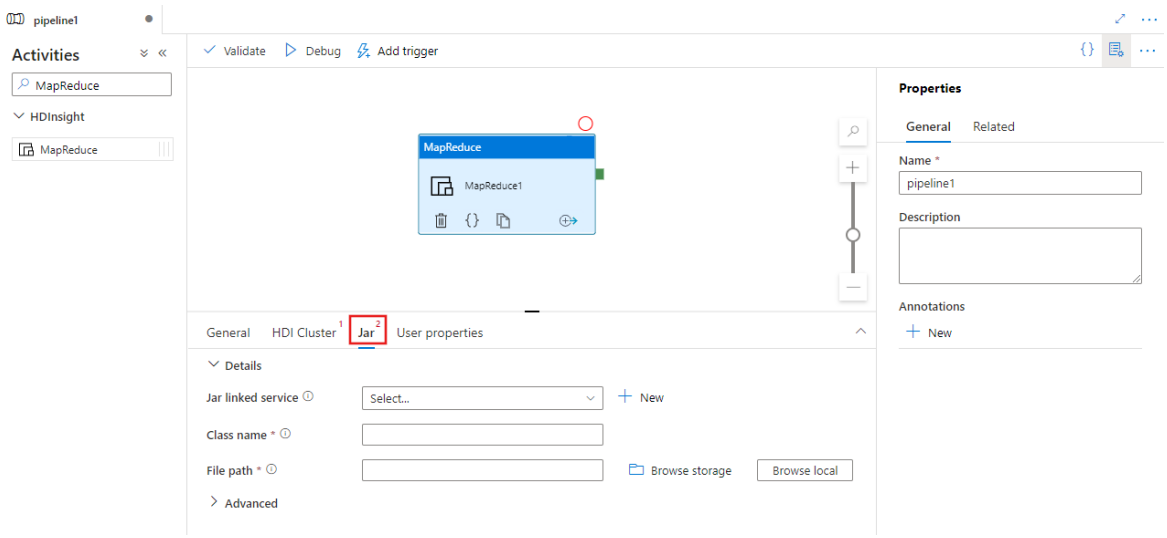
Add an HDInsight MapReduce activity to a pipeline with UI

To use an HDInsight MapReduce activity to a pipeline, complete the following steps:

1. Search for *MapReduce* in the pipeline Activities pane, and drag a MapReduce activity to the pipeline canvas.
2. Select the new MapReduce activity on the canvas if it is not already selected.
3. Select the **HDI Cluster** tab to select or create a new linked service to an HDInsight cluster that will be used to execute the MapReduce activity.



4. Select the **Jar** tab to select or create a new Jar linked service to an Azure Storage account that will host your script. Specify a class name to be executed there, and a file path within the storage location. You can also configure advanced details including a Jar libs location, debugging configuration, and arguments and parameters to be passed to the script.



Syntax

```
{
    "name": "Map Reduce Activity",
    "description": "Description",
    "type": "HDInsightMapReduce",
    "linkedServiceName": {
        "referenceName": "MyHDInsightLinkedService",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "className": "org.myorg.SampleClass",
        "jarLinkedService": {
            "referenceName": "MyAzureStorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "jarFilePath": "MyAzureStorage/jars/sample.jar",
        "getDebugInfo": "Failure",
        "arguments": [
            "-SampleHadoopJobArgument1"
        ],
        "defines": {
            "param1": "param1Value"
        }
    }
}
```

Syntax details

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity	Yes
description	Text describing what the activity is used for	No
type	For MapReduce Activity, the activity type is HDinsightMapReduce	Yes

PROPERTY	DESCRIPTION	REQUIRED
linkedServiceName	Reference to the HDInsight cluster registered as a linked service. To learn about this linked service, see Compute linked services article.	Yes
className	Name of the Class to be executed	Yes
jarLinkedService	Reference to an Azure Storage Linked Service used to store the Jar files. Only **Azure Blob Storage** and **ADLS Gen2** linked services are supported here. If you don't specify this Linked Service, the Azure Storage Linked Service defined in the HDInsight Linked Service is used.	No
jarFilePath	Provide the path to the Jar files stored in the Azure Storage referred by jarLinkedService. The file name is case-sensitive.	Yes
jarlibs	String array of the path to the Jar library files referenced by the job stored in the Azure Storage defined in jarLinkedService. The file name is case-sensitive.	No
getDebugInfo	Specifies when the log files are copied to the Azure Storage used by HDInsight cluster (or) specified by jarLinkedService. Allowed values: None, Always, or Failure. Default value: None.	No
arguments	Specifies an array of arguments for a Hadoop job. The arguments are passed as command-line arguments to each task.	No
defines	Specify parameters as key/value pairs for referencing within the Hive script.	No

Example

You can use the HDInsight MapReduce Activity to run any MapReduce jar file on an HDInsight cluster. In the following sample JSON definition of a pipeline, the HDInsight Activity is configured to run a Mahout JAR file.

```
{
    "name": "MapReduce Activity for Mahout",
    "description": "Custom MapReduce to generate Mahout result",
    "type": "HDInsightMapReduce",
    "linkedServiceName": {
        "referenceName": "MyHDInsightLinkedService",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "className": "org.apache.mahout.cf.taste.hadoop.similarity.item.ItemSimilarityJob",
        "jarLinkedService": {
            "referenceName": "MyStorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "jarFilePath": "adfsamples/Mahout/jars/mahout-examples-0.9.0.2.2.7.1-34.jar",
        "arguments": [
            "-s",
            "SIMILARITY_LOGLIKELIHOOD",
            "--input",
            "wasb://adfsamples@spestore.blob.core.windows.net/Mahout/input",
            "--output",
            "wasb://adfsamples@spestore.blob.core.windows.net/Mahout/output/",
            "--maxSimilaritiesPerItem",
            "500",
            "--tempDir",
            "wasb://adfsamples@spestore.blob.core.windows.net/Mahout/temp/mahout"
        ]
    }
}
```

You can specify any arguments for the MapReduce program in the **arguments** section. At runtime, you see a few extra arguments (for example: mapreduce.job.tags) from the MapReduce framework. To differentiate your arguments with the MapReduce arguments, consider using both option and value as arguments as shown in the following example (-s,--input,--output etc., are options immediately followed by their values).

Next steps

See the following articles that explain how to transform data in other ways:

- U-SQL activity
- Hive activity
- Pig activity
- Hadoop Streaming activity
- Spark activity
- .NET custom activity
- Stored procedure activity

Transform data using Hadoop Pig activity in Azure Data Factory or Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

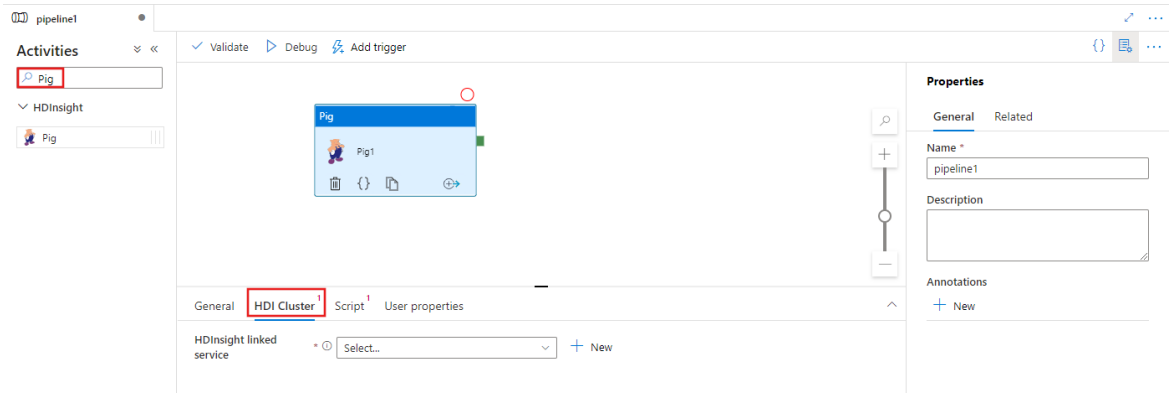
The HDInsight Pig activity in a Data Factory pipeline executes Pig queries on your own or on-demand HDInsight cluster. This article builds on the data transformation activities article, which presents a general overview of data transformation and the supported transformation activities.

To learn more, read through the introduction to Azure Data Factory or Synapse Analytics and do the Tutorial: transform data before reading this article.

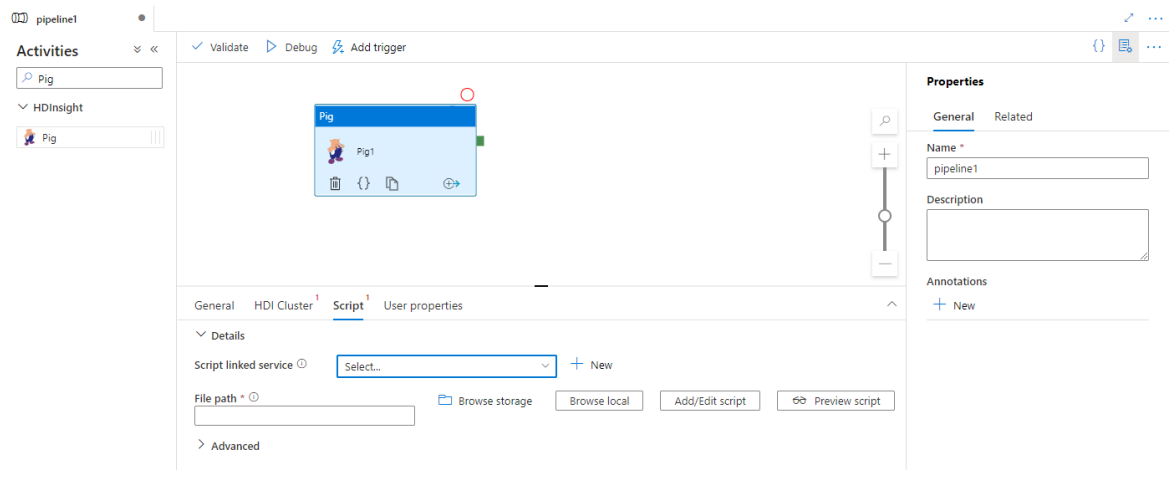
Add an HDInsight Pig activity to a pipeline with UI

To use an HDInsight Pig activity to a pipeline, complete the following steps:

1. Search for *Pig* in the pipeline Activities pane, and drag a Pig activity to the pipeline canvas.
2. Select the new Pig activity on the canvas if it is not already selected.
3. Select the **HDI Cluster** tab to select or create a new linked service to an HDInsight cluster that will be used to execute the MapReduce activity.



4. Select the **Script** tab to select or create a new script linked service to an Azure Storage location where your script will be hosted. Specify a class name to be executed there, and a file path within the storage location. You can also configure advanced details including debugging configuration, and arguments and parameters to be passed to the script.



Syntax

```
{
    "name": "Pig Activity",
    "description": "description",
    "type": "HDInsightPig",
    "linkedServiceName": {
        "referenceName": "MyHDInsightLinkedService",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "scriptLinkedService": {
            "referenceName": "MyAzureStorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "scriptPath": "MyAzureStorage\\PigScripts\\MyPigSript.pig",
        "getDebugInfo": "Failure",
        "arguments": [
            "SampleHadoopJobArgument1"
        ],
        "defines": {
            "param1": "param1Value"
        }
    }
}
```

Syntax details

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity	Yes
description	Text describing what the activity is used for	No
type	For Hive Activity, the activity type is HDInsightPig	Yes
linkedServiceName	Reference to the HDInsight cluster registered as a linked service. To learn about this linked service, see Compute linked services article.	Yes
scriptLinkedService	Reference to an Azure Storage Linked Service used to store the Pig script to be executed. Only **Azure Blob Storage** and **ADLS Gen2** linked services are supported here. If you don't specify this Linked Service, the Azure Storage Linked Service defined in the HDInsight Linked Service is used.	No
scriptPath	Provide the path to the script file stored in the Azure Storage referred by scriptLinkedService. The file name is case-sensitive.	No

PROPERTY	DESCRIPTION	REQUIRED
getDebugInfo	Specifies when the log files are copied to the Azure Storage used by HDInsight cluster (or) specified by scriptLinkedService. Allowed values: None, Always, or Failure. Default value: None.	No
arguments	Specifies an array of arguments for a Hadoop job. The arguments are passed as command-line arguments to each task.	No
defines	Specify parameters as key/value pairs for referencing within the Pig script.	No

Next steps

See the following articles that explain how to transform data in other ways:

- U-SQL activity
- Hive activity
- MapReduce activity
- Hadoop Streaming activity
- Spark activity
- .NET custom activity
- Stored procedure activity

Transform data using Spark activity in Azure Data Factory and Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

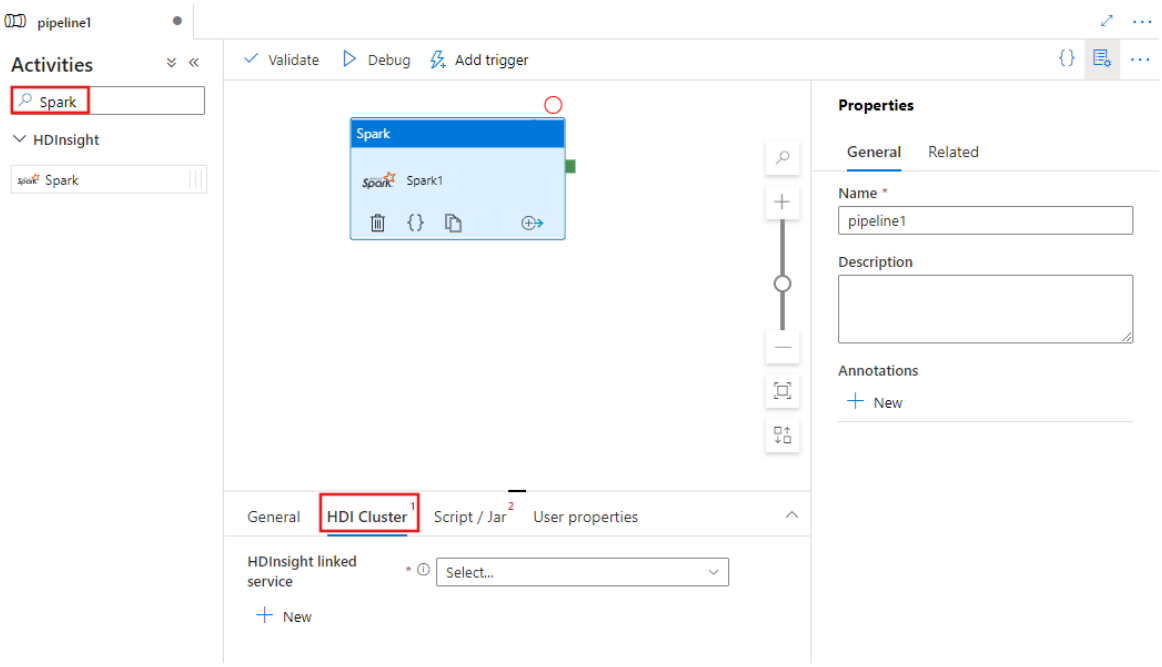
APPLIES TO: Azure Data Factory Azure Synapse Analytics

The Spark activity in a data factory and Synapse pipelines executes a Spark program on your own or on-demand HDInsight cluster. This article builds on the data transformation activities article, which presents a general overview of data transformation and the supported transformation activities. When you use an on-demand Spark linked service, the service automatically creates a Spark cluster for you just-in-time to process the data and then deletes the cluster once the processing is complete.

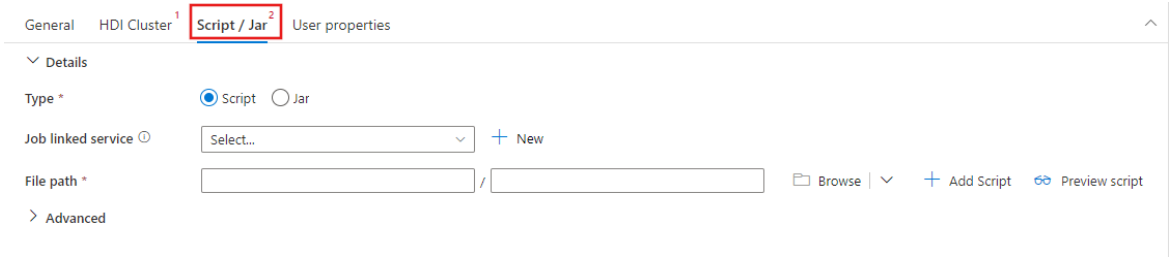
Add a Spark activity to a pipeline with UI

To use a Spark activity to a pipeline, complete the following steps:

1. Search for *Spark* in the pipeline Activities pane, and drag a Spark activity to the pipeline canvas.
2. Select the new Spark activity on the canvas if it is not already selected.
3. Select the **HDI Cluster** tab to select or create a new linked service to an HDInsight cluster that will be used to execute the Spark activity.



4. Select the **Script / Jar** tab to select or create a new job linked service to an Azure Storage account that will host your script. Specify a path to the file to be executed there. You can also configure advanced details including a proxy user, debugging configuration, and arguments and Spark configuration parameters to be passed to the script.



Spark activity properties

Here is the sample JSON definition of a Spark Activity:

```
{
    "name": "Spark Activity",
    "description": "Description",
    "type": "HDInsightSpark",
    "linkedServiceName": {
        "referenceName": "MyHDInsightLinkedService",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "sparkJobLinkedService": {
            "referenceName": "MyAzureStorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "rootPath": "adfspark",
        "entryFilePath": "test.py",
        "sparkConfig": {
            "ConfigItem1": "Value"
        },
        "getDebugInfo": "Failure",
        "arguments": [
            "SampleHadoopJobArgument1"
        ]
    }
}
```

The following table describes the JSON properties used in the JSON definition:

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity in the pipeline.	Yes
description	Text describing what the activity does.	No
type	For Spark Activity, the activity type is HDInsightSpark.	Yes
linkedServiceName	Name of the HDInsight Spark Linked Service on which the Spark program runs. To learn about this linked service, see Compute linked services article.	Yes

PROPERTY	DESCRIPTION	REQUIRED
SparkJobLinkedService	The Azure Storage linked service that holds the Spark job file, dependencies, and logs. Only **Azure Blob Storage** and **ADLS Gen2** linked services are supported here. If you do not specify a value for this property, the storage associated with HDInsight cluster is used. The value of this property can only be an Azure Storage linked service.	No
rootPath	The Azure Blob container and folder that contains the Spark file. The file name is case-sensitive. Refer to folder structure section (next section) for details about the structure of this folder.	Yes
entryFilePath	Relative path to the root folder of the Spark code/package. The entry file must be either a Python file or a .jar file.	Yes
className	Application's Java/Spark main class	No
arguments	A list of command-line arguments to the Spark program.	No
proxyUser	The user account to impersonate to execute the Spark program	No
sparkConfig	Specify values for Spark configuration properties listed in the topic: Spark Configuration - Application properties.	No
getDebugInfo	Specifies when the Spark log files are copied to the Azure storage used by HDInsight cluster (or) specified by sparkJobLinkedService. Allowed values: None, Always, or Failure. Default value: None.	No

Folder structure

Spark jobs are more extensible than Pig/Hive jobs. For Spark jobs, you can provide multiple dependencies such as jar packages (placed in the Java CLASSPATH), Python files (placed on the PYTHONPATH), and any other files.

Create the following folder structure in the Azure Blob storage referenced by the HDInsight linked service. Then, upload dependent files to the appropriate sub folders in the root folder represented by **entryFilePath**. For example, upload Python files to the pyFiles subfolder and jar files to the jars subfolder of the root folder. At runtime, the service expects the following folder structure in the Azure Blob storage:

PATH	DESCRIPTION	REQUIRED	TYPE

PATH	DESCRIPTION	REQUIRED	TYPE
`.` (root)	The root path of the Spark job in the storage linked service	Yes	Folder
<user defined >	The path pointing to the entry file of the Spark job	Yes	File
./jars	All files under this folder are uploaded and placed on the Java classpath of the cluster	No	Folder
./pyFiles	All files under this folder are uploaded and placed on the PYTHONPATH of the cluster	No	Folder
./files	All files under this folder are uploaded and placed on executor working directory	No	Folder
./archives	All files under this folder are uncompressed	No	Folder
./logs	The folder that contains logs from the Spark cluster.	No	Folder

Here is an example for a storage containing two Spark job files in the Azure Blob Storage referenced by the HDInsight linked service.

```
SparkJob1
 main.jar
 files
  input1.txt
  input2.txt
 jars
  package1.jar
  package2.jar
 logs

 archives

 pyFiles

SparkJob2
 main.py
 pyFiles
  scrip1.py
  script2.py
 logs

 archives

 jars

 files
```

Next steps

See the following articles that explain how to transform data in other ways:

- U-SQL activity
- Hive activity
- Pig activity
- MapReduce activity
- Hadoop Streaming activity
- Spark activity
- .NET custom activity
- Stored procedure activity

Transform data using Hadoop Streaming activity in Azure Data Factory or Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

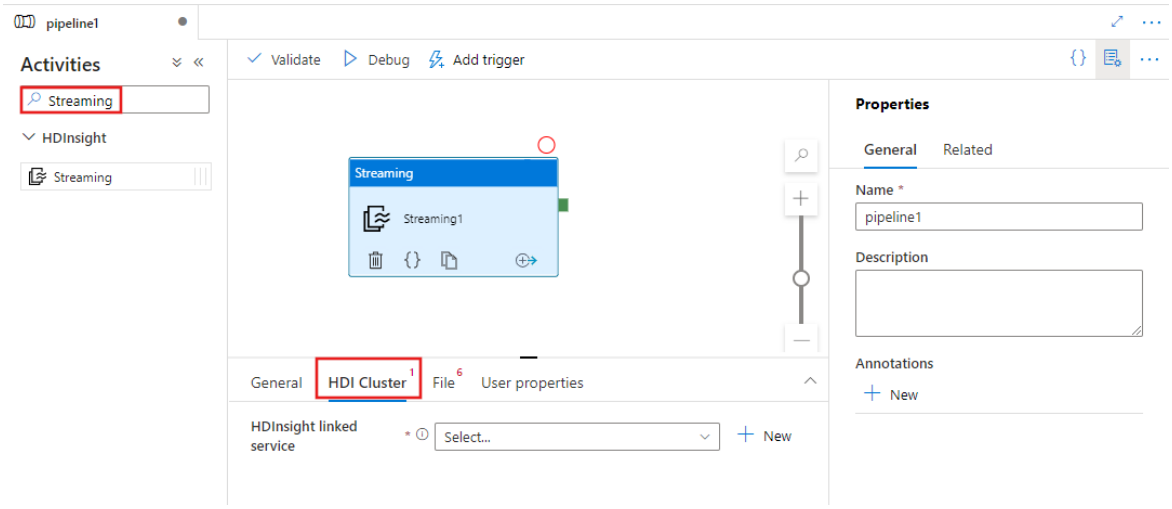
The HDInsight Streaming Activity in an Azure Data Factory or Synapse Analytics pipeline executes Hadoop Streaming programs on your own or on-demand HDInsight cluster. This article builds on the data transformation activities article, which presents a general overview of data transformation and the supported transformation activities.

To learn more, read through the introduction articles to Azure Data Factory and Synapse Analytics and do the Tutorial: transform data before reading this article.

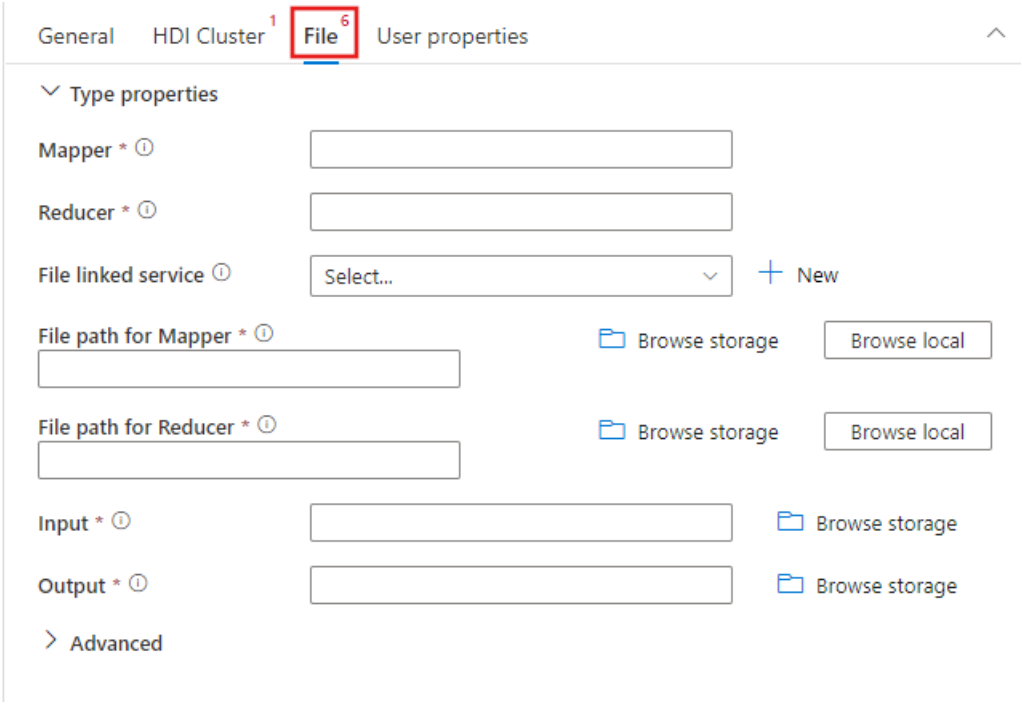
Add an HDInsight Streaming activity to a pipeline with UI

To use an HDInsight Streaming activity to a pipeline, complete the following steps:

1. Search for *Streaming* in the pipeline Activities pane, and drag a Streaming activity to the pipeline canvas.
2. Select the new Streaming activity on the canvas if it is not already selected.
3. Select the **HDI Cluster** tab to select or create a new linked service to an HDInsight cluster that will be used to execute the Streaming activity.



4. Select the **File** tab to specify the mapper and reducer names for your streaming job, and select or create a new linked service to an Azure Storage account that will the mapper, reducer, input, and output files for the job. You can also configure advanced details including debugging configuration, arguments, and parameters to be passed to the job.



JSON sample

```
{
    "name": "Streaming Activity",
    "description": "Description",
    "type": "HDInsightStreaming",
    "linkedServiceName": {
        "referenceName": "MyHDInsightLinkedService",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "mapper": "MyMapper.exe",
        "reducer": "MyReducer.exe",
        "combiner": "MyCombiner.exe",
        "fileLinkedService": {
            "referenceName": "MyAzureStorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "filePaths": [
            "<containername>/example/apps/MyMapper.exe",
            "<containername>/example/apps/MyReducer.exe",
            "<containername>/example/apps/MyCombiner.exe"
        ],
        "input": "wasb://<containername>@<accountname>.blob.core.windows.net/example/input/MapperInput.txt",
        "output":
"wasb://<containername>@<accountname>.blob.core.windows.net/example/output/ReducerOutput.txt",
        "commandEnvironment": [
            "CmdEnvVarName=CmdEnvVarValue"
        ],
        "getDebugInfo": "Failure",
        "arguments": [
            "SampleHadoopJobArgument1"
        ],
        "defines": {
            "param1": "param1Value"
        }
    }
}
```

Syntax details

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity	Yes
description	Text describing what the activity is used for	No
type	For Hadoop Streaming Activity, the activity type is HDInsightStreaming	Yes
linkedServiceName	Reference to the HDInsight cluster registered as a linked service. To learn about this linked service, see Compute linked services article.	Yes
mapper	Specifies the name of the mapper executable	Yes
reducer	Specifies the name of the reducer executable	Yes
combiner	Specifies the name of the combiner executable	No
fileLinkedService	Reference to an Azure Storage Linked Service used to store the Mapper, Combiner, and Reducer programs to be executed. Only **Azure Blob Storage** and **ADLS Gen2** linked services are supported here. If you don't specify this Linked Service, the Azure Storage Linked Service defined in the HDInsight Linked Service is used.	No
filePath	Provide an array of path to the Mapper, Combiner, and Reducer programs stored in the Azure Storage referred by fileLinkedService. The path is case-sensitive.	Yes
input	Specifies the WASB path to the input file for the Mapper.	Yes
output	Specifies the WASB path to the output file for the Reducer.	Yes
getDebugInfo	Specifies when the log files are copied to the Azure Storage used by HDInsight cluster (or) specified by scriptLinkedService. Allowed values: None, Always, or Failure. Default value: None.	No
arguments	Specifies an array of arguments for a Hadoop job. The arguments are passed as command-line arguments to each task.	No

PROPERTY	DESCRIPTION	REQUIRED
defines	Specify parameters as key/value pairs for referencing within the Hive script.	No

Next steps

See the following articles that explain how to transform data in other ways:

- U-SQL activity
- Hive activity
- Pig activity
- MapReduce activity
- Spark activity
- .NET custom activity
- Stored procedure activity

Execute Azure Machine Learning pipelines in Azure Data Factory and Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Run your Azure Machine Learning pipelines as a step in your Azure Data Factory and Synapse Analytics pipelines. The Machine Learning Execute Pipeline activity enables batch prediction scenarios such as identifying possible loan defaults, determining sentiment, and analyzing customer behavior patterns.

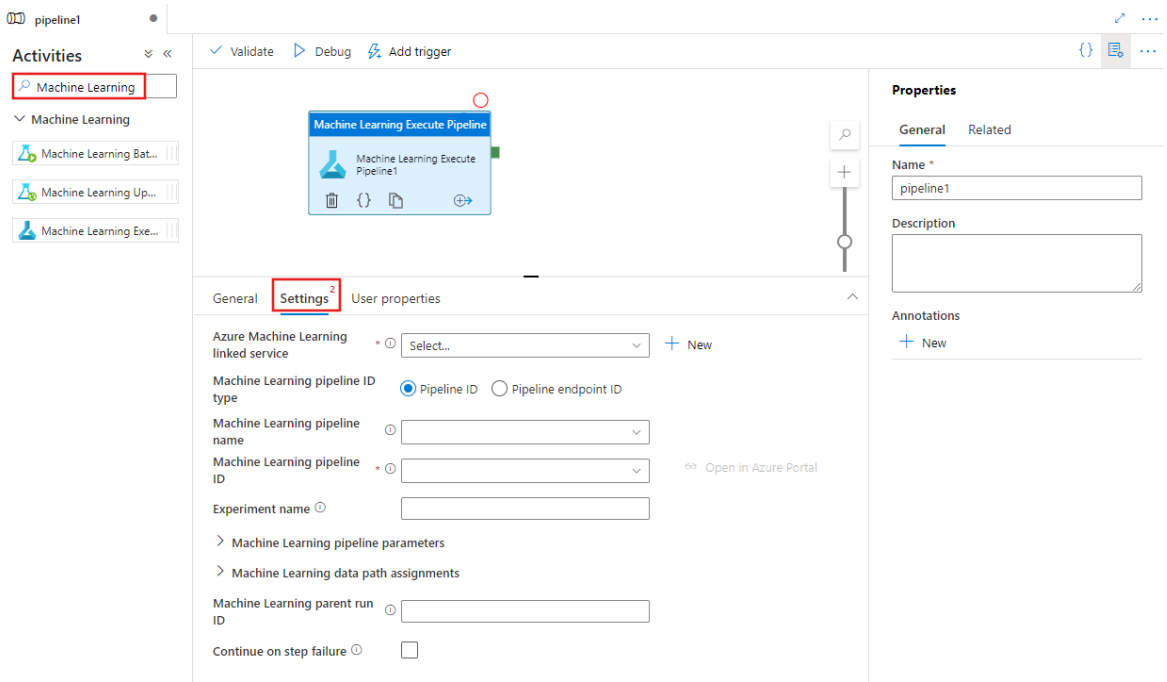
The below video features a six-minute introduction and demonstration of this feature.

Create a Machine Learning Execute Pipeline activity with UI

To use a Machine Learning Execute Pipeline activity in a pipeline, complete the following steps:

1. Search for *Machine Learning* in the pipeline Activities pane, and drag a Machine Learning Execute Pipeline activity to the pipeline canvas.

2. Select the new Machine Learning Execute Pipeline activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



3. Select an existing or create a new Azure Machine Learning linked service, and provide details of the pipeline and experiment, and any pipeline parameters or data path assignments required for the pipeline.

Syntax

```
{
    "name": "Machine Learning Execute Pipeline",
    "type": "AzureMLExecutePipeline",
    "linkedServiceName": {
        "referenceName": "AzureMLService",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "mlPipelineId": "machine learning pipeline ID",
        "experimentName": "experimentName",
        "mlPipelineParameters": {
            "mlParameterName": "mlParameterValue"
        }
    }
}
```

Type properties

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
name	Name of the activity in the pipeline	String	Yes
type	Type of activity is 'AzureMLExecutePipeline'	String	Yes
linkedServiceName	Linked Service to Azure Machine Learning	Linked service reference	Yes
mlPipelineId	ID of the published Azure Machine Learning pipeline	String (or expression with resultType of string)	Yes
experimentName	Run history experiment name of the Machine Learning pipeline run	String (or expression with resultType of string)	No
mlPipelineParameters	Key, Value pairs to be passed to the published Azure Machine Learning pipeline endpoint. Keys must match the names of pipeline parameters defined in the published Machine Learning pipeline	Object with key value pairs (or Expression with resultType object)	No
mlParentRunId	The parent Azure Machine Learning pipeline run ID	String (or expression with resultType of string)	No
dataPathAssignments	Dictionary used for changing datapaths in Azure Machine learning. Enables the switching of datapaths	Object with key value pairs	No

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
continueOnStepFailure	Whether to continue execution of other steps in the Machine Learning pipeline run if a step fails	boolean	No

> **NOTE**
> To populate the dropdown items in Machine Learning pipeline name and ID, the user needs to have permission to list ML pipelines. The UI calls AzureMLService APIs directly using the logged in user's credentials.

Next steps

See the following articles that explain how to transform data in other ways:

- Execute Data Flow activity
- U-SQL activity
- Hive activity
- Pig activity
- MapReduce activity
- Hadoop Streaming activity
- Spark activity
- .NET custom activity
- Stored procedure activity

Create a predictive pipeline using Machine Learning Studio (classic) with Azure Data Factory or Synapse Analytics

8/19/2022 • 8 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

IMPORTANT

Support for Machine Learning Studio (classic) will end on 31 August 2024. We recommend you transition to Azure Machine Learning by that date.

Beginning 1 December 2021, you will not be able to create new Machine Learning Studio (classic) resources (workspace and web service plan). Through 31 August 2024, you can continue to use the existing Machine Learning Studio (classic) experiments and web services.

- See information on moving machine learning projects from ML Studio (classic) to Azure Machine Learning.
- Learn more about Azure Machine Learning

ML Studio (classic) documentation is being retired and may not be updated in the future.

NOTE

Since Machine Learning Studio (classic) resources can no longer be created after 1 Dec, 2021, users are encouraged to use Azure Machine Learning with the Machine Learning Execute Pipeline activity rather than using the Batch Execution activity to execute Machine Learning Studio (classic) batches.

ML Studio (classic) enables you to build, test, and deploy predictive analytics solutions. From a high-level point of view, it is done in three steps:

1. **Create a training experiment**. You do this step by using the ML Studio (classic). ML Studio (classic) is a collaborative visual development environment that you use to train and test a predictive analytics model using training data.
2. **Convert it to a predictive experiment**. Once your model has been trained with existing data and you are ready to use it to score new data, you prepare and streamline your experiment for scoring.
3. **Deploy it as a web service**. You can publish your scoring experiment as an Azure web service. You can send data to your model via this web service end point and receive result predictions from the model.

Using Machine Learning Studio (classic) with Azure Data Factory or Synapse Analytics

Azure Data Factory and Synapse Analytics enable you to easily create pipelines that use a published Machine Learning Studio (classic) web service for predictive analytics. Using the **Batch Execution Activity** in a pipeline, you can invoke Machine Learning Studio (classic) web service to make predictions on the data in batch.

Over time, the predictive models in the Machine Learning Studio (classic) scoring experiments need to be retrained using new input datasets. You can retrain a model from a pipeline by doing the following steps:

1. Publish the training experiment (not predictive experiment) as a web service. You do this step in the ML Studio (classic) as you did to expose predictive experiment as a web service in the previous scenario.
2. Use the ML Studio (classic) Batch Execution Activity to invoke the web service for the training experiment.

Basically, you can use the ML Studio (classic) Batch Execution activity to invoke both training web service and scoring web service.

After you are done with retraining, update the scoring web service (predictive experiment exposed as a web service) with the newly trained model by using the **ML Studio (classic) Update Resource Activity**. See Updating models using Update Resource Activity article for details.

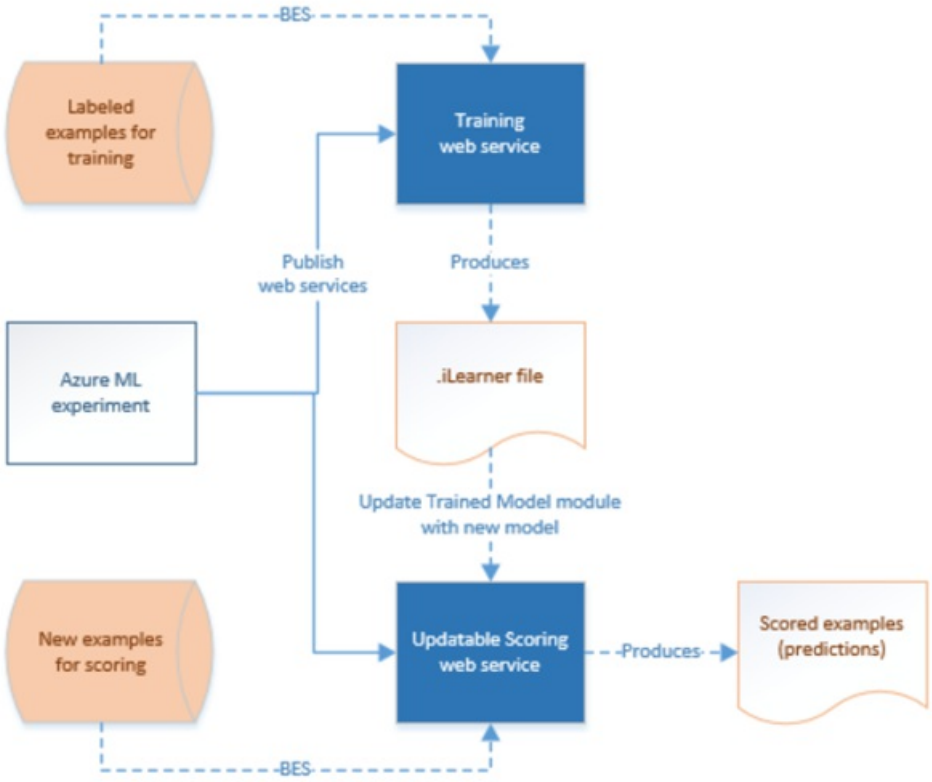
ML Studio (classic) linked service

You create an **Machine Learning Studio (classic)** linked service to link a Machine Learning Studio (classic) Web Service. The Linked Service is used by Machine Learning Studio (classic) Batch Execution Activity and Update Resource Activity.

```
{
    "type" : "linkedServices",
    "name": "AzureMLLinkedService",
    "properties": {
        "type": "AzureML",
        "typeProperties": {
            "mlEndpoint": "URL to Azure ML Predictive Web Service",
            "apiKey": {
                "type": "SecureString",
                "value": "api key"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

See Compute linked services article for descriptions about properties in the JSON definition.

Machine Learning Studio (classic) supports both Classic Web Services and New Web Services for your predictive experiment. You can choose the right one to use from your Data Factory or Synapse workspace. To get the information required to create the Machine Learning Studio (classic) Linked Service, go to https://services.azureml.net, where all your (new) Web Services and Classic Web Services are listed. Click the Web Service you would like to access, and click **Consume** page. Copy **Primary Key** for **apiKey** property, and **Batch Requests** for **mlEndpoint** property.



ML Studio (classic) Batch Execution activity

The following JSON snippet defines an ML Studio (classic) Batch Execution activity. The activity definition has a reference to the ML Studio (classic) linked service you created earlier.

```
{
    "name": "AzureMLExecutionActivityTemplate",
    "description": "description",
    "type": "AzureMLBatchExecution",
    "linkedServiceName": {
        "referenceName": "AzureMLLinkedService",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "webServiceInputs": {
            "<web service input name 1>": {
                "LinkedServiceName":{
                    "referenceName": "AzureStorageLinkedService1",
                    "type": "LinkedServiceReference"
                },
                "FilePath":"path1"
            },
            "<web service input name 2>": {
                "LinkedServiceName":{
                    "referenceName": "AzureStorageLinkedService1",
                    "type": "LinkedServiceReference"
                },
                "FilePath":"path2"
            }
        },
        "webServiceOutputs": {
            "<web service output name 1>": {
                "LinkedServiceName":{
                    "referenceName": "AzureStorageLinkedService2",
                    "type": "LinkedServiceReference"
                },
                "FilePath":"path3"
            },
            "<web service output name 2>": {
                "LinkedServiceName":{
                    "referenceName": "AzureStorageLinkedService2",
                    "type": "LinkedServiceReference"
                },
                "FilePath":"path4"
            }
        },
        "globalParameters": {
            "<Parameter 1 Name>": "<parameter value>",
            "<parameter 2 name>": "<parameter 2 value>"
        }
    }
}
```

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity in the pipeline	Yes
description	Text describing what the activity does.	No
type	For Data Lake Analytics U-SQL activity, the activity type is **AzureMLBatchExecution**.	Yes
linkedServiceName	Linked Services to the ML Studio (classic) Linked Service. To learn about this linked service, see Compute linked services article.	Yes

PROPERTY	DESCRIPTION	REQUIRED
webServiceInputs	Key, Value pairs, mapping the names of ML Studio (classic) Web Service Inputs. Key must match the input parameters defined in the published ML Studio (classic) Web Service. Value is an Azure Storage Linked Services and FilePath properties pair specifying the input Blob locations.	No
webServiceOutputs	Key, Value pairs, mapping the names of ML Studio (classic) Web Service Outputs. Key must match the output parameters defined in the published ML Studio (classic) Web Service. Value is an Azure Storage Linked Services and FilePath properties pair specifying the output Blob locations.	No
globalParameters	Key, Value pairs to be passed to the ML Studio (classic) Batch Execution Service endpoint. Keys must match the names of web service parameters defined in the published ML Studio (classic) web service. Values are passed in the GlobalParameters property of the ML Studio (classic) batch execution request	No

Scenario 1: Experiments using Web service inputs/outputs that refer to data in Azure Blob Storage

In this scenario, the Machine Learning Studio (classic) Web service makes predictions using data from a file in an Azure blob storage and stores the prediction results in the blob storage. The following JSON defines a pipeline with an AzureMLBatchExecution activity. The input and output data in Azure Blog Storage is referenced using a LinkedName and FilePath pair. In the sample Linked Service of inputs and outputs are different, you can use different Linked Services for each of your inputs/outputs for the service to be able to pick up the right files and send to Machine Learning Studio (classic) Web Service.

> **IMPORTANT**
> In your ML Studio (classic) experiment, web service input and output ports, and global parameters have default names ("input1", "input2") that you can customize. The names you use for webServiceInputs, webServiceOutputs, and globalParameters settings must exactly match the names in the experiments. You can view the sample request payload on the Batch Execution Help page for your ML Studio (classic) endpoint to verify the expected mapping.

```
{
    "name": "AzureMLExecutionActivityTemplate",
    "description": "description",
    "type": "AzureMLBatchExecution",
    "linkedServiceName": {
        "referenceName": "AzureMLLinkedService",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "webServiceInputs": {
            "input1": {
                "LinkedServiceName":{
                    "referenceName": "AzureStorageLinkedService1",
                    "type": "LinkedServiceReference"
                },
                "FilePath":"amltest/input/in1.csv"
            },
            "input2": {
                "LinkedServiceName":{
                    "referenceName": "AzureStorageLinkedService1",
                    "type": "LinkedServiceReference"
                },
                "FilePath":"amltest/input/in2.csv"
            }
        },
        "webServiceOutputs": {
            "outputName1": {
                "LinkedServiceName":{
                    "referenceName": "AzureStorageLinkedService2",
                    "type": "LinkedServiceReference"
                },
                "FilePath":"amltest2/output/out1.csv"
            },
            "outputName2": {
                "LinkedServiceName":{
                    "referenceName": "AzureStorageLinkedService2",
                    "type": "LinkedServiceReference"
                },
                "FilePath":"amltest2/output/out2.csv"
            }
        }
    }
}
```

Scenario 2: Experiments using Reader/Writer Modules to refer to data in various storages

Another common scenario when creating ML Studio (classic) experiments is to use Import Data and Output Data modules. The Import Data module is used to load data into an experiment and the Output Data module is to save data from your experiments. For details about Import Data and Output Data modules, see Import Data and Output Data topics on MSDN Library.

When using the Import Data and Output Data modules, it is good practice to use a Web service parameter for each property of these modules. These web parameters enable you to configure the values during runtime. For example, you could create an experiment with an Import Data module that uses an Azure SQL Database: XXX.database.windows.net. After the web service has been deployed, you want to enable the consumers of the web service to specify another logical SQL server called `YYY.database.windows.net`. You can use a Web service parameter to allow this value to be configured.

> **NOTE**
> Web service input and output are different from Web service parameters. In the first scenario, you have seen how an input and output can be specified for an ML Studio (classic) Web service. In this scenario, you pass parameters for a Web service that correspond to properties of Import Data/Output Data modules.

Let's look at a scenario for using Web service parameters. You have a deployed ML Studio (classic) web service that uses a reader module to read data from one of the data sources supported by ML Studio (classic) (for example: Azure SQL Database). After the batch execution is performed, the results are written using a Writer module (Azure SQL Database). No web service inputs and outputs are defined in the experiments. In this case, we recommend that you configure relevant web service parameters for the reader and writer modules. This configuration allows the reader/writer modules to be configured when using the AzureMLBatchExecution activity. You specify Web service parameters in the **globalParameters** section in the activity JSON as follows.

```
"typeProperties": {
    "globalParameters": {
        "Database server name": "<myserver>.database.windows.net",
        "Database name": "<database>",
        "Server user account name": "<user name>",
        "Server user account password": "<password>"
    }
}
```

> **NOTE**
> The Web service parameters are case-sensitive, so ensure that the names you specify in the activity JSON match the ones exposed by the Web service.

After you are done with retraining, update the scoring web service (predictive experiment exposed as a web service) with the newly trained model by using the **ML Studio (classic) Update Resource Activity**. See Updating models using Update Resource Activity article for details.

Next steps

See the following articles that explain how to transform data in other ways:

- U-SQL activity
- Hive activity
- Pig activity
- MapReduce activity
- Hadoop Streaming activity
- Spark activity
- .NET custom activity
- Stored procedure activity

Update Machine Learning Studio (classic) models by using Update Resource activity

8/19/2022 • 7 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

IMPORTANT

Support for Machine Learning Studio (classic) will end on 31 August 2024. We recommend you transition to Azure Machine Learning by that date.

Beginning 1 December 2021, you will not be able to create new Machine Learning Studio (classic) resources (workspace and web service plan). Through 31 August 2024, you can continue to use the existing Machine Learning Studio (classic) experiments and web services.

- See information on moving machine learning projects from ML Studio (classic) to Azure Machine Learning.
- Learn more about Azure Machine Learning

ML Studio (classic) documentation is being retired and may not be updated in the future.

NOTE

Since Machine Learning Studio (classic) resources can no longer be created after 1 Dec, 2021, users are encouraged to use Azure Machine Learning with the Machine Learning Execute Pipeline activity rather than using the Update Resource activity to update Machine Learning Studio (classic) models.

This article complements the main Machine Learning Studio (classic) integration article: Create predictive pipelines using Machine Learning Studio (classic). If you haven't already done so, review the main article before reading through this article.

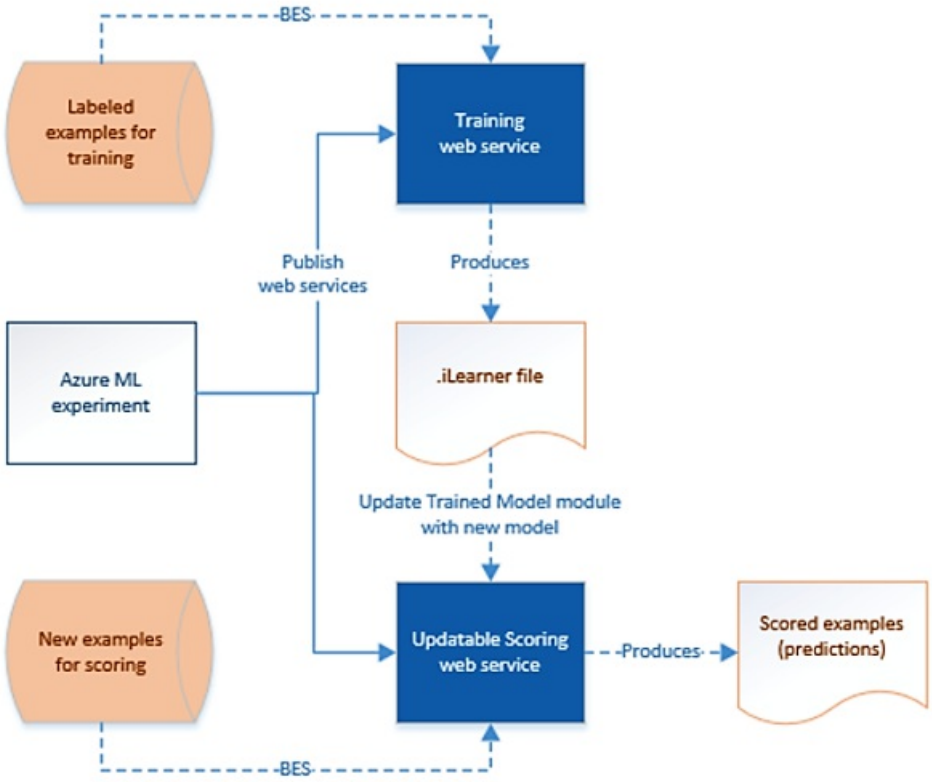
Overview

As part of the process of operationalizing Machine Learning Studio (classic) models, your model is trained and saved. You then use it to create a predictive Web service. The Web service can then be consumed in web sites, dashboards, and mobile apps.

Models you create using Machine Learning Studio (classic) are typically not static. As new data becomes available or when the consumer of the API has their own data the model needs to be retrained.

Retraining may occur frequently. With Batch Execution activity and Update Resource activity, you can operationalize the Machine Learning Studio (classic) model retraining and updating the predictive Web Service.

The following picture depicts the relationship between training and predictive Web Services.



Machine Learning Studio (classic) update resource activity

The following JSON snippet defines an Machine Learning Studio (classic) Batch Execution activity.

```
{
    "name": "amlUpdateResource",
    "type": "AzureMLUpdateResource",
    "description": "description",
    "linkedServiceName": {
        "type": "LinkedServiceReference",
        "referenceName": "updatableScoringEndpoint2"
    },
    "typeProperties": {
        "trainedModelName": "ModelName",
        "trainedModelLinkedServiceName": {
                    "type": "LinkedServiceReference",
                    "referenceName": "StorageLinkedService"
                },
        "trainedModelFilePath": "ilearner file path"
    }
}
```

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity in the pipeline	Yes
description	Text describing what the activity does.	No
type	For Machine Learning Studio (classic) Update Resource activity, the activity type is **AzureMLUpdateResource**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
linkedServiceName	Machine Learning Studio (classic) linked service that contains updateResourceEndpoint property.	Yes
trainedModelName	Name of the Trained Model module in the Web Service experiment to be updated	Yes
trainedModelLinkedServiceName	Name of Azure Storage linked service holding the ilearner file that is uploaded by the update operation	Yes
trainedModelFilePath	The relative file path in trainedModelLinkedService to represent the ilearner file that is uploaded by the update operation	Yes

End-to-end workflow

The entire process of operationalizing retraining a model and update the predictive Web Services involves the following steps:

- Invoke the **training Web Service** by using the **Batch Execution activity**. Invoking a training Web Service is the same as invoking a predictive Web Service described in Create predictive pipelines using Machine Learning Studio (classic) and the Batch Execution activity. The output of the training Web Service is an iLearner file that you can use to update the predictive Web Service.
- Invoke the **update resource endpoint** of the **predictive Web Service** by using the **Update Resource activity** to update the Web Service with the newly trained model.

Machine Learning Studio (classic) linked service

For the above mentioned end-to-end workflow to work, you need to create two Machine Learning Studio (classic) linked services:

1. An Machine Learning Studio (classic) linked service to the training web service, this linked service is used by Batch Execution activity in the same way as what's mentioned in Create predictive pipelines using Machine Learning Studio (classic) and the Batch Execution activity. Difference is the output of the training web service is an iLearner file, which is then used by Update Resource activity to update the predictive web service.
2. An Machine Learning Studio (classic) linked service to the update resource endpoint of the predictive web service. This linked service is used by Update Resource activity to update the predictive web service using the iLearner file returned from above step.

For the second Machine Learning Studio (classic) linked service, the configuration is different when your Machine Learning Studio (classic) Web Service is a classic Web Service or a new Web Service. The differences are discussed separately in the following sections.

Web service is new Azure Resource Manager web service

If the web service is the new type of web service that exposes an Azure Resource Manager endpoint, you do not need to add the second **non-default** endpoint. The **updateResourceEndpoint** in the linked service is of the format:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resource-group-name}/providers/Microsoft.MachineLearning/webServices/{web-service-name}?api-version=2016-05-01-preview
```

You can get values for place holders in the URL when querying the web service on the Machine Learning Studio (classic) Web Services Portal.

The new type of update resource endpoint requires service principal authentication. To use service principal authentication, register an application entity in Azure Active Directory (Azure AD) and grant it the **Contributor** or **Owner** role of the subscription or the resource group where the web service belongs to. The See how to create service principal and assign permissions to manage Azure resource. Make note of the following values, which you use to define the linked service:

- Application ID
- Application key
- Tenant ID

Here is a sample linked service definition:

```
{
    "name": "AzureMLLinkedService",
    "properties": {
        "type": "AzureML",
        "description": "The linked service for AML web service.",
        "typeProperties": {
            "mlEndpoint": "https://ussouthcentral.services.azureml.net/workspaces/0000000000000000000000000000000000000000000000000000/services/00000000000000000000000000000000/jobs?api-version=2.0",
            "apiKey": {
                "type": "SecureString",
                "value": "APIKeyOfEndpoint1"
            },
            "updateResourceEndpoint": "https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resource-group-name}/providers/Microsoft.MachineLearning/webServices/{web-service-name}?api-version=2016-05-01-preview",
            "servicePrincipalId": "000000000-0000-0000-0000-0000000000000",
            "servicePrincipalKey": {
                "type": "SecureString",
                "value": "servicePrincipalKey"
            },
            "tenant": "mycompany.com"
        }
    }
}
```

The following scenario provides more details. It has an example for retraining and updating Machine Learning Studio (classic) models from a pipeline.

Sample: Retraining and updating an Machine Learning Studio (classic) model

This section provides a sample pipeline that uses the **Azure Machine Learning Studio (classic) Batch Execution activity** to retrain a model. The pipeline also uses the **Azure Machine Learning Studio (classic) Update Resource activity** to update the model in the scoring web service. The section also provides JSON snippets for all the linked services, datasets, and pipeline in the example.

Azure Blob storage linked service:

The Azure Storage holds the following data:

- training data. The input data for the Machine Learning Studio (classic) training web service.

- iLearner file. The output from the Machine Learning Studio (classic) training web service. This file is also the input to the Update Resource activity.

Here is the sample JSON definition of the linked service:

```
{
    "name": "StorageLinkedService",
    "properties": {
        "type": "AzureStorage",
        "typeProperties": {
            "connectionString": "DefaultEndpointsProtocol=https;AccountName=name;AccountKey=key"
        }
    }
}
```

Linked service for Machine Learning Studio (classic) training endpoint

The following JSON snippet defines an Machine Learning Studio (classic) linked service that points to the default endpoint of the training web service.

```
{
    "name": "trainingEndpoint",
    "properties": {
        "type": "AzureML",
        "typeProperties": {
            "mlEndpoint": "https://ussouthcentral.services.azureml.net/workspaces/xxx/services/--training
experiment--/jobs",
            "apiKey": "myKey"
        }
    }
}
```

In **Azure Machine Learning Studio (classic)**, do the following to get values for **mlEndpoint** and **apiKey**:

1. Click **WEB SERVICES** on the left menu.
2. Click the **training web service** in the list of web services.
3. Click copy next to **API key** text box. Paste the key in the clipboard into the Data Factory JSON editor.
4. In the **Azure Machine Learning Studio (classic)**, click **BATCH EXECUTION** link.
5. Copy the **Request URI** from the **Request** section and paste it into the JSON editor.

Linked service for Azure Machine Learning Studio (classic) updatable scoring endpoint:

The following JSON snippet defines an Azure Machine Learning Studio (classic) linked service that points to updatable endpoint of the scoring web service.

```
{
    "name": "updatableScoringEndpoint2",
    "properties": {
        "type": "AzureML",
        "typeProperties": {
            "mlEndpoint":
"https://ussouthcentral.services.azureml.net/workspaces/00000000eb0abe4d6bbb1d7886062747d7/services/00000000
026734a5889e02fbb1f65cefd/jobs?api-version=2.0",
            "apiKey":
"soooooooooh3WvG1hBfKS2BNNcfwSO7hhY6dY98noLfOdqQydYDIXyf2KoIaN3JpALu/AKtflHWMOCuicm/Q==",
            "updateResourceEndpoint": "https://management.azure.com/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/Default-MachineLearning-
SouthCentralUS/providers/Microsoft.MachineLearning/webServices/myWebService?api-version=2016-05-01-preview",
            "servicePrincipalId": "fe200044-c008-4008-a005-94000000731",
            "servicePrincipalKey": "zWa0000000000Tp6FjtZOspK/WMA2tQ08c8U+gZRBlw=",
            "tenant": "mycompany.com"
        }
    }
}
```

Pipeline

The pipeline has two activities: **AzureMLBatchExecution** and **AzureMLUpdateResource**. The Batch Execution activity takes the training data as input and produces an iLearner file as an output. The Update Resource activity then takes this iLearner file and use it to update the predictive web service.

```
{
    "name": "LookupPipelineDemo",
    "properties": {
        "activities": [
            {
                "name": "amlBEGetilearner",
                "description": "Use AML BES to get the ileaner file from training web service",
                "type": "AzureMLBatchExecution",
                "linkedServiceName": {
                    "referenceName": "trainingEndpoint",
                    "type": "LinkedServiceReference"
                },
                "typeProperties": {
                    "webServiceInputs": {
                        "input1": {
                            "LinkedServiceName":{
                                "referenceName": "StorageLinkedService",
                                "type": "LinkedServiceReference"
                            },
                            "FilePath":"azuremltesting/input"
                        },
                        "input2": {
                            "LinkedServiceName":{
                                "referenceName": "StorageLinkedService",
                                "type": "LinkedServiceReference"
                            },
                            "FilePath":"azuremltesting/input"
                        }
                    },
                    "webServiceOutputs": {
                        "output1": {
                            "LinkedServiceName":{
                                "referenceName": "StorageLinkedService",
                                "type": "LinkedServiceReference"
                            },
                            "FilePath":"azuremltesting/output"
                        }
                    }
                }
            },
            {
                "name": "amlUpdateResource",
                "type": "AzureMLUpdateResource",
                "description": "Use AML Update Resource to update the predict web service",
                "linkedServiceName": {
                    "type": "LinkedServiceReference",
                    "referenceName": "updatableScoringEndpoint2"
                },
                "typeProperties": {
                    "trainedModelName": "ADFV2Sample Model [trained model]",
                    "trainedModelLinkedServiceName": {
                        "type": "LinkedServiceReference",
                        "referenceName": "StorageLinkedService"
                    },
                    "trainedModelFilePath": "azuremltesting/output/newModelForArm.ilearner"
                },
                "dependsOn": [
                    {
                        "activity": "amlbeGetilearner",
                        "dependencyConditions": [ "Succeeded" ]
                    }
                ]
            }
        ]
    }
}
```

Next steps

See the following articles that explain how to transform data in other ways:

- U-SQL activity
- Hive activity
- Pig activity
- MapReduce activity
- Hadoop Streaming activity
- Spark activity
- .NET custom activity
- Stored procedure activity

Transform data by using the SQL Server Stored Procedure activity in Azure Data Factory or Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

You use data transformation activities in a Data Factory or Synapse pipeline to transform and process raw data into predictions and insights. The Stored Procedure Activity is one of the transformation activities that pipelines support. This article builds on the transform data article, which presents a general overview of data transformation and the supported transformation activities.

> **NOTE**
> If you are new to Azure Data Factory, read through Introduction to Azure Data Factory and do the tutorial: Tutorial: transform data before reading this article. To learn more about Synapse Analytics, read What is Azure Synapse Analytics.

You can use the Stored Procedure Activity to invoke a stored procedure in one of the following data stores in your enterprise or on an Azure virtual machine (VM):

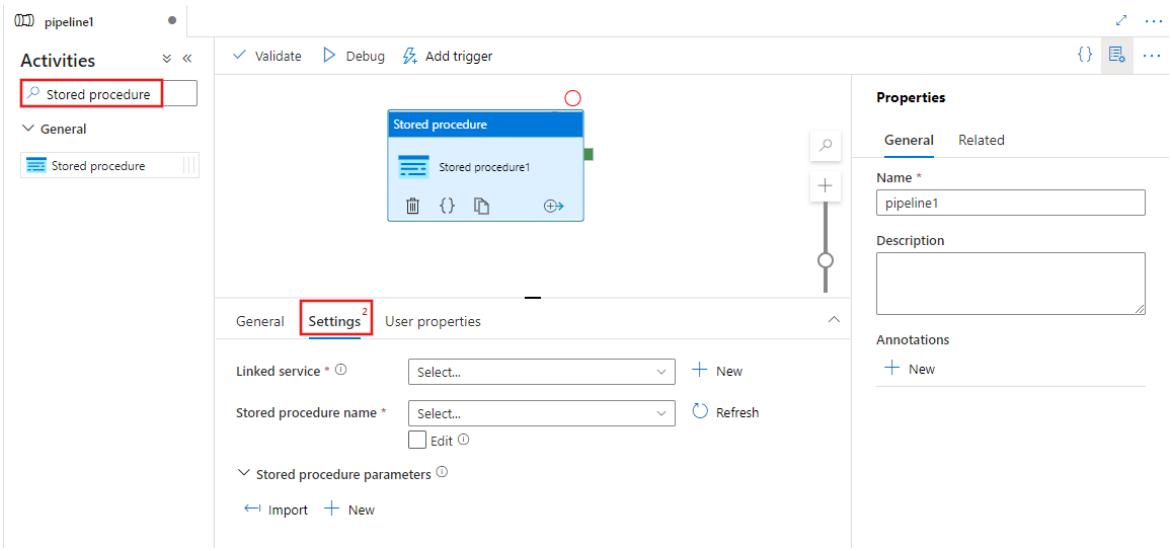
- Azure SQL Database
- Azure Synapse Analytics
- SQL Server Database. If you are using SQL Server, install Self-hosted integration runtime on the same machine that hosts the database or on a separate machine that has access to the database. Self-Hosted integration runtime is a component that connects data sources on-premises/on Azure VM with cloud services in a secure and managed way. See Self-hosted integration runtime article for details.

> **IMPORTANT**
> When copying data into Azure SQL Database or SQL Server, you can configure the **SqlSink** in copy activity to invoke a stored procedure by using the **sqlWriterStoredProcedureName** property. For details about the property, see following connector articles: Azure SQL Database, SQL Server. Invoking a stored procedure while copying data into an Azure Synapse Analytics by using a copy activity is not supported. But, you can use the stored procedure activity to invoke a stored procedure in Azure Synapse Analytics.
>
> When copying data from Azure SQL Database or SQL Server or Azure Synapse Analytics, you can configure **SqlSource** in copy activity to invoke a stored procedure to read data from the source database by using the **sqlReaderStoredProcedureName** property. For more information, see the following connector articles: Azure SQL Database, SQL Server, Azure Synapse Analytics

Create a Stored Procedure activity with UI

To use a Stored Procedure activity in a pipeline, complete the following steps:

1. Search for *Stored Procedure* in the pipeline Activities pane, and drag a Stored Procedure activity to the pipeline canvas.
2. Select the new Stored Procedure activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



3. Select an existing or create a new linked service to an Azure SQL Database, Azure Synapse Analytics, or SQL Server.

4. Choose a stored procedure, and provide any parameters for its execution.

Syntax details

Here is the JSON format for defining a Stored Procedure Activity:

```
{
    "name": "Stored Procedure Activity",
    "description":"Description",
    "type": "SqlServerStoredProcedure",
    "linkedServiceName": {
        "referenceName": "AzureSqlLinkedService",
        "type": "LinkedServiceReference"
    },
    "typeProperties": {
        "storedProcedureName": "usp_sample",
        "storedProcedureParameters": {
            "identifier": { "value": "1", "type": "Int" },
            "stringData": { "value": "str1" }

        }
    }
}
```

The following table describes these JSON properties:

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity	Yes
description	Text describing what the activity is used for	No
type	For Stored Procedure Activity, the activity type is **SqlServerStoredProcedure**	Yes

PROPERTY	DESCRIPTION	REQUIRED
linkedServiceName	Reference to the **Azure SQL Database** or **Azure Synapse Analytics** or **SQL Server** registered as a linked service in Data Factory. To learn about this linked service, see Compute linked services article.	Yes
storedProcedureName	Specify the name of the stored procedure to invoke.	Yes
storedProcedureParameters	Specify the values for stored procedure parameters. Use `"param1": { "value": "param1Value","type":"param1Type" }` to pass parameter values and their type supported by the data source. If you need to pass null for a parameter, use `"param1": { "value": null }` (all lower case).	No

Parameter data type mapping

The data type you specify for the parameter is the internal service type that maps to the data type in the data source you are using. You can find the data type mappings for your data source described in the connectors documentation. For example:

- Azure Synapse Analytics
- Azure SQL Database data type mapping
- Oracle data type mapping
- SQL Server data type mapping

Next steps

See the following articles that explain how to transform data in other ways:

- U-SQL Activity
- Hive Activity
- Pig Activity
- MapReduce Activity
- Hadoop Streaming Activity
- Spark Activity
- .NET custom activity
- Stored procedure activity

Transform data by using the Script activity in Azure Data Factory or Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

You use data transformation activities in a Data Factory or Synapsepipelineto transform and process raw data into predictions and insights. The Script activity is one of the transformation activities that pipelines support. This article builds on thetransform dataarticle, which presents a general overview of data transformation and the supported transformation activities.

Using the script activity, you can execute common operations with Data Manipulation Language (DML), and Data Definition Language (DDL). DML statements like INSERT, UPDATE, DELETE and SELECT let users insert, modify, delete and retrieve data in the database. DDL statements like CREATE, ALTER and DROPallow a database manager to create, modify, and remove database objects such as tables, indexes, and users.

You can use the Script activity to invoke a SQL script in one of the following data stores in your enterprise or on an Azure virtual machine (VM):

- Azure SQL Database
- Azure Synapse Analytics
- SQL Server Database. If you are using SQL Server, install Self-hosted integration runtime on the same machine that hosts the database or on a separate machine that has access to the database. Self-Hosted integration runtime is a component that connects data sources on-premises/on Azure VM with cloud services in a secure and managed way. Seethe Self-hosted integration runtimearticle for details.
- Oracle
- Snowflake

The script may contain either a single SQL statement or multiple SQL statements that run sequentially.You can use the Execute SQL task for the following purposes:

- Truncate a table in preparation for inserting data.
- Create, alter, and drop database objects such as tables and views.
- Re-create fact and dimension tables before loading data into them.
- Run stored procedures. If the SQL statement invokes a stored procedure that returns results from a temporary table, use the WITH RESULT SETS option to define metadata for the result set.
- Save the rowset returned from a query as activity output for downstream consumption.

Syntax details

Here is the JSON format for defining a Script activity:

```
{
   "name": "<activity name>",
   "type": "Script",
   "linkedServiceName": {
      "referenceName": "<name>",
      "type": "LinkedServiceReference"
    },
   "typeProperties": {
      "scripts" : [
         {
            "text": "<Script Block>",
            "type": "<Query> or <NonQuery>",
            "parameters":[
               {
                  "name": "<name>",
                  "value": "<value>",
                  "type": "<type>",
                  "direction": "<Input> or <Output> or <InputOutput>",
                  "size": 256
               },
               ...
            ]
         },
         ...
      ],
         ...
         ]
      },
      "logSettings": {
         "logDestination": "<ActivityOutput> or <ExternalStore>",
         "logLocationSettings":{
            "linkedServiceName":{
               "referenceName": "<name>",
               "type": "<LinkedServiceReference>"
            },
            "path": "<folder path>"
         }
      }
    }
}
```

The following table describes these JSON properties:

PROPERTY NAME	DESCRIPTION	REQUIRED
name	The name of the activity.	Yes
type	The type of the activity, set to "Script".	Yes
typeProperties	Specify properties to configure the Script Activity.	Yes
linkedServiceName	The target database the script runs on. It should be a reference to a linked service.	Yes
scripts	An array of objects to represent the script.	No
scripts.text	The plain text of a block of queries.	No

PROPERTY NAME	DESCRIPTION	REQUIRED
scripts.type	The type of the block of queries. It can be Query or NonQuery. Default: Query.	No
scripts.parameter	The array of parameters of the script.	No
scripts.parameter.name	The name of the parameter.	No
scripts.parameter.value	The value of the parameter.	No
scripts.parameter.type	The data type of the parameter. The type is logical type and follows type mapping of each connector.	No
scripts.parameter.direction	The direction of the parameter. It can be Input, Output, InputOutput. The value is ignored if the direction is Output. ReturnValue type is not supported. Set the return value of SP to an output parameter to retrieve it.	No
scripts.parameter.size	The max size of the parameter. Only applies to Output/InputOutput direction parameter of type string/byte[].	No
logSettings	The settings to store the output logs. If not specified, script log is disabled.	No
logSettings.logDestination	The destination of log output. It can be ActivityOutput or ExternalStore. Default: ActivityOutput.	No
logSettings.logLocationSettings	The settings of the target location if logDestination is ExternalStore.	No
logSettiongs.logLocationSettings.linkedServiceName	The linked service of the target location. Only blob storage is supported.	No
logSettings.logLocationSettings.path	The folder path under which logs will be stored.	No

Activity output

Sample output:

```
{
  "resultSetCount": 2,
  "resultSets": [
    {
      "rowCount": 10,
      "rows":[
        {
          "<columnName1>": "<value1>",
          "<columnName2>": "<value2>",
          ...
        }
      ]
    },
    ...
  ],
  "recordsAffected": 123,
  "outputParameters":{
    "<parameterName1>": "<value1>",
    "<parameterName2>": "<value2>"
  },
  "outputLogs": "<logs>",
  "outputLogsLocation": "<folder path>",
  "outputTruncated": true,
    ...
}
```

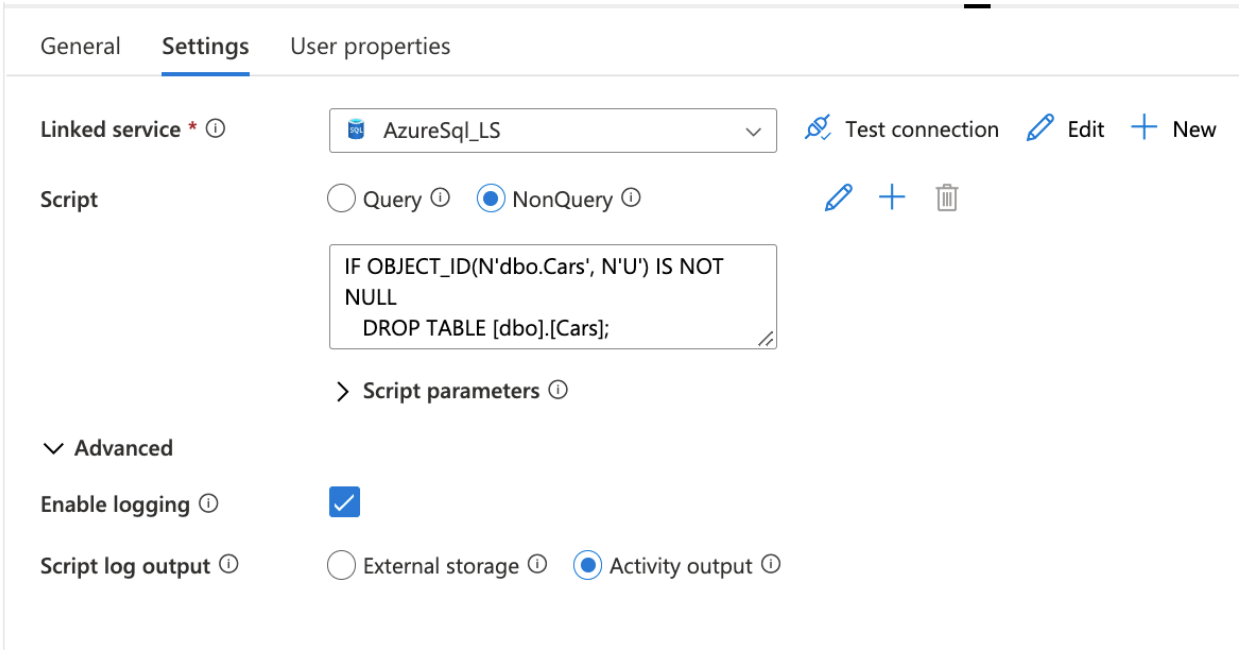
PROPERTY NAME	DESCRIPTION	CONDITION
resultSetCount	The count of result sets returned by the script.	Always
resultSets	The array which contains all the result sets.	Always
resultSets.rowCount	Total rows in the result set.	Always
resultSets.rows	The array of rows in the result set.	Always
recordsAffected	The row count of affected rows by the script.	If scriptType is NonQuery.
outputParameters	The output parameters of the script.	If parameter type is Output or InputOutput.
outputLogs	The logs written by the script, for example, print statement.	If connector supports log statement and enableScriptLogs is true and logLocationSettings is not provided.
outputLogsPath	The full path of the log file.	If enableScriptLogs is true and logLocationSettings is provided.
outputTruncated	Indicator of whether the output exceeds the limits and get truncated.	If output exceeds the limits.

NOTE

- The output is collected every time a script block is executed. The final output is the merged result of all script block outputs. The output parameter with same name in different script block will get overwritten.
- Since the output has size / rows limitation, the output will be truncated in following order: logs -> parameters -> rows. Note, this applies to a single script block, which means the output rows of next script block won't evict previous logs.
- Any error caused by log won't fail the activity.
- For consuming activity output resultSets in down stream activity please refer to the Lookup activity result documentation.
- Use outputLogs when you are using 'PRINT' statements for logging purpose. If query returns resultSets, it will be available in the activity output and will be limited to 5000 rows/ 2MB size limit.

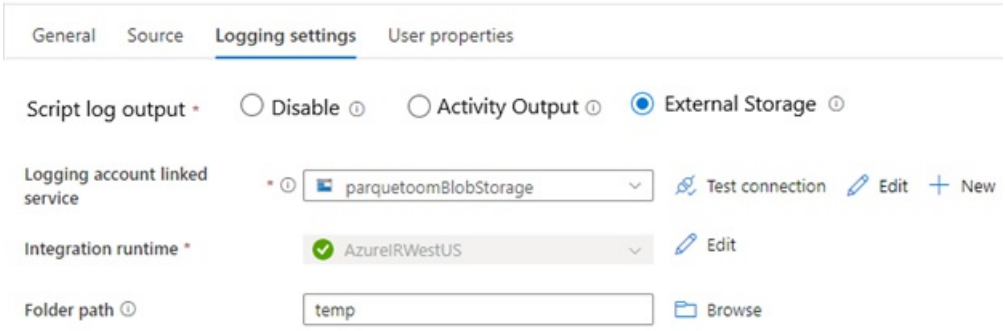
Configure the Script activity using UI

Inline script



Inline scripts integrate well with Pipeline CI/CD since the script is stored as part of the pipeline metadata.

Logging



Logging options:

- *Disable* - No execution output is logged.
- *Activity output* – The script execution output is appended to the activity output. It can be consumed by

downstream activities. The output size is limited to 2MB.

- *External storage* – Persists output to storage. Use this option if the output size is greater than 2MB or you would like to explicitly persist the output on your storage account.

> **NOTE**
>
> **Billing** - The Script activity will be billed as **Pipeline activities**.

Next steps

See the following articles that explain how to transform data in other ways:

- U-SQL activity
- Hive activity
- Pig activity
- MapReduce activity
- Hadoop Streaming activity
- Spark activity
- .NET custom activity
- Stored procedure activity

Compute environments supported by Azure Data Factory and Synapse pipelines

8/19/2022 • 23 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

IMPORTANT

Support for Machine Learning Studio (classic) will end on 31 August 2024. We recommend you transition to Azure Machine Learning by that date.

Beginning 1 December 2021, you will not be able to create new Machine Learning Studio (classic) resources (workspace and web service plan). Through 31 August 2024, you can continue to use the existing Machine Learning Studio (classic) experiments and web services.

- See information on moving machine learning projects from ML Studio (classic) to Azure Machine Learning.
- Learn more about Azure Machine Learning

ML Studio (classic) documentation is being retired and may not be updated in the future.

This article explains different compute environments that you can use to process or transform data. It also provides details about different configurations (on-demand vs. bring your own) supported when configuring linked services linking these compute environments.

The following table provides a list of supported compute environments and the activities that can run on them.

COMPUTE ENVIRONMENT	ACTIVITIES
On-demand HDInsight cluster or your own HDInsight cluster	Hive, Pig, Spark, MapReduce, Hadoop Streaming
Azure Batch	Custom
ML Studio (classic)	ML Studio (classic) activities: Batch Execution and Update Resource
Azure Machine Learning	Azure Machine Learning Execute Pipeline
Azure Data Lake Analytics	Data Lake Analytics U-SQL
Azure SQL, Azure Synapse Analytics, SQL Server	Stored Procedure
Azure Databricks	Notebook, Jar, Python
Azure Function	Azure Function activity

HDInsight compute environment

Refer to below table for details about the supported storage linked service types for configuration in On-demand and BYOC (Bring your own compute) environment.

IN COMPUTE LINKED SERVICE	PROPERTY NAME	DESCRIPTION	BLOB	ADLS GEN2	AZURE SQL DB	ADLS GEN 1
On-demand	linkedService Name	Azure Storage linked service to be used by the on-demand cluster for storing and processing data.	Yes	Yes	No	No
	additionalLinkedServiceNames	Specifies additional storage accounts for the HDInsight linked service so that the service can register them on your behalf.	Yes	No	No	No
	hcatalogLinkedServiceName	The name of Azure SQL linked service that point to the HCatalog database. The on-demand HDInsight cluster is created by using the Azure SQL database as the metastore.	No	No	Yes	No
BYOC	linkedService Name	The Azure Storage linked service reference.	Yes	Yes	No	No
	additionalLinkedServiceNames	Specifies additional storage accounts for the HDInsight linked service so that the service can register them on your behalf.	No	No	No	No

IN COMPUTE LINKED SERVICE	PROPERTY NAME	DESCRIPTION	BLOB	ADLS GEN2	AZURE SQL DB	ADLS GEN 1
	hcatalogLinkedServiceName	A reference to the Azure SQL linked service that points to the HCatalog database.	No	No	No	No

Azure HDInsight on-demand linked service

In this type of configuration, the computing environment is fully managed by the service. It is automatically created by the service before a job is submitted to process data and removed when the job is completed. You can create a linked service for the on-demand compute environment, configure it, and control granular settings for job execution, cluster management, and bootstrapping actions.

> **NOTE**
> The on-demand configuration is currently supported only for Azure HDInsight clusters. Azure Databricks also supports on-demand jobs using job clusters. For more information, see Azure databricks linked service.

The service can automatically create an on-demand HDInsight cluster to process data. The cluster is created in the same region as the storage account (linkedServiceName property in the JSON) associated with the cluster. The storage account `must` be a general-purpose standard Azure Storage account.

Note the following **important** points about on-demand HDInsight linked service:

- The on-demand HDInsight cluster is created under your Azure subscription. You are able to see the cluster in your Azure portal when the cluster is up and running.
- The logs for jobs that are run on an on-demand HDInsight cluster are copied to the storage account associated with the HDInsight cluster. The clusterUserName, clusterPassword, clusterSshUserName, clusterSshPassword defined in your linked service definition are used to log in to the cluster for in-depth troubleshooting during the lifecycle of the cluster.
- You are charged only for the time when the HDInsight cluster is up and running jobs.
- You can use a **Script Action** with the Azure HDInsight on-demand linked service.

> **IMPORTANT**
> It typically takes **20 minutes** or more to provision an Azure HDInsight cluster on demand.

Example

The following JSON defines a Linux-based on-demand HDInsight linked service. The service automatically creates a **Linux-based** HDInsight cluster to process the required activity.

```
{
  "name": "HDInsightOnDemandLinkedService",
  "properties": {
    "type": "HDInsightOnDemand",
    "typeProperties": {
      "clusterType": "hadoop",
      "clusterSize": 1,
      "timeToLive": "00:15:00",
      "hostSubscriptionId": "<subscription ID>",
      "servicePrincipalId": "<service principal ID>",
      "servicePrincipalKey": {
        "value": "<service principal key>",
        "type": "SecureString"
      },
      "tenant": "<tenent id>",
      "clusterResourceGroup": "<resource group name>",
      "version": "3.6",
      "osType": "Linux",
      "linkedServiceName": {
        "referenceName": "AzureStorageLinkedService",
        "type": "LinkedServiceReference"
      }
    },
    "connectVia": {
      "referenceName": "<name of Integration Runtime>",
      "type": "IntegrationRuntimeReference"
    }
  }
}
```

IMPORTANT

The HDInsight cluster creates a **default container** in the blob storage you specified in the JSON (**linkedServiceName**). HDInsight does not delete this container when the cluster is deleted. This behavior is by design. With on-demand HDInsight linked service, a HDInsight cluster is created every time a slice needs to be processed unless there is an existing live cluster (**timeToLive**) and is deleted when the processing is done.

As more activity runs, you see many containers in your Azure blob storage. If you do not need them for troubleshooting of the jobs, you may want to delete them to reduce the storage cost. The names of these containers follow a pattern: `adf**yourfactoryorworkspacename**-**linkedservicename**-datetimestamp`. Use tools such as Microsoft Azure Storage Explorer to delete containers in your Azure blob storage.

Properties

PROPERTY	DESCRIPTION	REQUIRED
type	The type property should be set to **HDInsightOnDemand**.	Yes
clusterSize	Number of worker/data nodes in the cluster. The HDInsight cluster is created with 2 head nodes along with the number of worker nodes you specify for this property. The nodes are of size Standard_D3 that has 4 cores, so a 4 worker node cluster takes 24 cores (4*4 = 16 cores for worker nodes, plus 2*4 = 8 cores for head nodes). See Set up clusters in HDInsight with Hadoop, Spark, Kafka, and more for details.	Yes

PROPERTY	DESCRIPTION	REQUIRED
linkedServiceName	Azure Storage linked service to be used by the on-demand cluster for storing and processing data. The HDInsight cluster is created in the same region as this Azure Storage account. Azure HDInsight has limitation on the total number of cores you can use in each Azure region it supports. Make sure you have enough core quotas in that Azure region to meet the required clusterSize. For details, refer to Set up clusters in HDInsight with Hadoop, Spark, Kafka, and more Currently, you cannot create an on-demand HDInsight cluster that uses an Azure Data Lake Storage (Gen 2) as the storage. If you want to store the result data from HDInsight processing in an Azure Data Lake Storage (Gen 2), use a Copy Activity to copy the data from the Azure Blob Storage to the Azure Data Lake Storage (Gen 2).	Yes
clusterResourceGroup	The HDInsight cluster is created in this resource group.	Yes

PROPERTY	DESCRIPTION	REQUIRED
timetolive	The allowed idle time for the on-demand HDInsight cluster. Specifies how long the on-demand HDInsight cluster stays alive after completion of an activity run if there are no other active jobs in the cluster. The minimal allowed value is 5 minutes (00:05:00). For example, if an activity run takes 6 minutes and timetolive is set to 5 minutes, the cluster stays alive for 5 minutes after the 6 minutes of processing the activity run. If another activity run is executed with the 6-minutes window, it is processed by the same cluster. Creating an on-demand HDInsight cluster is an expensive operation (could take a while), so use this setting as needed to improve performance of the service by reusing an on-demand HDInsight cluster. If you set timetolive value to 0, the cluster is deleted as soon as the activity run completes. Whereas, if you set a high value, the cluster may stay idle for you to log on for some troubleshooting purpose but it could result in high costs. Therefore, it is important that you set the appropriate value based on your needs. If the timetolive property value is appropriately set, multiple pipelines can share the instance of the on-demand HDInsight cluster.	Yes
clusterType	The type of the HDInsight cluster to be created. Allowed values are "hadoop" and "spark". If not specified, default value is hadoop. Enterprise Security Package enabled cluster cannot be created on-demand, instead use an existing cluster/ bring your own compute.	No
version	Version of the HDInsight cluster. If not specified, it's using the current HDInsight defined default version.	No
hostSubscriptionId	The Azure subscription ID used to create HDInsight cluster. If not specified, it uses the Subscription ID of your Azure login context.	No

PROPERTY	DESCRIPTION	REQUIRED
clusterNamePrefix	The prefix of HDI cluster name, a timestamp automatically appends at the end of the cluster name	No
sparkVersion	The version of spark if the cluster type is "Spark"	No
additionalLinkedServiceNames	Specifies additional storage accounts for the HDInsight linked service so that the service can register them on your behalf. These storage accounts must be in the same region as the HDInsight cluster, which is created in the same region as the storage account specified by linkedServiceName.	No
osType	Type of operating system. Allowed values are: Linux and Windows (for HDInsight 3.3 only). Default is Linux.	No
hcatalogLinkedServiceName	The name of Azure SQL linked service that point to the HCatalog database. The on-demand HDInsight cluster is created by using the Azure SQL Database as the metastore.	No
connectVia	The Integration Runtime to be used to dispatch the activities to this HDInsight linked service. For on-demand HDInsight linked service, it only supports Azure Integration Runtime. If not specified, it uses the default Azure Integration Runtime.	No
clusterUserName	The username to access the cluster.	No
clusterPassword	The password in type of secure string to access the cluster.	No
clusterSshUserName	The username to SSH remotely connects to cluster's node (for Linux).	No
clusterSshPassword	The password in type of secure string to SSH remotely connect cluster's node (for Linux).	No
scriptActions	Specify script for HDInsight cluster customizations during on-demand cluster creation. Currently, the UI authoring tool supports specifying only 1 script action, but you can get through this limitation in the JSON (specify multiple script actions in the JSON).	No

IMPORTANT

HDInsight supports multiple Hadoop cluster versions that can be deployed. Each version choice creates a specific version of the Hortonworks Data Platform (HDP) distribution and a set of components that are contained within that distribution. The list of supported HDInsight versions keeps being updated to provide latest Hadoop ecosystem components and fixes. Make sure you always refer to latest information of Supported HDInsight version and OS Type to ensure you are using supported version of HDInsight.

IMPORTANT

Currently, HDInsight linked services does not support HBase, Interactive Query (Hive LLAP), Storm.

- additionalLinkedServiceNames JSON example

```
"additionalLinkedServiceNames": [{
    "referenceName": "MyStorageLinkedService2",
    "type": "LinkedServiceReference"
}]
```

Service principal authentication

The On-Demand HDInsight linked service requires a service principal authentication to create HDInsight clusters on your behalf. To use service principal authentication, register an application entity in Azure Active Directory (Azure AD) and grant it the **Contributor** role of the subscription or the resource group in which the HDInsight cluster is created. For detailed steps, see Use portal to create an Azure Active Directory application and service principal that can access resources. Make note of the following values, which you use to define the linked service:

- Application ID
- Application key
- Tenant ID

Use service principal authentication by specifying the following properties:

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key.	Yes
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. You can retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Yes

Advanced Properties

You can also specify the following properties for the granular configuration of the on-demand HDInsight cluster.

PROPERTY	DESCRIPTION	REQUIRED
coreConfiguration	Specifies the core configuration parameters (as in core-site.xml) for the HDInsight cluster to be created.	No

PROPERTY	DESCRIPTION	REQUIRED
hBaseConfiguration	Specifies the HBase configuration parameters (hbase-site.xml) for the HDInsight cluster.	No
hdfsConfiguration	Specifies the HDFS configuration parameters (hdfs-site.xml) for the HDInsight cluster.	No
hiveConfiguration	Specifies the hive configuration parameters (hive-site.xml) for the HDInsight cluster.	No
mapReduceConfiguration	Specifies the MapReduce configuration parameters (mapred-site.xml) for the HDInsight cluster.	No
oozieConfiguration	Specifies the Oozie configuration parameters (oozie-site.xml) for the HDInsight cluster.	No
stormConfiguration	Specifies the Storm configuration parameters (storm-site.xml) for the HDInsight cluster.	No
yarnConfiguration	Specifies the Yarn configuration parameters (yarn-site.xml) for the HDInsight cluster.	No

- Example – On-demand HDInsight cluster configuration with advanced properties

```
{
    "name": " HDInsightOnDemandLinkedService",
    "properties": {
      "type": "HDInsightOnDemand",
      "typeProperties": {
          "clusterSize": 16,
          "timeToLive": "01:30:00",
          "hostSubscriptionId": "<subscription ID>",
          "servicePrincipalId": "<service principal ID>",
          "servicePrincipalKey": {
            "value": "<service principal key>",
            "type": "SecureString"
          },
          "tenant": "<tenent id>",
          "clusterResourceGroup": "<resource group name>",
          "version": "3.6",
          "osType": "Linux",
          "linkedServiceName": {
              "referenceName": "AzureStorageLinkedService",
              "type": "LinkedServiceReference"
            },
            "coreConfiguration": {
                "templeton.mapper.memory.mb": "5000"
            },
            "hiveConfiguration": {
                "templeton.mapper.memory.mb": "5000"
            },
            "mapReduceConfiguration": {
                "mapreduce.reduce.java.opts": "-Xmx4000m",
                "mapreduce.map.java.opts": "-Xmx4000m",
                "mapreduce.map.memory.mb": "5000",
                "mapreduce.reduce.memory.mb": "5000",
                "mapreduce.job.reduce.slowstart.completedmaps": "0.8"
            },
            "yarnConfiguration": {
                "yarn.app.mapreduce.am.resource.mb": "5000",
                "mapreduce.map.memory.mb": "5000"
            },
            "additionalLinkedServiceNames": [{
                "referenceName": "MyStorageLinkedService2",
                "type": "LinkedServiceReference"
            }]
        }
    },
      "connectVia": {
      "referenceName": "<name of Integration Runtime>",
      "type": "IntegrationRuntimeReference"
    }
}
```

Node sizes

You can specify the sizes of head, data, and zookeeper nodes using the following properties:

PROPERTY	DESCRIPTION	REQUIRED
headNodeSize	Specifies the size of the head node. The default value is: Standard_D3. See the **Specifying node sizes** section for details.	No
dataNodeSize	Specifies the size of the data node. The default value is: Standard_D3.	No

PROPERTY	DESCRIPTION	REQUIRED
zookeeperNodeSize	Specifies the size of the Zoo Keeper node. The default value is: Standard_D3.	No

- Specifying node sizes See the Sizes of Virtual Machines article for string values you need to specify for the properties mentioned in the previous section. The values need to conform to the **CMDLETs & APIS** referenced in the article. As you can see in the article, the data node of Large (default) size has 7-GB memory, which may not be good enough for your scenario.

If you want to create D4 sized head nodes and worker nodes, specify **Standard_D4** as the value for headNodeSize and dataNodeSize properties.

```
"headNodeSize": "Standard_D4",
"dataNodeSize": "Standard_D4",
```

If you specify a wrong value for these properties, you may receive the following **error:** Failed to create cluster. Exception: Unable to complete the cluster create operation. Operation failed with code '400'. Cluster left behind state: 'Error'. Message: 'PreClusterCreationValidationFailure'. When you receive this error, ensure that you are using the **CMDLET & APIS** name from the table in the Sizes of Virtual Machines article.

Bring your own compute environment

In this type of configuration, users can register an already existing computing environment as a linked service. The computing environment is managed by the user and the service uses it to execute the activities.

This type of configuration is supported for the following compute environments:

- Azure HDInsight
- Azure Batch
- Azure Machine Learning
- Azure Data Lake Analytics
- Azure SQL DB, Azure Synapse Analytics, SQL Server

Azure HDInsight linked service

You can create an Azure HDInsight linked service to register your own HDInsight cluster with a data factory or Synapse workspace.

Example

```
{
    "name": "HDInsightLinkedService",
    "properties": {
      "type": "HDInsight",
      "typeProperties": {
        "clusterUri": " https://<hdinsightclustername>.azurehdinsight.net/",
        "userName": "username",
        "password": {
            "value": "passwordvalue",
            "type": "SecureString"
          },
        "linkedServiceName": {
              "referenceName": "AzureStorageLinkedService",
              "type": "LinkedServiceReference"
        }
      },
      "connectVia": {
        "referenceName": "<name of Integration Runtime>",
        "type": "IntegrationRuntimeReference"
      }
    }
  }
```

Properties

PROPERTY	DESCRIPTION	REQUIRED
type	The type property should be set to **HDInsight**.	Yes
clusterUri	The URI of the HDInsight cluster.	Yes
username	Specify the name of the user to be used to connect to an existing HDInsight cluster.	Yes
password	Specify password for the user account.	Yes
linkedServiceName	Name of the Azure Storage linked service that refers to the Azure blob storage used by the HDInsight cluster. Currently, you cannot specify an Azure Data Lake Storage (Gen 2) linked service for this property. If the HDInsight cluster has access to the Data Lake Store, you may access data in the Azure Data Lake Storage (Gen 2) from Hive/Pig scripts.	Yes
isEspEnabled	Specify '*true*' if the HDInsight cluster is Enterprise Security Package enabled. Default is '*false*'.	No

PROPERTY	DESCRIPTION	REQUIRED
connectVia	The Integration Runtime to be used to dispatch the activities to this linked service. You can use Azure Integration Runtime or Self-hosted Integration Runtime. If not specified, it uses the default Azure Integration Runtime. For Enterprise Security Package (ESP) enabled HDInsight cluster use a self-hosted integration runtime, which has a line of sight to the cluster or it should be deployed inside the same Virtual Network as the ESP HDInsight cluster.	No

IMPORTANT

HDInsight supports multiple Hadoop cluster versions that can be deployed. Each version choice creates a specific version of the Hortonworks Data Platform (HDP) distribution and a set of components that are contained within that distribution. The list of supported HDInsight versions keeps being updated to provide latest Hadoop ecosystem components and fixes. Make sure you always refer to latest information of Supported HDInsight version and OS Type to ensure you are using supported version of HDInsight.

IMPORTANT

Currently, HDInsight linked services does not support HBase, Interactive Query (Hive LLAP), Storm.

Azure Batch linked service

NOTE

This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

You can create an Azure Batch linked service to register a Batch pool of virtual machines (VMs) to a data or Synapse workspace. You can run Custom activity using Azure Batch.

See following articles if you are new to Azure Batch service:

- Azure Batch basics for an overview of the Azure Batch service.
- New-AzBatchAccount cmdlet to create an Azure Batch account (or) Azure portal to create the Azure Batch account using Azure portal. See Using PowerShell to manage Azure Batch Account article for detailed instructions on using the cmdlet.
- New-AzBatchPool cmdlet to create an Azure Batch pool.

IMPORTANT

When creating a new Azure Batch pool, 'VirtualMachineConfiguration' must be used and NOT 'CloudServiceConfiguration'. For more details refer Azure Batch Pool migration guidance.

Example

```
{
    "name": "AzureBatchLinkedService",
    "properties": {
      "type": "AzureBatch",
      "typeProperties": {
        "accountName": "batchaccount",
        "accessKey": {
          "type": "SecureString",
          "value": "access key"
        },
        "batchUri": "https://batchaccount.region.batch.azure.com",
        "poolName": "poolname",
        "linkedServiceName": {
          "referenceName": "StorageLinkedService",
          "type": "LinkedServiceReference"
        }
      },
      "connectVia": {
        "referenceName": "<name of Integration Runtime>",
        "type": "IntegrationRuntimeReference"
      }
    }
  }
```

Properties

PROPERTY	DESCRIPTION	REQUIRED
type	The type property should be set to **AzureBatch**.	Yes
accountName	Name of the Azure Batch account.	Yes
accessKey	Access key for the Azure Batch account.	Yes
batchUri	URL to your Azure Batch account, in format of https://*batchaccountname.region*.batch.azure.com.	Yes
poolName	Name of the pool of virtual machines.	Yes
linkedServiceName	Name of the Azure Storage linked service associated with this Azure Batch linked service. This linked service is used for staging files required to run the activity.	Yes
connectVia	The Integration Runtime to be used to dispatch the activities to this linked service. You can use Azure Integration Runtime or Self-hosted Integration Runtime. If not specified, it uses the default Azure Integration Runtime.	No

Machine Learning Studio (classic) linked service

IMPORTANT

Support for Machine Learning Studio (classic) will end on 31 August 2024. We recommend you transition to Azure Machine Learning by that date.

Beginning 1 December 2021, you will not be able to create new Machine Learning Studio (classic) resources (workspace and web service plan). Through 31 August 2024, you can continue to use the existing Machine Learning Studio (classic) experiments and web services.

- See information on moving machine learning projects from ML Studio (classic) to Azure Machine Learning.
- Learn more about Azure Machine Learning

ML Studio (classic) documentation is being retired and may not be updated in the future.

You create an Machine Learning Studio (classic) linked service to register a Machine Learning Studio (classic) batch scoring endpoint to a data factory or Synapse workspace.

Example

```
{
    "name": "AzureMLLinkedService",
    "properties": {
      "type": "AzureML",
      "typeProperties": {
        "mlEndpoint": "https://[batch scoring endpoint]/jobs",
        "apiKey": {
            "type": "SecureString",
            "value": "access key"
        }
     },
     "connectVia": {
        "referenceName": "<name of Integration Runtime>",
        "type": "IntegrationRuntimeReference"
      }
    }
}
```

Properties

PROPERTY	DESCRIPTION	REQUIRED
Type	The type property should be set to: **AzureML**.	Yes
mlEndpoint	The batch scoring URL.	Yes
apiKey	The published workspace model's API.	Yes
updateResourceEndpoint	The Update Resource URL for an ML Studio (classic) Web Service endpoint used to update the predictive Web Service with trained model file	No
servicePrincipalId	Specify the application's client ID.	Required if updateResourceEndpoint is specified
servicePrincipalKey	Specify the application's key.	Required if updateResourceEndpoint is specified

PROPERTY	DESCRIPTION	REQUIRED
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. You can retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Required if updateResourceEndpoint is specified
connectVia	The Integration Runtime to be used to dispatch the activities to this linked service. You can use Azure Integration Runtime or Self-hosted Integration Runtime. If not specified, it uses the default Azure Integration Runtime.	No

Azure Machine Learning linked service

You create an Azure Machine Learning linked service to connect an Azure Machine Learning workspace to a data factory or Synapse workspace.

> **NOTE**
> Currently only service principal authentication is supported for the Azure Machine Learning linked service.

Example

```
{
    "name": "AzureMLServiceLinkedService",
    "properties": {
        "type": "AzureMLService",
        "typeProperties": {
            "subscriptionId": "subscriptionId",
            "resourceGroupName": "resourceGroupName",
            "mlWorkspaceName": "mlWorkspaceName",
            "servicePrincipalId": "service principal id",
            "servicePrincipalKey": {
                "value": "service principal key",
                "type": "SecureString"
            },
            "tenant": "tenant ID"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime?",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Properties

PROPERTY	DESCRIPTION	REQUIRED
Type	The type property should be set to: **AzureMLService**.	Yes
subscriptionId	Azure subscription ID	Yes
resourceGroupName	name	Yes

PROPERTY	DESCRIPTION	REQUIRED
mlWorkspaceName	Azure Machine Learning workspace name	Yes
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key.	Yes
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. You can retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Required if updateResourceEndpoint is specified
connectVia	The Integration Runtime to be used to dispatch the activities to this linked service. You can use Azure Integration Runtime or Self-hosted Integration Runtime. If not specified, it uses the default Azure Integration Runtime.	No

Azure Data Lake Analytics linked service

You create an **Azure Data Lake Analytics** linked service to link an Azure Data Lake Analytics compute service to a data factory or Synapse workspace. The Data Lake Analytics U-SQL activity in the pipeline refers to this linked service.

Example

```
{
    "name": "AzureDataLakeAnalyticsLinkedService",
    "properties": {
        "type": "AzureDataLakeAnalytics",
        "typeProperties": {
            "accountName": "adftestaccount",
            "dataLakeAnalyticsUri": "azuredatalakeanalytics URI",
            "servicePrincipalId": "service principal id",
            "servicePrincipalKey": {
                "value": "service principal key",
                "type": "SecureString"
            },
            "tenant": "tenant ID",
            "subscriptionId": "<optional, subscription ID of ADLA>",
            "resourceGroupName": "<optional, resource group name of ADLA>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Properties

PROPERTY	DESCRIPTION	REQUIRED
type	The type property should be set to: **AzureDataLakeAnalytics**.	Yes

PROPERTY	DESCRIPTION	REQUIRED
accountName	Azure Data Lake Analytics Account Name.	Yes
dataLakeAnalyticsUri	Azure Data Lake Analytics URI.	No
subscriptionId	Azure subscription ID	No
resourceGroupName	Azure resource group name	No
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key.	Yes
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. You can retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Yes
connectVia	The Integration Runtime to be used to dispatch the activities to this linked service. You can use Azure Integration Runtime or Self-hosted Integration Runtime. If not specified, it uses the default Azure Integration Runtime.	No

Azure Databricks linked service

You can create **Azure Databricks linked service** to register Databricks workspace that you use to run the Databricks workloads(notebook, jar, python).

> **IMPORTANT**
> Databricks linked services supports Instance pools & System-assigned managed identity authentication.

Example - Using new job cluster in Databricks

```
{
    "name": "AzureDatabricks_LS",
    "properties": {
        "type": "AzureDatabricks",
        "typeProperties": {
            "domain": "https://eastus.azuredatabricks.net",
            "newClusterNodeType": "Standard_D3_v2",
            "newClusterNumOfWorker": "1:10",
            "newClusterVersion": "4.0.x-scala2.11",
            "accessToken": {
                "type": "SecureString",
                "value": "dapif33c9c721144c3a790b35000b57f7124f"
            }
        }
    }
}
```

Example - Using existing Interactive cluster in Databricks

```
{
    "name": " AzureDataBricksLinedService",
    "properties": {
      "type": " AzureDatabricks",
      "typeProperties": {
        "domain": "https://westeurope.azuredatabricks.net",
        "accessToken": {
            "type": "SecureString",
            "value": "dapif33c9c72344c3a790b35000b57f7124f"
          },
        "existingClusterId": "{clusterId}"
        }
}
```

Properties

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the Linked Service	Yes
type	The type property should be set to: **Azure Databricks**.	Yes
domain	Specify the Azure Region accordingly based on the region of the Databricks workspace. Example: https://eastus.azuredatabricks.net	Yes
accessToken	Access token is required for the service to authenticate to Azure Databricks. Access token needs to be generated from the databricks workspace. More detailed steps to find the access token can be found here	No
MSI	Use the service's managed identity (system-assigned) to authenticate to Azure Databricks. You do not need Access Token when using 'MSI' authentication. More details about Managed Identity authentication can be found here	No
existingClusterId	Cluster ID of an existing cluster to run all jobs on this. This should be an already created Interactive Cluster. You may need to manually restart the cluster if it stops responding. Databricks suggest running jobs on new clusters for greater reliability. You can find the Cluster ID of an Interactive Cluster on Databricks workspace -> Clusters -> Interactive Cluster Name -> Configuration -> Tags. More details	No

PROPERTY	DESCRIPTION	REQUIRED
instancePoolId	Instance Pool ID of an existing pool in databricks workspace.	No
newClusterVersion	The Spark version of the cluster. It creates a job cluster in databricks.	No
newClusterNumOfWorker	Number of worker nodes that this cluster should have. A cluster has one Spark Driver and num_workers Executors for a total of num_workers + 1 Spark nodes. A string formatted Int32, like "1" means numOfWorker is 1 or "1:10" means autoscale from 1 as min and 10 as max.	No
newClusterNodeType	This field encodes, through a single value, the resources available to each of the Spark nodes in this cluster. For example, the Spark nodes can be provisioned and optimized for memory or compute intensive workloads. This field is required for new cluster	No
newClusterSparkConf	a set of optional, user-specified Spark configuration key-value pairs. Users can also pass in a string of extra JVM options to the driver and the executors via spark.driver.extraJavaOptions and spark.executor.extraJavaOptions respectively.	No
newClusterInitScripts	a set of optional, user-defined initialization scripts for the new cluster. Specifying the DBFS path to the init scripts.	No

Azure SQL Database linked service

You create an Azure SQL linked service and use it with the Stored Procedure Activity to invoke a stored procedure from a pipeline. See Azure SQL Connector article for details about this linked service.

Azure Synapse Analytics linked service

You create an Azure Synapse Analytics linked service and use it with the Stored Procedure Activity to invoke a stored procedure from a pipeline. See Azure Synapse Analytics Connector article for details about this linked service.

SQL Server linked service

You create a SQL Server linked service and use it with the Stored Procedure Activity to invoke a stored procedure from a pipeline. See SQL Server connector article for details about this linked service.

Azure Function linked service

You create an Azure Function linked service and use it with the Azure Function activity to run Azure Functions in

a pipeline. The return type of the Azure function has to be a valid `JObject`. (Keep in mind that JArray is *not* a `JObject`.) Any return type other than `JObject` fails and raises the user error *Response Content is not a valid JObject*.

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to: **AzureFunction**	yes
function app url	URL for the Azure Function App. Format is `https://<accountname>.azurewebsites.net`. This URL is the value under **URL** section when viewing your Function App in the Azure portal	yes
function key	Access key for the Azure Function. Click on the **Manage** section for the respective function, and copy either the **Function Key** or the **Host key**. Find out more here: Azure Functions HTTP triggers and bindings	yes

Next steps

For a list of the supported transformation activities, see Transform data.

Append Variable activity in Azure Data Factory and Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

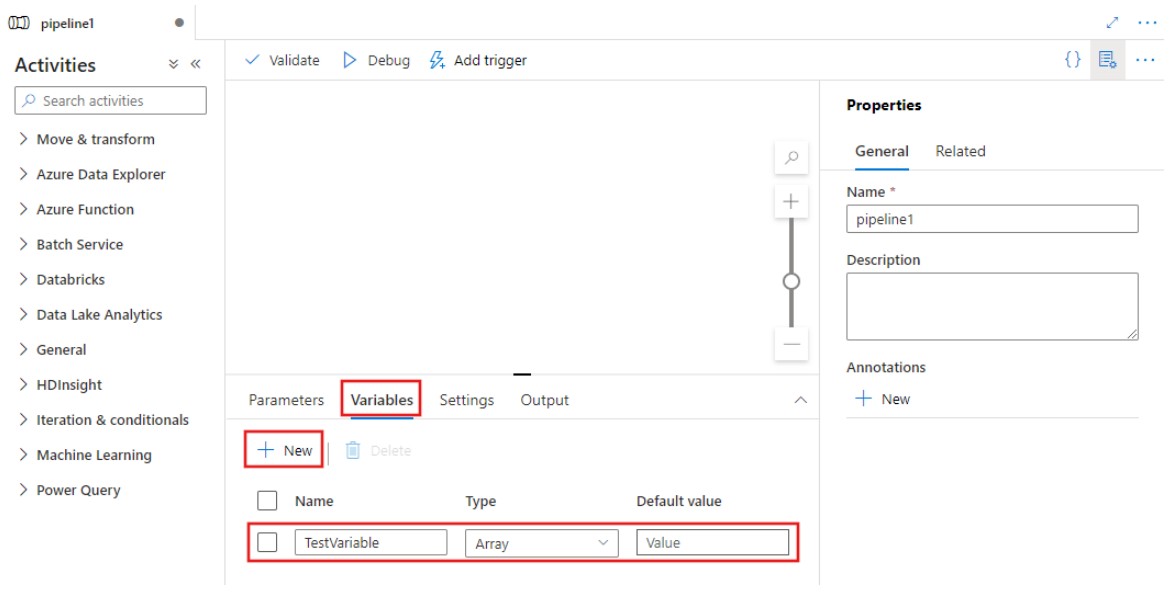
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Use the Append Variable activity to add a value to an existing array variable defined in a Data Factory or Synapse Analytics pipeline.

Create an Append Variable activity with UI

To use a Append Variable activity in a pipeline, complete the following steps:

1. Select the background of the pipeline canvas and use the Variables tab to add an array type variable:

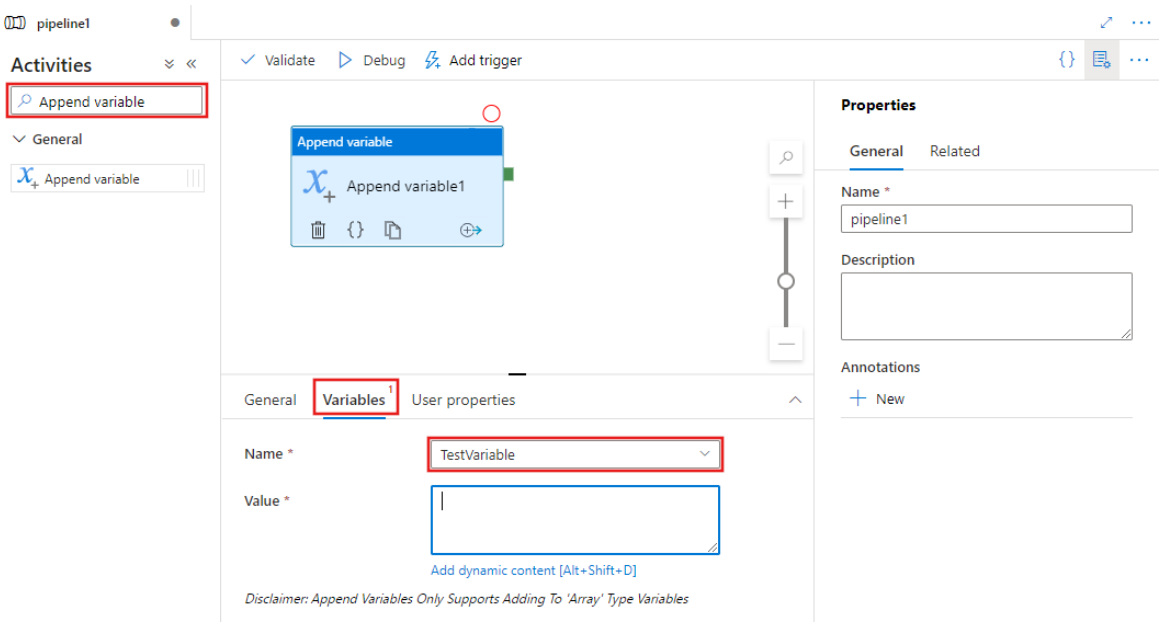


2. Search for *Append Variable* in the pipeline Activities pane, and drag an Append Variable activity to the pipeline canvas.

3. Select the Append Variable activity on the canvas if it is not already selected, and its **Variables** tab, to edit its details.

4. Select the variable for the Name property.

5. Enter an expression for the value, which will be appended to the array in the variable. This can be a literal string expression, or any combination of dynamic expressions, functions, system variables, or outputs from other activities.



Type properties

PROPERTY	DESCRIPTION	REQUIRED
Name	Name of the activity in pipeline	Yes
Description	Text describing what the activity does	No
Type	Activity Type is AppendVariable	Yes
Value	String literal or expression object value used to append into specified variable	Yes
VariableName	Name of the variable that will be modified by activity, the variable must be of type 'Array'	Yes

Next steps

Learn about a related control flow activity:

- Set Variable Activity

Execute Pipeline activity in Azure Data Factory and Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

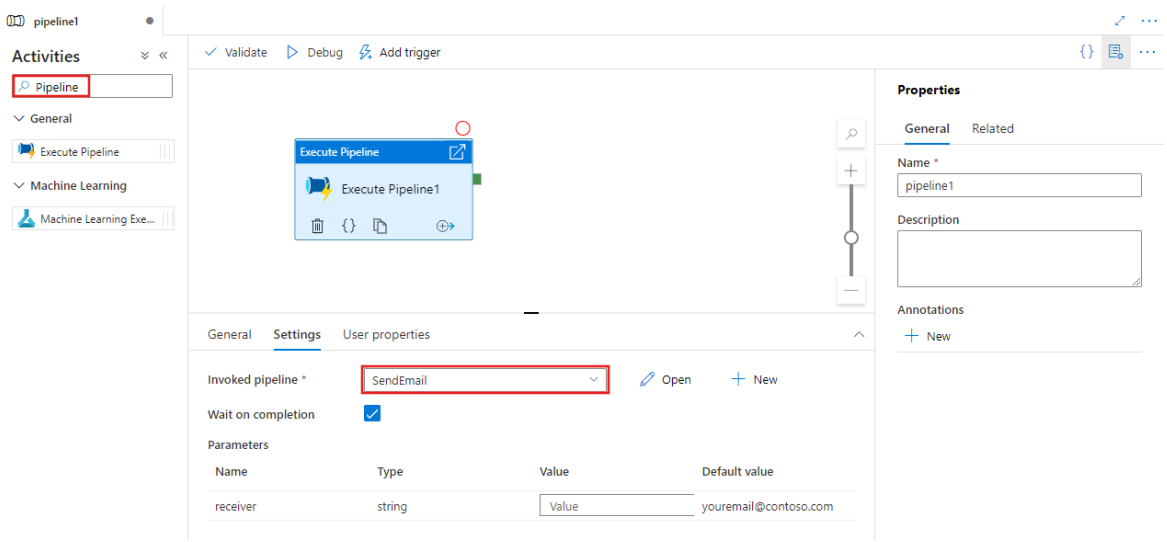
The Execute Pipeline activity allows a Data Factory or Synapse pipeline to invoke another pipeline.

Create an Execute Pipeline activity with UI

To use an Execute Pipeline activity in a pipeline, complete the following steps:

1. Search for *pipeline* in the pipeline Activities pane, and drag an Execute Pipeline activity to the pipeline canvas.

2. Select the new Execute Pipeline activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



3. Select an existing pipeline or create a new one using the New button. Select other options and configure any parameters for the pipeline as required to complete your configuration.

Syntax

```
{
    "name": "MyPipeline",
    "properties": {
        "activities": [
            {
                "name": "ExecutePipelineActivity",
                "type": "ExecutePipeline",
                "typeProperties": {
                    "parameters": {
                        "mySourceDatasetFolderPath": {
                            "value": "@pipeline().parameters.mySourceDatasetFolderPath",
                            "type": "Expression"
                        }
                    },
                    "pipeline": {
                        "referenceName": "<InvokedPipelineName>",
                        "type": "PipelineReference"
                    },
                    "waitOnCompletion": true
                }
            }
        ],
        "parameters": [
            {
                "mySourceDatasetFolderPath": {
                    "type": "String"
                }
            }
        ]
    }
}
```

Type properties

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
name	Name of the execute pipeline activity.	String	Yes
type	Must be set to: **ExecutePipeline**.	String	Yes
pipeline	Pipeline reference to the dependent pipeline that this pipeline invokes. A pipeline reference object has two properties: **referenceName** and **type**. The referenceName property specifies the name of the reference pipeline. The type property must be set to PipelineReference.	PipelineReference	Yes
parameters	Parameters to be passed to the invoked pipeline	A JSON object that maps parameter names to argument values	No

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
waitOnCompletion	Defines whether activity execution waits for the dependent pipeline execution to finish. Default is true.	Boolean	No

Sample

This scenario has two pipelines:

- **Master pipeline** - This pipeline has one Execute Pipeline activity that calls the invoked pipeline. The master pipeline takes two parameters: `masterSourceBlobContainer`, `masterSinkBlobContainer`.
- **Invoked pipeline** - This pipeline has one Copy activity that copies data from an Azure Blob source to Azure Blob sink. The invoked pipeline takes two parameters: `sourceBlobContainer`, `sinkBlobContainer`.

Master pipeline definition

```
{
  "name": "masterPipeline",
  "properties": {
    "activities": [
      {
        "type": "ExecutePipeline",
        "typeProperties": {
          "pipeline": {
            "referenceName": "invokedPipeline",
            "type": "PipelineReference"
          },
          "parameters": {
            "sourceBlobContainer": {
              "value": "@pipeline().parameters.masterSourceBlobContainer",
              "type": "Expression"
            },
            "sinkBlobContainer": {
              "value": "@pipeline().parameters.masterSinkBlobContainer",
              "type": "Expression"
            }
          },
          "waitOnCompletion": true
        },
        "name": "MyExecutePipelineActivity"
      }
    ],
    "parameters": {
      "masterSourceBlobContainer": {
        "type": "String"
      },
      "masterSinkBlobContainer": {
        "type": "String"
      }
    }
  }
}
```

Invoked pipeline definition

```
{
  "name": "invokedPipeline",
  "properties": {
    "activities": [
      {
        "type": "Copy",
        "typeProperties": {
          "source": {
            "type": "BlobSource"
          },
          "sink": {
            "type": "BlobSink"
          }
        },
        "name": "CopyBlobtoBlob",
        "inputs": [
          {
            "referenceName": "SourceBlobDataset",
            "type": "DatasetReference"
          }
        ],
        "outputs": [
          {
            "referenceName": "sinkBlobDataset",
            "type": "DatasetReference"
          }
        ]
      }
    ],
    "parameters": {
      "sourceBlobContainer": {
        "type": "String"
      },
      "sinkBlobContainer": {
        "type": "String"
      }
    }
  }
}
```

Linked service

```
{
    "name": "BlobStorageLinkedService",
    "properties": {
    "type": "AzureStorage",
    "typeProperties": {
      "connectionString": "DefaultEndpointsProtocol=https;AccountName=*****;AccountKey=*****"
    }
  }
}
```

Source dataset

```
{
    "name": "SourceBlobDataset",
    "properties": {
    "type": "AzureBlob",
    "typeProperties": {
      "folderPath": {
        "value": "@pipeline().parameters.sourceBlobContainer",
        "type": "Expression"
      },
      "fileName": "salesforce.txt"
    },
    "linkedServiceName": {
      "referenceName": "BlobStorageLinkedService",
      "type": "LinkedServiceReference"
    }
  }
}
```

Sink dataset

```
{
    "name": "sinkBlobDataset",
    "properties": {
    "type": "AzureBlob",
    "typeProperties": {
      "folderPath": {
        "value": "@pipeline().parameters.sinkBlobContainer",
        "type": "Expression"
      }
    },
    "linkedServiceName": {
      "referenceName": "BlobStorageLinkedService",
      "type": "LinkedServiceReference"
    }
  }
}
```

Running the pipeline

To run the master pipeline in this example, the following values are passed for the masterSourceBlobContainer and masterSinkBlobContainer parameters:

```
{
  "masterSourceBlobContainer": "executetest",
  "masterSinkBlobContainer": "executesink"
}
```

The master pipeline forwards these values to the invoked pipeline as shown in the following example:

```
{
    "type": "ExecutePipeline",
    "typeProperties": {
      "pipeline": {
        "referenceName": "invokedPipeline",
        "type": "PipelineReference"
      },
      "parameters": {
        "sourceBlobContainer": {
          "value": "@pipeline().parameters.masterSourceBlobContainer",
          "type": "Expression"
        },
        "sinkBlobContainer": {
          "value": "@pipeline().parameters.masterSinkBlobContainer",
          "type": "Expression"
        }
      },

      ....
}
```

Next steps

See other supported control flow activities:

- For Each Activity
- Get Metadata Activity
- Lookup Activity
- Web Activity

Filter activity in Azure Data Factory and Synapse Analytics pipelines

8/19/2022 • 2 minutes to read • Edit Online

You can use a Filter activity in a pipeline to apply a filter expression to an input array.

APPLIES TO: Azure Data Factory Azure Synapse Analytics

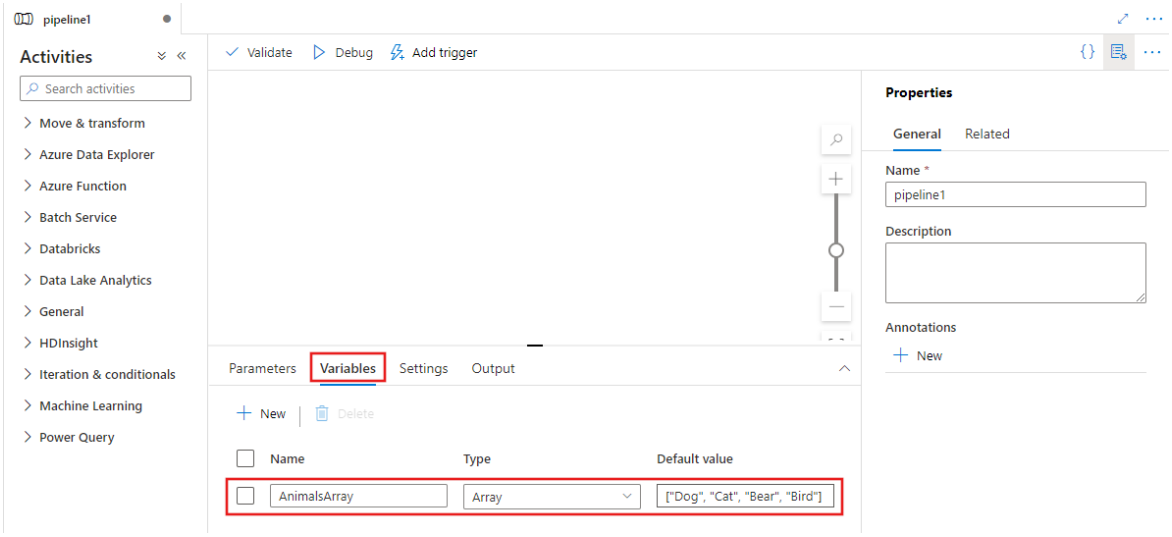
Syntax

```
{
    "name": "MyFilterActivity",
    "type": "filter",
    "typeProperties": {
        "condition": "<condition>",
        "items": "<input array>"
    }
}
```

Create a Filter activity with UI

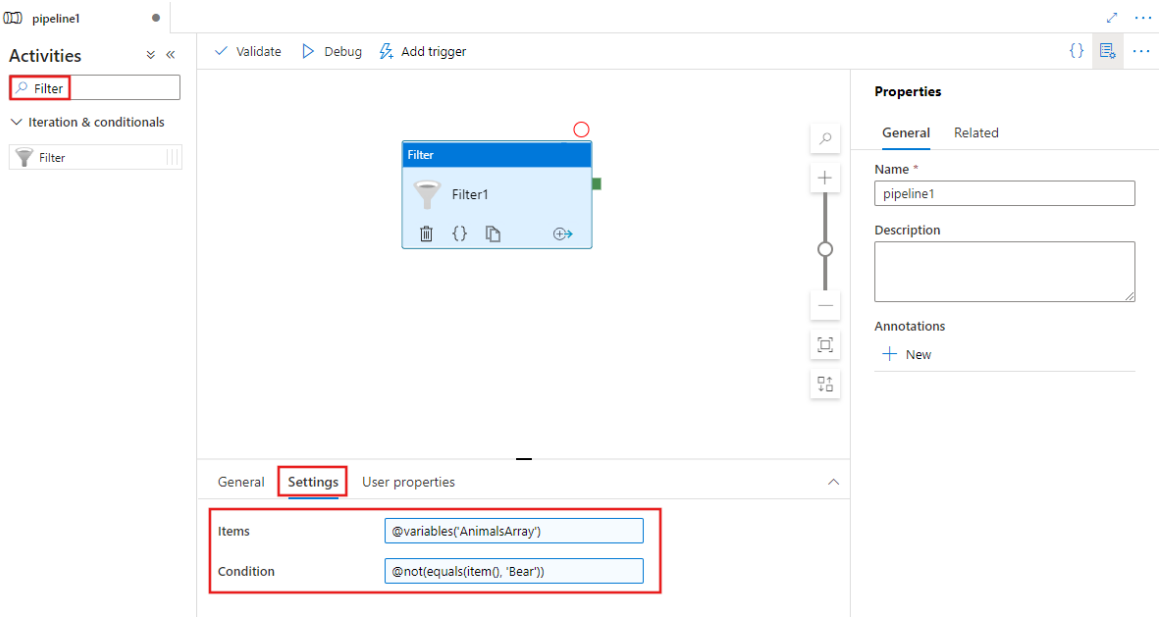
To use a Filter activity in a pipeline, complete the following steps:

1. You can use any array type variable or outputs from other activities as the input for your filter condition. To create an array variable, select the background of the pipeline canvas and then select the **Variables** tab to add an array type variable as shown below.

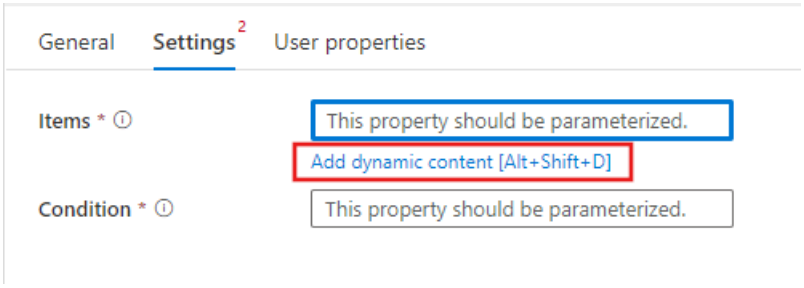


2. Search for *Filter* in the pipeline Activities pane, and drag a Filter activity to the pipeline canvas.

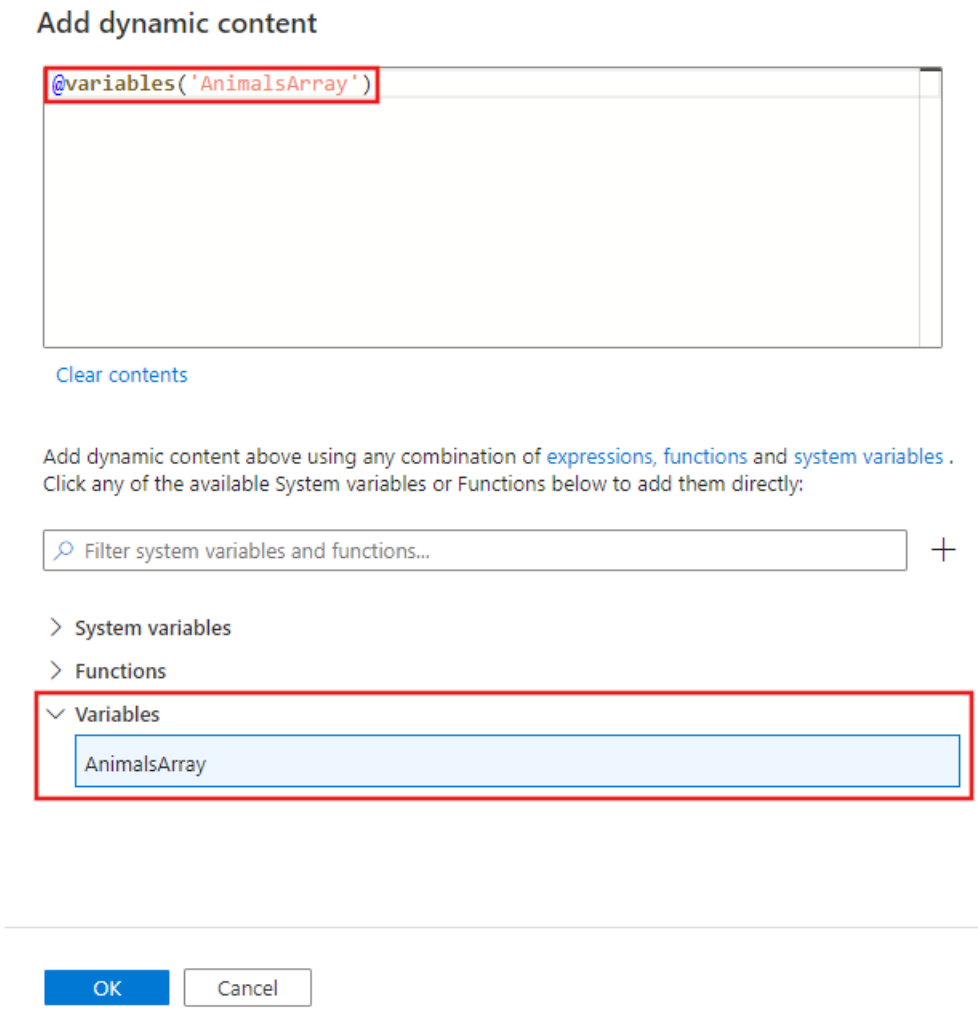
3. Select the new Filter activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



4. Select the **Items** field and then select the **Add dynamic content** link to open the dynamic content editor pane.



5. Select your input array to be filtered in the dynamic content editor. In this example, we select the variable created in the first step.



6. Use the dynamic content editor again to specify a filter condition for the Condition property, as shown above.

7. You can use the output from the Filter activity as an input to other activities like the ForEach activity.

Type properties

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
name	Name of the `Filter` activity.	String	Yes
type	Must be set to **filter**.	String	Yes
condition	Condition to be used for filtering the input.	Expression	Yes
items	Input array on which filter should be applied.	Expression	Yes

Example

In this example, the pipeline has two activities: **Filter** and **ForEach**. The Filter activity is configured to filter the input array for items with a value greater than 3. The ForEach activity then iterates over the filtered values and sets the variable **test** to the current value.

```
{
    "name": "PipelineName",
    "properties": {
        "activities": [{
                "name": "MyFilterActivity",
                "type": "filter",
                "typeProperties": {
                    "condition": "@greater(item(),3)",
                    "items": "@pipeline().parameters.inputs"
                }
            },
            {
            "name": "MyForEach",
            "type": "ForEach",
            "dependsOn": [
                {
                    "activity": "MyFilterActivity",
                    "dependencyConditions": [
                        "Succeeded"
                    ]
                }
            ],
            "userProperties": [],
            "typeProperties": {
                "items": {
                    "value": "@activity('MyFilterActivity').output.value",
                    "type": "Expression"
                },
                "isSequential": "false",
                "batchCount": 1,
                "activities": [
                    {
                        "name": "Set Variable1",
                        "type": "SetVariable",
                        "dependsOn": [],
                        "userProperties": [],
                        "typeProperties": {
                            "variableName": "test",
                            "value": {
                                "value": "@string(item())",
                                "type": "Expression"
                            }
                        }
                    }
                ]
            }
        }],
        "parameters": {
            "inputs": {
                "type": "Array",
                "defaultValue": [1, 2, 3, 4, 5, 6]
            }
        },
        "variables": {
            "test": {
                "type": "String"
            }
        },
        "annotations": []
    }
}
```

Next steps

See other supported control flow activities:

- If Condition Activity
- Execute Pipeline Activity
- For Each Activity
- Get Metadata Activity
- Lookup Activity
- Web Activity
- Until Activity

ForEach activity in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

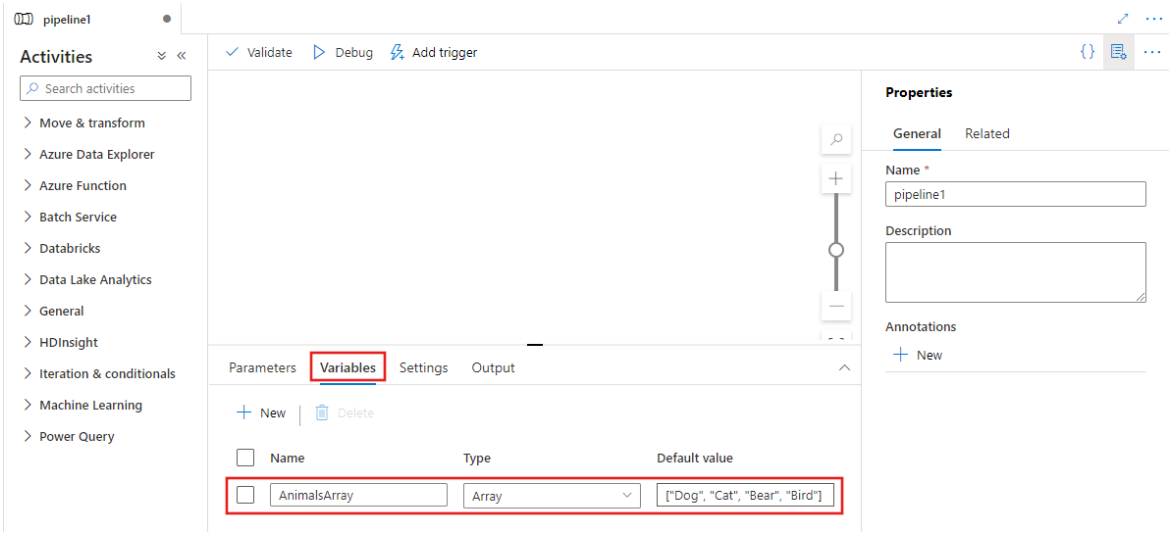
APPLIES TO: Azure Data Factory Azure Synapse Analytics

The ForEach Activity defines a repeating control flow in an Azure Data Factory or Synapse pipeline. This activity is used to iterate over a collection and executes specified activities in a loop. The loop implementation of this activity is similar to Foreach looping structure in programming languages.

Create a ForEach activity with UI

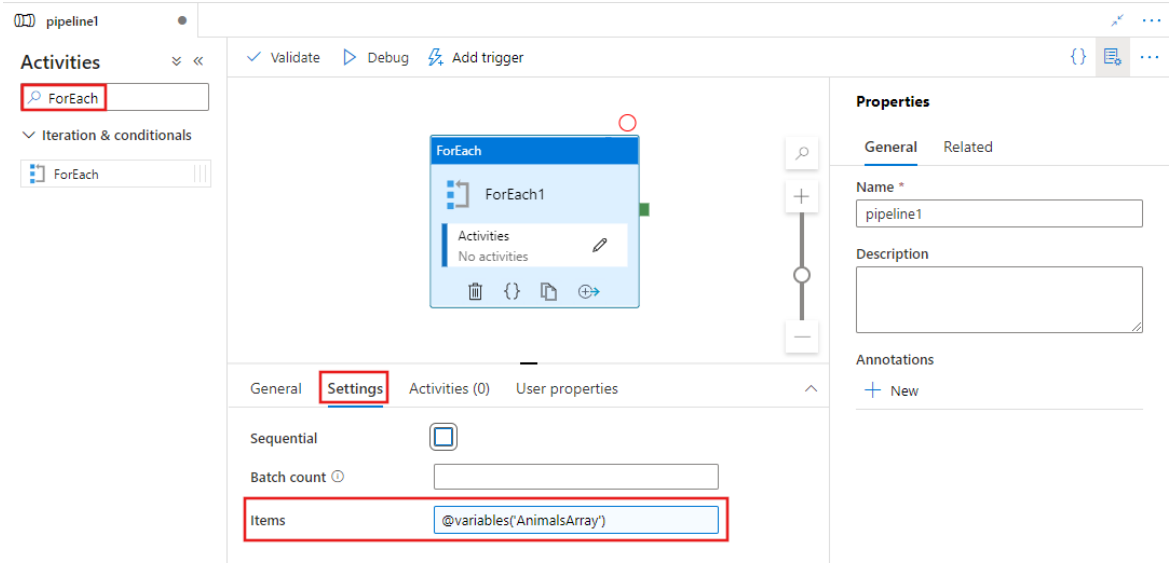
To use a ForEach activity in a pipeline, complete the following steps:

1. You can use any array type variable or outputs from other activities as the input for your ForEach activity. To create an array variable, select the background of the pipeline canvas and then select the **Variables** tab to add an array type variable as shown below.

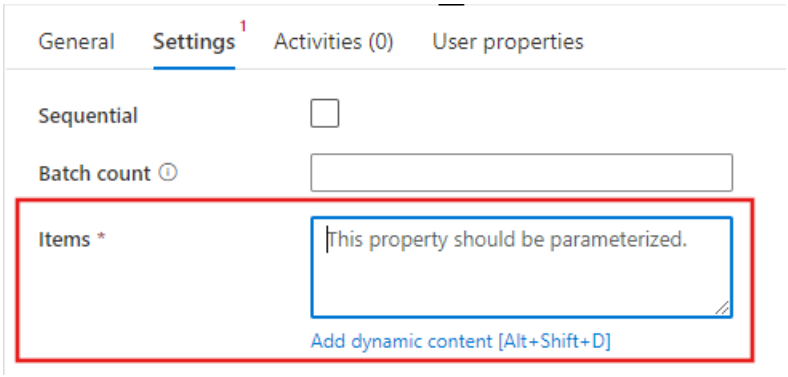


2. Search for *ForEach* in the pipeline Activities pane, and drag a ForEach activity to the pipeline canvas.

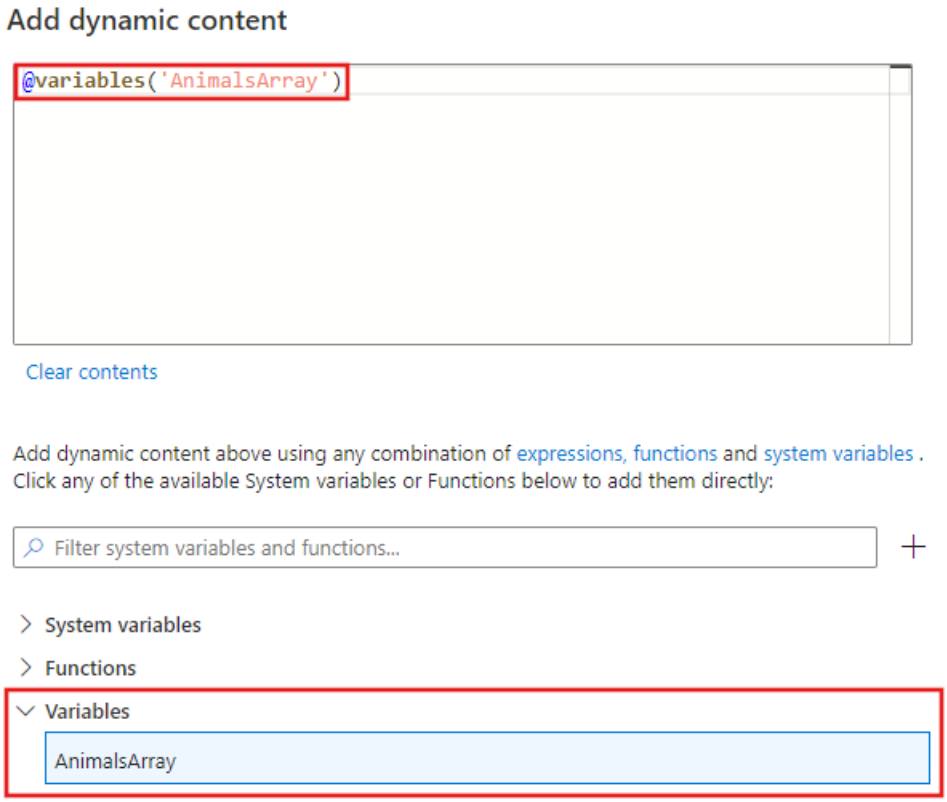
3. Select the new ForEach activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



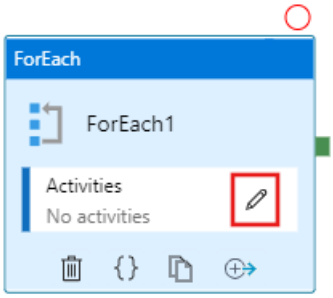
4. Select the **Items** field and then select the **Add dynamic content** link to open the dynamic content editor pane.



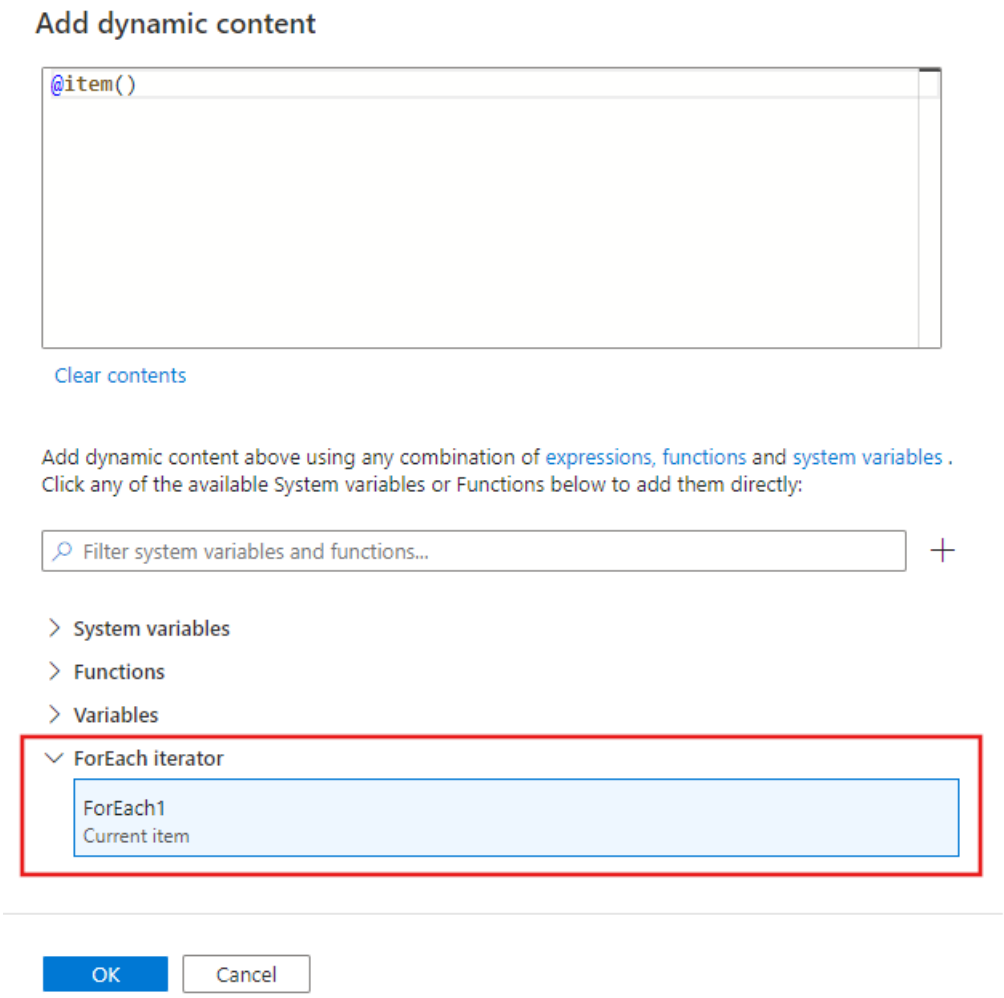
5. Select your input array to be filtered in the dynamic content editor. In this example, we select the variable created in the first step.



6. Select the Activities editor on the ForEach activity to add one or more activities to be executed for each item in the input **Items** array.



7. In any activities you create within the ForEach activity, you can reference the current item the ForEach activity is iterating through from the **Items** list. You can reference the current item anywhere you can use a dynamic expression to specify a property value. In the dynamic content editor, select the ForEach iterator to return the current item.



Syntax

The properties are described later in this article. The items property is the collection and each item in the collection is referred to by using the `@item()` as shown in the following syntax:

```
{
   "name":"MyForEachActivityName",
   "type":"ForEach",
   "typeProperties":{
      "isSequential":"true",
        "items": {
            "value": "@pipeline().parameters.mySinkDatasetFolderPathCollection",
            "type": "Expression"
        },
      "activities":[
         {
            "name":"MyCopyActivity",
            "type":"Copy",
            "typeProperties":{
               ...
            },
            "inputs":[
               {
                  "referenceName":"MyDataset",
                  "type":"DatasetReference",
                  "parameters":{
                     "MyFolderPath":"@pipeline().parameters.mySourceDatasetFolderPath"
                  }
               }
            ],
            "outputs":[
               {
                  "referenceName":"MyDataset",
                  "type":"DatasetReference",
                  "parameters":{
                     "MyFolderPath":"@item()"
                  }
               }
            ]
         }
      ]
   }
}
```

Type properties

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
name	Name of the for-each activity.	String	Yes
type	Must be set to **ForEach**	String	Yes

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
isSequential	Specifies whether the loop should be executed sequentially or in parallel. Maximum of 50 loop iterations can be executed at once in parallel). For example, if you have a ForEach activity iterating over a copy activity with 10 different source and sink datasets with **isSequential** set to False, all copies are executed at once. Default is False. If "isSequential" is set to False, ensure that there is a correct configuration to run multiple executables. Otherwise, this property should be used with caution to avoid incurring write conflicts. For more information, see Parallel execution section.	Boolean	No. Default is False.
batchCount	Batch count to be used for controlling the number of parallel execution (when isSequential is set to false). This is the upper concurrency limit, but the for-each activity will not always execute at this number	Integer (maximum 50)	No. Default is 20.
Items	An expression that returns a JSON Array to be iterated over.	Expression (which returns a JSON Array)	Yes
Activities	The activities to be executed.	List of Activities	Yes

Parallel execution

If **isSequential** is set to false, the activity iterates in parallel with a maximum of 50 concurrent iterations. This setting should be used with caution. If the concurrent iterations are writing to the same folder but to different files, this approach is fine. If the concurrent iterations are writing concurrently to the exact same file, this approach most likely causes an error.

Iteration expression language

In the ForEach activity, provide an array to be iterated over for the property **items**." Use `@item()` to iterate over a single enumeration in ForEach activity. For example, if **items** is an array: [1, 2, 3], `@item()` returns 1 in the first iteration, 2 in the second iteration, and 3 in the third iteration. You can also use `@range(0,10)` like expression to iterate ten times starting with 0 ending with 9.

Iterating over a single activity

Scenario: Copy from the same source file in Azure Blob to multiple destination files in Azure Blob.

Pipeline definition

```
{
    "name": "<MyForEachPipeline>",
    "properties": {
        "activities": [
            {
                "name": "<MyForEachActivity>",
                "type": "ForEach",
                "typeProperties": {
                    "isSequential": "true",
                    "items": {
                        "value": "@pipeline().parameters.mySinkDatasetFolderPath",
                        "type": "Expression"
                    },
                    "activities": [
                        {
                            "name": "MyCopyActivity",
                            "type": "Copy",
                            "typeProperties": {
                                "source": {
                                    "type": "BlobSource",
                                    "recursive": "false"
                                },
                                "sink": {
                                    "type": "BlobSink",
                                    "copyBehavior": "PreserveHierarchy"
                                }
                            },
                            "inputs": [
                                {
                                    "referenceName": "<MyDataset>",
                                    "type": "DatasetReference",
                                    "parameters": {
                                        "MyFolderPath": "@pipeline().parameters.mySourceDatasetFolderPath"
                                    }
                                }
                            ],
                            "outputs": [
                                {
                                    "referenceName": "MyDataset",
                                    "type": "DatasetReference",
                                    "parameters": {
                                        "MyFolderPath": "@item()"
                                    }
                                }
                            ]
                        }
                    ]
                }
            }
        ],
        "parameters": {
            "mySourceDatasetFolderPath": {
                "type": "String"
            },
            "mySinkDatasetFolderPath": {
                "type": "String"
            }
        }
    }
}
```

Blob dataset definition

```
{
   "name":"<MyDataset>",
   "properties":{
      "type":"AzureBlob",
      "typeProperties":{
         "folderPath":{
            "value":"@dataset().MyFolderPath",
            "type":"Expression"
         }
      },
      "linkedServiceName":{
         "referenceName":"StorageLinkedService",
         "type":"LinkedServiceReference"
      },
      "parameters":{
         "MyFolderPath":{
            "type":"String"
         }
      }
   }
}
```

Run parameter values

```
{
    "mySourceDatasetFolderPath": "input/",
    "mySinkDatasetFolderPath": [ "outputs/file1", "outputs/file2" ]
}
```

Iterate over multiple activities

It's possible to iterate over multiple activities (for example: copy and web activities) in a ForEach activity. In this scenario, we recommend that you abstract out multiple activities into a separate pipeline. Then, you can use the ExecutePipeline activity in the pipeline with ForEach activity to invoke the separate pipeline with multiple activities.

Syntax

```
{
  "name": "masterPipeline",
  "properties": {
    "activities": [
      {
        "type": "ForEach",
        "name": "<MyForEachMultipleActivities>"
        "typeProperties": {
          "isSequential": true,
          "items": {
            ...
          },
          "activities": [
            {
              "type": "ExecutePipeline",
              "name": "<MyInnerPipeline>"
              "typeProperties": {
                "pipeline": {
                  "referenceName": "<copyHttpPipeline>",
                  "type": "PipelineReference"
                },
                "parameters": {
                  ...
                },
                "waitOnCompletion": true
              }
            }
          ]
        }
      }
    ],
    "parameters": {
      ...
    }
  }
}
```

Example

Scenario: Iterate over an InnerPipeline within a ForEach activity with Execute Pipeline activity. The inner pipeline copies with schema definitions parameterized.

Master Pipeline definition

```
{
  "name": "masterPipeline",
  "properties": {
    "activities": [
      {
        "type": "ForEach",
        "name": "MyForEachActivity",
        "typeProperties": {
          "isSequential": true,
          "items": {
            "value": "@pipeline().parameters.inputtables",
            "type": "Expression"
          },
          "activities": [
            {
              "type": "ExecutePipeline",
              "typeProperties": {
                "pipeline": {
                  "referenceName": "InnerCopyPipeline",
                  "type": "PipelineReference"
                },
                "parameters": {
                  "sourceTableName": {
                    "value": "@item().SourceTable",
                    "type": "Expression"
                  },
                  "sourceTableStructure": {
                    "value": "@item().SourceTableStructure",
                    "type": "Expression"
                  },
                  "sinkTableName": {
                    "value": "@item().DestTable",
                    "type": "Expression"
                  },
                  "sinkTableStructure": {
                    "value": "@item().DestTableStructure",
                    "type": "Expression"
                  }
                },
                "waitOnCompletion": true
              },
              "name": "ExecuteCopyPipeline"
            }
          ]
        }
      }
    ],
    "parameters": {
      "inputtables": {
        "type": "Array"
      }
    }
  }
}
```

Inner pipeline definition

```
{
  "name": "InnerCopyPipeline",
  "properties": {
    "activities": [
      {
        "type": "Copy",
        "typeProperties": {
          "source": {
            "type": "SqlSource"
```

```
            "type": "SqlSource"
          }
        },
        "sink": {
          "type": "SqlSink"
        }
      },
      "name": "CopyActivity",
      "inputs": [
        {
          "referenceName": "sqlSourceDataset",
          "parameters": {
            "SqlTableName": {
              "value": "@pipeline().parameters.sourceTableName",
              "type": "Expression"
            },
            "SqlTableStructure": {
              "value": "@pipeline().parameters.sourceTableStructure",
              "type": "Expression"
            }
          },
          "type": "DatasetReference"
        }
      ],
      "outputs": [
        {
          "referenceName": "sqlSinkDataset",
          "parameters": {
            "SqlTableName": {
              "value": "@pipeline().parameters.sinkTableName",
              "type": "Expression"
            },
            "SqlTableStructure": {
              "value": "@pipeline().parameters.sinkTableStructure",
              "type": "Expression"
            }
          },
          "type": "DatasetReference"
        }
      ]
    }
  ],
  "parameters": {
    "sourceTableName": {
      "type": "String"
    },
    "sourceTableStructure": {
      "type": "String"
    },
    "sinkTableName": {
      "type": "String"
    },
    "sinkTableStructure": {
      "type": "String"
    }
  }
 }
}
```

Source dataset definition

```
{
  "name": "sqlSourceDataset",
  "properties": {
    "type": "SqlServerTable",
    "typeProperties": {
      "tableName": {
        "value": "@dataset().SqlTableName",
        "type": "Expression"
      }
    },
    "structure": {
      "value": "@dataset().SqlTableStructure",
      "type": "Expression"
    },
    "linkedServiceName": {
      "referenceName": "sqlserverLS",
      "type": "LinkedServiceReference"
    },
    "parameters": {
      "SqlTableName": {
        "type": "String"
      },
      "SqlTableStructure": {
        "type": "String"
      }
    }
  }
}
```

Sink dataset definition

```
{
  "name": "sqlSinkDataSet",
  "properties": {
    "type": "AzureSqlTable",
    "typeProperties": {
      "tableName": {
        "value": "@dataset().SqlTableName",
        "type": "Expression"
      }
    },
    "structure": {
      "value": "@dataset().SqlTableStructure",
      "type": "Expression"
    },
    "linkedServiceName": {
      "referenceName": "azureSqlLS",
      "type": "LinkedServiceReference"
    },
    "parameters": {
      "SqlTableName": {
        "type": "String"
      },
      "SqlTableStructure": {
        "type": "String"
      }
    }
  }
}
```

Master pipeline parameters

```
{
    "inputtables": [
        {
            "SourceTable": "department",
            "SourceTableStructure": [
              {
                "name": "departmentid",
                "type": "int"
              },
              {
                "name": "departmentname",
                "type": "string"
              }
            ],
            "DestTable": "department2",
            "DestTableStructure": [
              {
                "name": "departmentid",
                "type": "int"
              },
              {
                "name": "departmentname",
                "type": "string"
              }
            ]
        }
    ]

}
```

Aggregating outputs

To aggregate outputs of **foreach** activity, please utilize *Variables* and *Append Variable* activity.

First, declare an `array` *variable* in the pipeline. Then, invoke *Append Variable* activity inside each **foreach** loop. Subsequently, you can retrieve the aggregation from your array.

Limitations and workarounds

Here are some limitations of the ForEach activity and suggested workarounds.

LIMITATION	WORKAROUND
You can't nest a ForEach loop inside another ForEach loop (or an Until loop).	Design a two-level pipeline where the outer pipeline with the outer ForEach loop iterates over an inner pipeline with the nested loop.
The ForEach activity has a maximum `batchCount` of 50 for parallel processing, and a maximum of 100,000 items.	Design a two-level pipeline where the outer pipeline with the ForEach activity iterates over an inner pipeline.
SetVariable can't be used inside a ForEach activity that runs in parallel as the variables are global to the whole pipeline, they are not scoped to a ForEach or any other activity.	Consider using sequential ForEach or use Execute Pipeline inside ForEach (Variable/Parameter handled in child Pipeline).

Next steps

See other supported control flow activities:

- Execute Pipeline Activity
- Get Metadata Activity
- Lookup Activity
- Web Activity

Get Metadata activity in Azure Data Factory or Azure Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

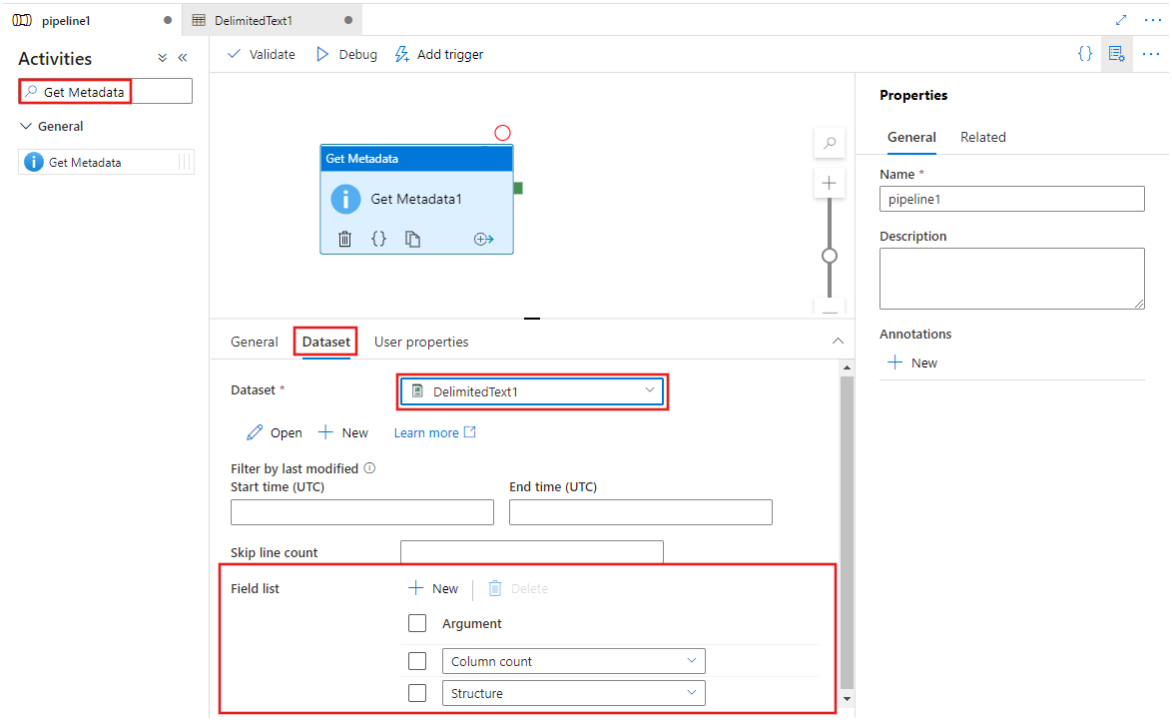
APPLIES TO: Azure Data Factory Azure Synapse Analytics

You can use the Get Metadata activity to retrieve the metadata of any data in Azure Data Factory or a Synapse pipeline. You can use the output from the Get Metadata activity in conditional expressions to perform validation, or consume the metadata in subsequent activities.

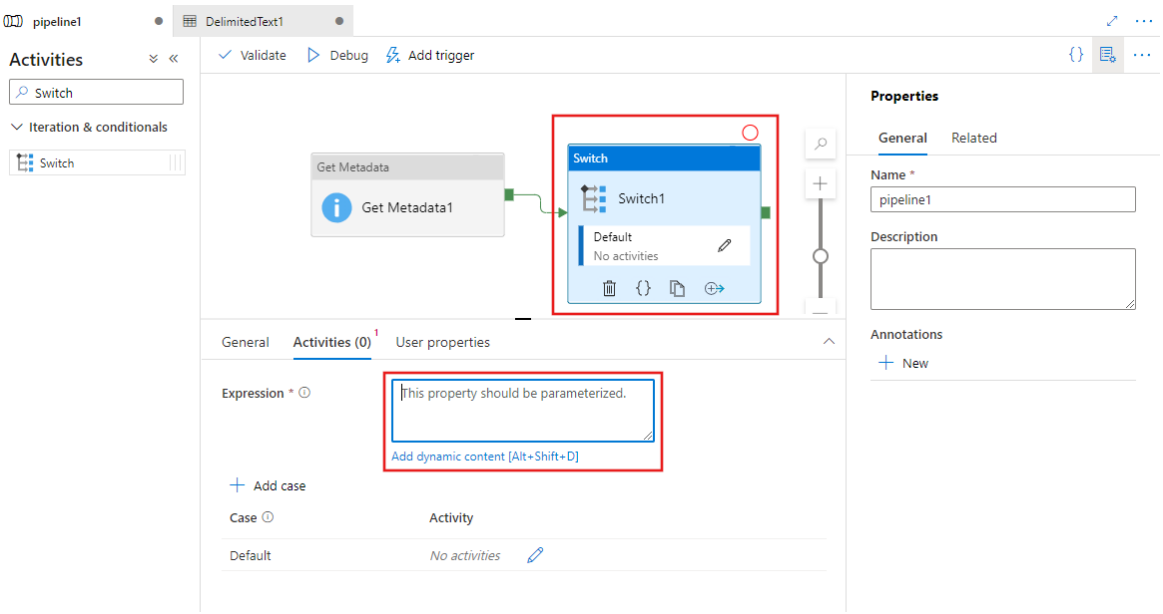
Create a Get Metadata activity with UI

To use a Get Metadata activity in a pipeline, complete the following steps:

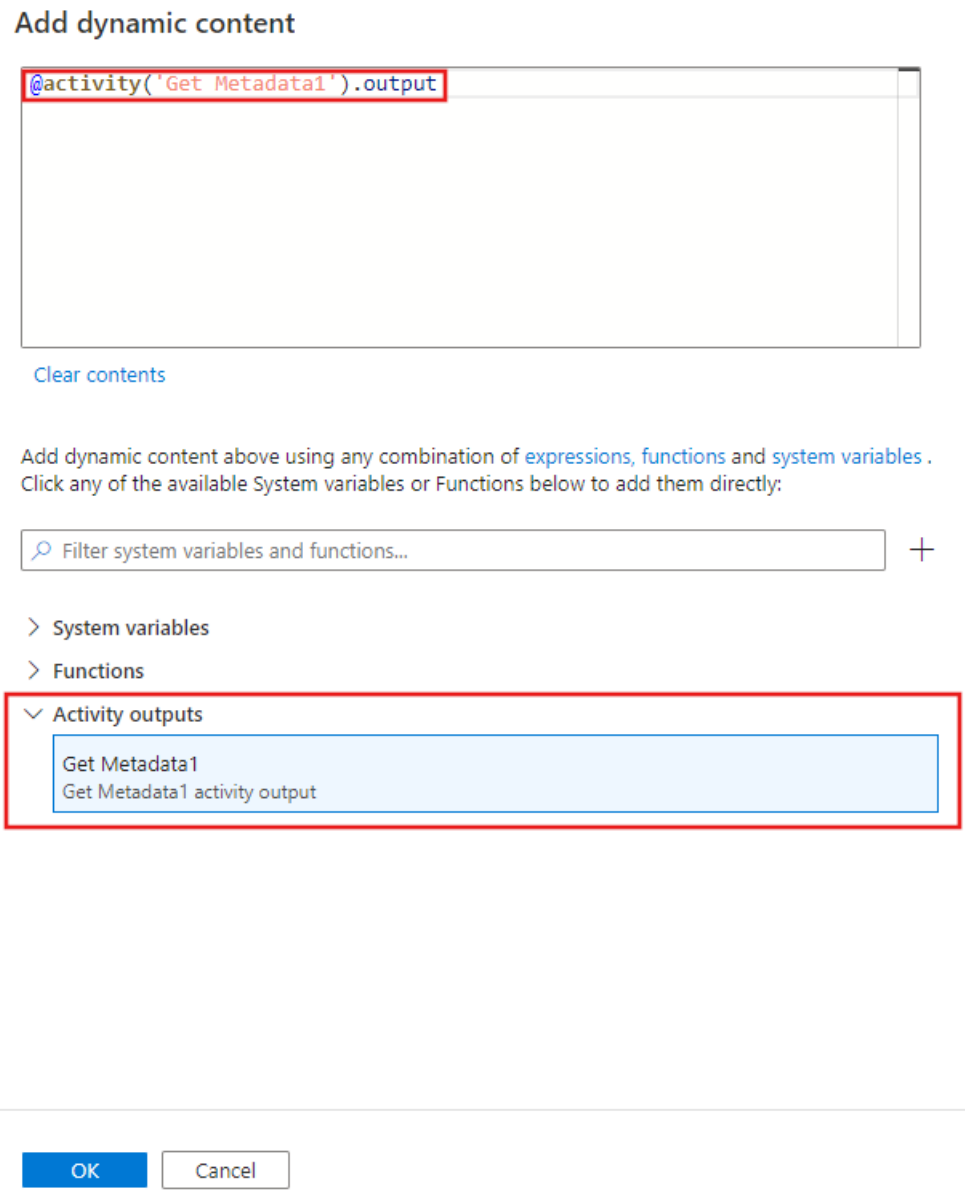
1. Search for *Get Metadata* in the pipeline Activities pane, and drag a Fail activity to the pipeline canvas.
2. Select the new Get Metadata activity on the canvas if it is not already selected, and its **Dataset** tab, to edit its details.
3. Choose a dataset, or create a new one with the New button. Then you can specify filter options and add columns from the available metadata for the dataset.



4. Use the output of the activity as an input to another activity, like a Switch activity in this example. You can reference the output of the Metadata Activity anywhere dynamic content is supported in the other activity.



5. In the dynamic content editor, select the Get Metadata activity output to reference it in the other activity.



Supported capabilities

The Get Metadata activity takes a dataset as an input and returns metadata information as output. Currently, the

following connectors and the corresponding retrievable metadata are supported. The maximum size of returned metadata is **4 MB**.

Supported connectors

File storage

CONNECTOR/METADATA	ITEMNAME (FILE/FOLDER)	ITEMTYPE (FILE/FOLDER)	SIZE (FILE)	CREATED (FILE/FOLDER)	LASTMODIFIED[1] (FILE/FOLDER)	CHILDITEMS (FOLDER)	CONTENTMD5 (FILE)	STRUCTURE[2] (FILE)	COLUMNCOUNT[2] (FILE)	EXISTS[3] (FILE/FOLDER)
Amazon S3	Ã/Ã	Ã/Ã	Ã	x/x	Ã/Ã	Ã	x	Ã	Ã	Ã/Ã
Amazon S3 Compatible Storage	Ã/Ã	Ã/Ã	Ã	x/x	Ã/Ã	Ã	x	Ã	Ã	Ã/Ã
Google Cloud Storage	Ã/Ã	Ã/Ã	Ã	x/x	Ã/Ã	Ã	x	Ã	Ã	Ã/Ã
Oracle Cloud Storage	Ã/Ã	Ã/Ã	Ã	x/x	Ã/Ã	Ã	x	Ã	Ã	Ã/Ã
Azure Blob storage	Ã/Ã	Ã/Ã	Ã	x/x	Ã/Ã	Ã	Ã	Ã	Ã	Ã/Ã
Azure Data Lake Storage Gen1	Ã/Ã	Ã/Ã	Ã	x/x	Ã/Ã	Ã	x	Ã	Ã	Ã/Ã
Azure Data Lake Storage Gen2	Ã/Ã	Ã/Ã	Ã	x/x	Ã/Ã	Ã	Ã	Ã	Ã	Ã/Ã
Azure Files	Ã/Ã	Ã/Ã	Ã	Ã/Ã	Ã/Ã	Ã	x	Ã	Ã	Ã/Ã
File system	Ã/Ã	Ã/Ã	Ã	Ã/Ã	Ã/Ã	Ã	x	Ã	Ã	Ã/Ã
SFTP	Ã/Ã	Ã/Ã	Ã	x/x	Ã/Ã	Ã	x	Ã	Ã	Ã/Ã
FTP	Ã/Ã	Ã/Ã	Ã	x/x	x/x	Ã	x	Ã	Ã	Ã/Ã

[1] Metadata `lastModified`:

- For Amazon S3, Amazon S3 Compatible Storage, Google Cloud Storage and Oracle Cloud Storage, `lastModified` applies to the bucket and the key but not to the virtual folder, and `exists` applies to the bucket and the key but not to the prefix or virtual folder.
- For Azure Blob storage, `lastModified` applies to the container and the blob but not to the virtual folder.

[2] Metadata `structure` and `columnCount` are not supported when getting metadata from Binary, JSON, or XML files.

[3] Metadata `exists`: For Amazon S3, Amazon S3 Compatible Storage, Google Cloud Storage and Oracle Cloud Storage, `exists` applies to the bucket and the key but not to the prefix or virtual folder.

Note the following:

- When using Get Metadata activity against a folder, make sure you have LIST/EXECUTE permission to the given folder.
- Wildcard filter on folders/files is not supported for Get Metadata activity.
- `modifiedDatetimeStart` and `modifiedDatetimeEnd` filter set on connector:
  - These two properties are used to filter the child items when getting metadata from a folder. It does not apply when getting metadata from a file.
  - When such filter is used, the `childItems` in output includes only the files that are modified within the specified range but not folders.
  - To apply such filter, GetMetadata activity will enumerate all the files in the specified folder and check the modified time. Avoid pointing to a folder with a large number of files even if the expected qualified file count is small.

Relational database

CONNECTOR/METADATA	STRUCTURE	COLUMNCOUNT	EXISTS
Amazon RDS for SQL Server	Ã	Ã	Ã
Azure SQL Database	Ã	Ã	Ã
Azure SQL Managed Instance	Ã	Ã	Ã
Azure Synapse Analytics	Ã	Ã	Ã
SQL Server	Ã	Ã	Ã

Metadata options

You can specify the following metadata types in the Get Metadata activity field list to retrieve the corresponding information:

METADATA TYPE	DESCRIPTION
itemName	Name of the file or folder.
itemType	Type of the file or folder. Returned value is `File` or `Folder`.

METADATA TYPE	DESCRIPTION
size	Size of the file, in bytes. Applicable only to files.
created	Created datetime of the file or folder.
lastModified	Last modified datetime of the file or folder.
childItems	List of subfolders and files in the given folder. Applicable only to folders. Returned value is a list of the name and type of each child item.
contentMD5	MD5 of the file. Applicable only to files.
structure	Data structure of the file or relational database table. Returned value is a list of column names and column types.
columnCount	Number of columns in the file or relational table.
exists	Whether a file, folder, or table exists. If `exists` is specified in the Get Metadata field list, the activity won't fail even if the file, folder, or table doesn't exist. Instead, `exists: false` is returned in the output.

TIP

When you want to validate that a file, folder, or table exists, specify `exists` in the Get Metadata activity field list. You can then check the `exists: true/false` result in the activity output. If `exists` isn't specified in the field list, the Get Metadata activity will fail if the object isn't found.

NOTE

When you get metadata from file stores and configure `modifiedDatetimeStart` or `modifiedDatetimeEnd`, the `childItems` in the output includes only files in the specified path that have a last modified time within the specified range. Items in subfolders are not included.

NOTE

For the **Structure** field list to provide the actual data structure for delimited text and Excel format datasets, you must enable the `First Row as Header` property, which is supported only for these data sources.

Syntax

Get Metadata activity

```
{
    "name":"MyActivity",
    "type":"GetMetadata",
    "dependsOn":[

    ],
    "policy":{
        "timeout":"7.00:00:00",
        "retry":0,
        "retryIntervalInSeconds":30,
        "secureOutput":false,
        "secureInput":false
    },
    "userProperties":[

    ],
    "typeProperties":{
        "dataset":{
            "referenceName":"MyDataset",
            "type":"DatasetReference"
        },
        "fieldList":[
            "size",
            "lastModified",
            "structure"
        ],
        "storeSettings":{
            "type":"AzureBlobStorageReadSettings"
        },
        "formatSettings":{
            "type":"JsonReadSettings"
        }
    }
}
```

Dataset

```
{
    "name":"MyDataset",
    "properties":{
        "linkedServiceName":{
            "referenceName":"AzureStorageLinkedService",
            "type":"LinkedServiceReference"
        },
        "annotations":[

        ],
        "type":"Json",
        "typeProperties":{
            "location":{
                "type":"AzureBlobStorageLocation",
                "fileName":"file.json",
                "folderPath":"folder",
                "container":"container"
            }
        }
    }
}
```

Type properties

Currently, the Get Metadata activity can return the following types of metadata information:

PROPERTY	DESCRIPTION	REQUIRED
fieldList	The types of metadata information required. For details on supported metadata, see the Metadata options section of this article.	Yes
dataset	The reference dataset whose metadata is to be retrieved by the Get Metadata activity. See the Capabilities section for information on supported connectors. Refer to the specific connector topics for dataset syntax details.	Yes
formatSettings	Apply when using format type dataset.	No
storeSettings	Apply when using format type dataset.	No

Sample output

The Get Metadata results are shown in the activity output. Following are two samples showing extensive metadata options. To use the results in a subsequent activity, use this pattern: `@{activity('MyGetMetadataActivity').output.itemName}`.

Get a file's metadata

```
{
  "exists": true,
  "itemName": "test.csv",
  "itemType": "File",
  "size": 104857600,
  "lastModified": "2017-02-23T06:17:09Z",
  "created": "2017-02-23T06:17:09Z",
  "contentMD5": "cMauY+Kz5zDm3eWa9VpoyQ==",
  "structure": [
    {
        "name": "id",
        "type": "Int64"
    },
    {
        "name": "name",
        "type": "String"
    }
  ],
  "columnCount": 2
}
```

Get a folder's metadata

```
{
  "exists": true,
  "itemName": "testFolder",
  "itemType": "Folder",
  "lastModified": "2017-02-23T06:17:09Z",
  "created": "2017-02-23T06:17:09Z",
  "childItems": [
    {
      "name": "test.avro",
      "type": "File"
    },
    {
      "name": "folder hello",
      "type": "Folder"
    }
  ]
}
```

Next steps

Learn about other supported control flow activities:

- Execute Pipeline activity
- ForEach activity
- Lookup activity
- Web activity

If Condition activity in Azure Data Factory and Synapse Analytics pipelines

8/19/2022 • 4 minutes to read • Edit Online

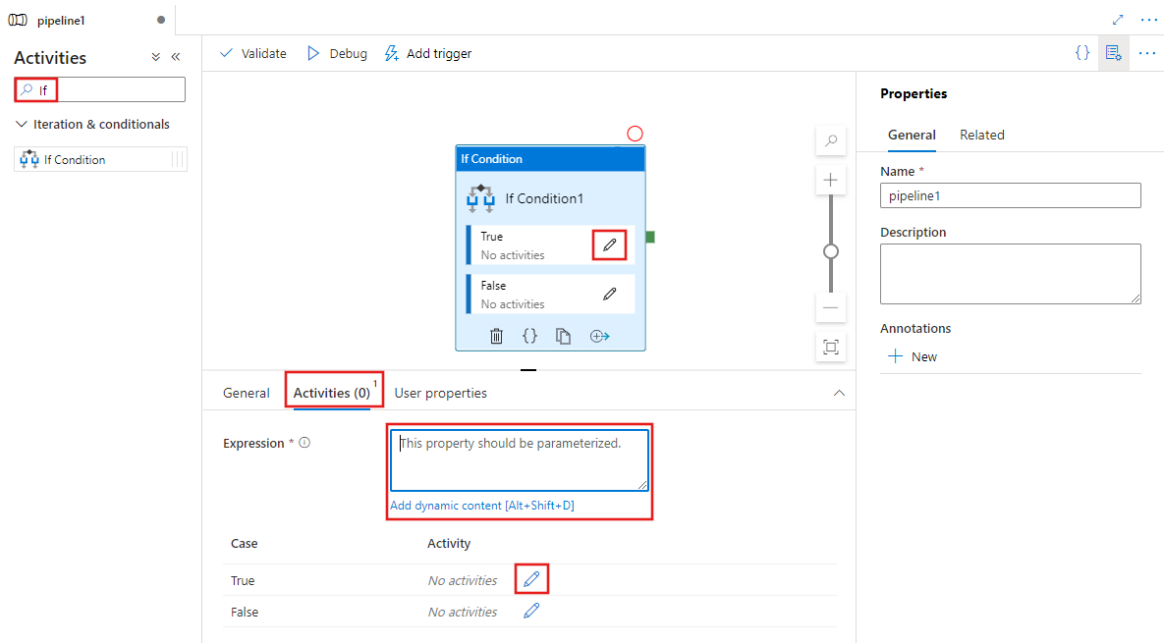
APPLIES TO: Azure Data Factory Azure Synapse Analytics

The If Condition activity provides the same functionality that an if statement provides in programming languages. It executes a set of activities when the condition evaluates to `true` and another set of activities when the condition evaluates to `false`.

Create an If Condition activity with UI

To use an If Condition activity in a pipeline, complete the following steps:

1. Search for *If* in the pipeline Activities pane, and drag an If Condition activity to the pipeline canvas.
2. Select the new If Condition activity on the canvas if it is not already selected, and its **Activities** tab, to edit its details.



3. Enter an expression that returns a boolean true or false value. This can be any combination of dynamic expressions, functions, system variables, or outputs from other activities.
4. Select the Edit Activities buttons on the Activities tab for the If Condition, or directly from the If Condition on the pipeline canvas, to add activities that will be executed when the expression evaluates to `true` or `false`.

Syntax

```
{
    "name": "<Name of the activity>",
    "type": "IfCondition",
    "typeProperties": {
      "expression": {
            "value": "<expression that evaluates to true or false>",
            "type": "Expression"
      },

      "ifTrueActivities": [
            {
                "<Activity 1 definition>"
            },
            {
                "<Activity 2 definition>"
            },
            {
                "<Activity N definition>"
            }
        ],

        "ifFalseActivities": [
            {
                "<Activity 1 definition>"
            },
            {
                "<Activity 2 definition>"
            },
            {
                "<Activity N definition>"
            }
      ]
    }
}
```

Type properties

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
name	Name of the if-condition activity.	String	Yes
type	Must be set to **IfCondition**	String	Yes
expression	Expression that must evaluate to true or false	Expression with result type boolean	Yes
ifTrueActivities	Set of activities that are executed when the expression evaluates to `true`.	Array	Yes
ifFalseActivities	Set of activities that are executed when the expression evaluates to `false`.	Array	Yes

Example

The pipeline in this example copies data from an input folder to an output folder. The output folder is determined by the value of pipeline parameter: routeSelection. If the value of routeSelection is true, the data is copied to outputPath1. And, if the value of routeSelection is false, the data is copied to outputPath2.

> **NOTE**
>
> This section provides JSON definitions and sample PowerShell commands to run the pipeline. For a walkthrough with step-by-step instructions to create a pipeline by using Azure PowerShell and JSON definitions, see tutorial: create a data factory by using Azure PowerShell.

Pipeline with IF-Condition activity (Adfv2QuickStartPipeline.json)

```
{
    "name": "Adfv2QuickStartPipeline",
    "properties": {
        "activities": [
            {
                "name": "MyIfCondition",
                "type": "IfCondition",
                "typeProperties": {
                    "expression":  {
                        "value":  "@bool(pipeline().parameters.routeSelection)",
                        "type": "Expression"
                     },

                    "ifTrueActivities": [
                        {
                            "name": "CopyFromBlobToBlob1",
                            "type": "Copy",
                            "inputs": [
                                {
                                    "referenceName": "BlobDataset",
                                    "parameters": {
                                        "path": "@pipeline().parameters.inputPath"
                                    },
                                    "type": "DatasetReference"
                                }
                            ],
                            "outputs": [
                                {
                                    "referenceName": "BlobDataset",
                                    "parameters": {
                                        "path": "@pipeline().parameters.outputPath1"
                                    },
                                    "type": "DatasetReference"
                                }
                            ],
                            "typeProperties": {
                                "source": {
                                    "type": "BlobSource"
                                },
                                "sink": {
                                    "type": "BlobSink"
                                }
                            }
                        }
                    ],
                    "ifFalseActivities": [
                        {
                            "name": "CopyFromBlobToBlob2",
                            "type": "Copy",
                            "inputs": [
                                {
                                    "referenceName": "BlobDataset",
                                    "parameters": {
```

```
                                    "path": "@pipeline().parameters.inputPath"
                                },
                                "type": "DatasetReference"
                            }
                        ],
                        "outputs": [
                            {
                                "referenceName": "BlobDataset",
                                "parameters": {
                                    "path": "@pipeline().parameters.outputPath2"
                                },
                                "type": "DatasetReference"
                            }
                        ],
                        "typeProperties": {
                            "source": {
                                "type": "BlobSource"
                            },
                            "sink": {
                                "type": "BlobSink"
                            }
                        }
                    }
                ]
            }
        }
    ],
    "parameters": {
        "inputPath": {
            "type": "String"
        },
        "outputPath1": {
            "type": "String"
        },
        "outputPath2": {
            "type": "String"
        },
        "routeSelection": {
            "type": "String"
        }
    }
}
```

Another example for expression is:

```
"expression":  {
    "value":  "@equals(pipeline().parameters.routeSelection,1)",
    "type": "Expression"
}
```

Azure Storage linked service (AzureStorageLinkedService.json)

```
{
    "name": "AzureStorageLinkedService",
    "properties": {
        "type": "AzureStorage",
        "typeProperties": {
            "connectionString": "DefaultEndpointsProtocol=https;AccountName=<Azure Storage account
name>;AccountKey=<Azure Storage account key>"
        }
    }
}
```

Parameterized Azure Blob dataset (BlobDataset.json)

The pipeline sets the **folderPath** to the value of either **outputPath1** or **outputPath2** parameter of the pipeline.

```
{
    "name": "BlobDataset",
    "properties": {
        "type": "AzureBlob",
        "typeProperties": {
            "folderPath": {
                "value": "@{dataset().path}",
                "type": "Expression"
            }
        },
        "linkedServiceName": {
            "referenceName": "AzureStorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "parameters": {
            "path": {
                "type": "String"
            }
        }
    }
}
```

Pipeline parameter JSON (PipelineParameters.json)

```
{
    "inputPath": "adftutorial/input",
    "outputPath1": "adftutorial/outputIf",
    "outputPath2": "adftutorial/outputElse",
    "routeSelection": "false"
}
```

PowerShell commands

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

These commands assume that you have saved the JSON files into the folder: C:\ADF.

```
Connect-AzAccount
Select-AzSubscription "<Your subscription name>"

$resourceGroupName = "<Resource Group Name>"
$dataFactoryName = "<Data Factory Name. Must be globally unique>";
Remove-AzDataFactoryV2 $dataFactoryName -ResourceGroupName $resourceGroupName -force


Set-AzDataFactoryV2 -ResourceGroupName $resourceGroupName -Location "East US" -Name $dataFactoryName
Set-AzDataFactoryV2LinkedService -DataFactoryName $dataFactoryName -ResourceGroupName $resourceGroupName -
Name "AzureStorageLinkedService" -DefinitionFile "C:\ADF\AzureStorageLinkedService.json"
Set-AzDataFactoryV2Dataset -DataFactoryName $dataFactoryName -ResourceGroupName $resourceGroupName -Name
"BlobDataset" -DefinitionFile "C:\ADF\BlobDataset.json"
Set-AzDataFactoryV2Pipeline -DataFactoryName $dataFactoryName -ResourceGroupName $resourceGroupName -Name
"Adfv2QuickStartPipeline" -DefinitionFile "C:\ADF\Adfv2QuickStartPipeline.json"
$runId = Invoke-AzDataFactoryV2Pipeline -DataFactoryName $dataFactoryName -ResourceGroupName
$resourceGroupName -PipelineName "Adfv2QuickStartPipeline" -ParameterFile C:\ADF\PipelineParameters.json
while ($True) {
    $run = Get-AzDataFactoryV2PipelineRun -ResourceGroupName $resourceGroupName -DataFactoryName
$DataFactoryName -PipelineRunId $runId

    if ($run) {
        if ($run.Status -ne 'InProgress') {
            Write-Host "Pipeline run finished. The status is: " $run.Status -foregroundcolor "Yellow"
            $run
            break
        }
        Write-Host  "Pipeline is running...status: InProgress" -foregroundcolor "Yellow"
    }

    Start-Sleep -Seconds 30
}
Write-Host "Activity run details:" -foregroundcolor "Yellow"
$result = Get-AzDataFactoryV2ActivityRun -DataFactoryName $dataFactoryName -ResourceGroupName
$resourceGroupName -PipelineRunId $runId -RunStartedAfter (Get-Date).AddMinutes(-30) -RunStartedBefore (Get-
Date).AddMinutes(30)
$result

Write-Host "Activity 'Output' section:" -foregroundcolor "Yellow"
$result.Output -join "`r`n"

Write-Host "\nActivity 'Error' section:" -foregroundcolor "Yellow"
$result.Error -join "`r`n"
```

Next steps

See other supported control flow activities:

- Execute Pipeline Activity
- For Each Activity
- Get Metadata Activity
- Lookup Activity
- Web Activity

Lookup activity in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 8 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

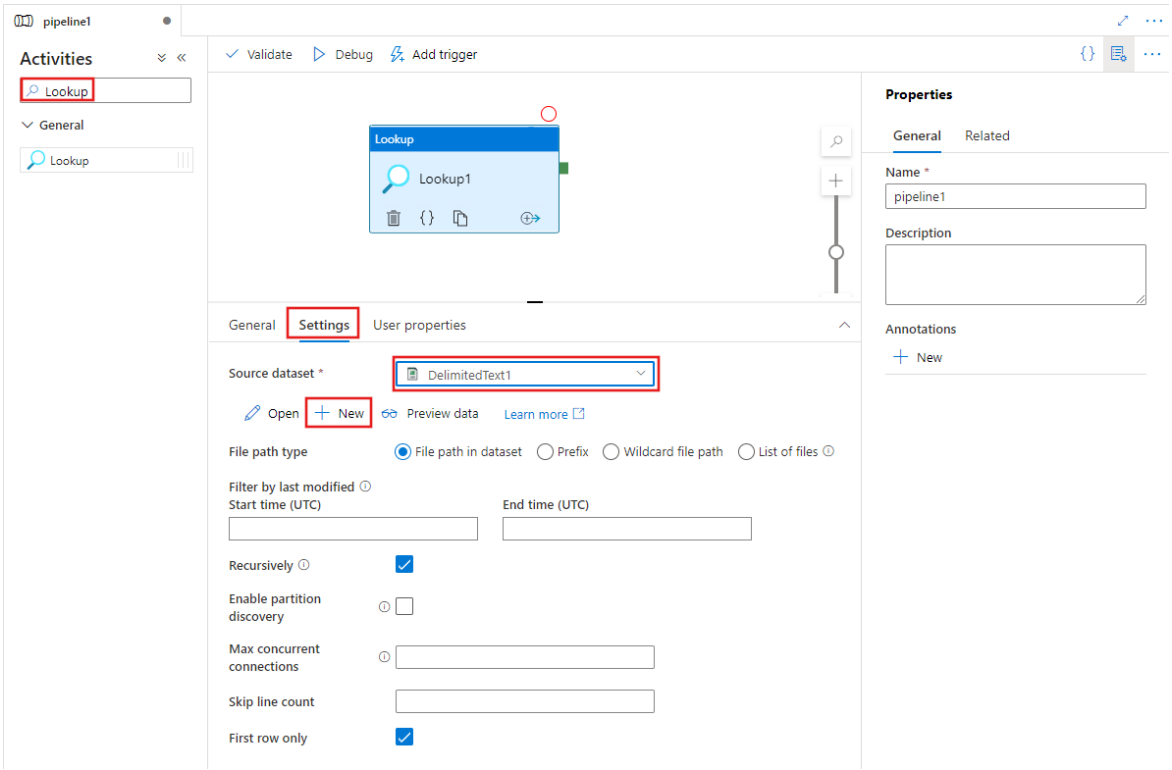
Lookup activity can retrieve a dataset from any of the data sources supported by data factory and Synapse pipelines. You can use it to dynamically determine which objects to operate on in a subsequent activity, instead of hard coding the object name. Some object examples are files and tables.

Lookup activity reads and returns the content of a configuration file or table. It also returns the result of executing a query or stored procedure. The output can be a singleton value or an array of attributes, which can be consumed in a subsequent copy, transformation, or control flow activities like ForEach activity.

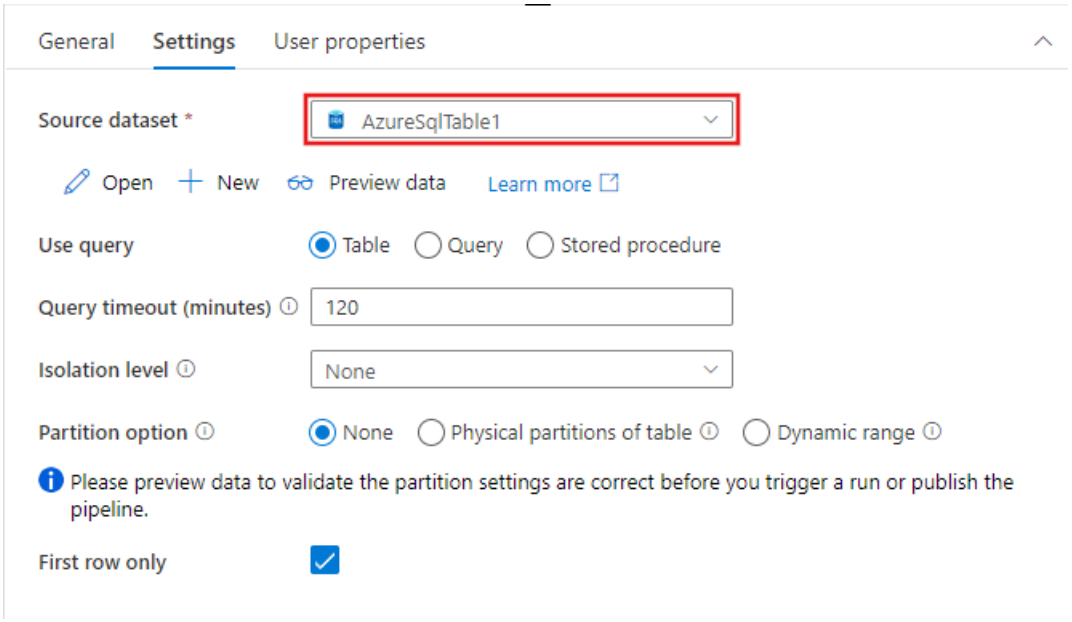
Create a Lookup activity with UI

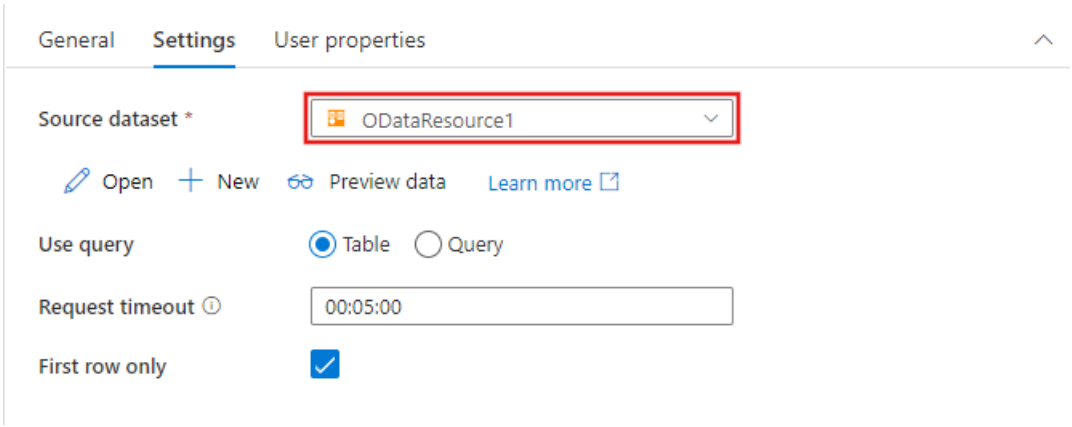
To use a Lookup activity in a pipeline, complete the following steps:

1. Search for *Lookup* in the pipeline Activities pane, and drag a Lookup activity to the pipeline canvas.
2. Select the new Lookup activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



3. Choose an existing source dataset or select the **New** button to create a new one.
4. The options for identifying rows to include from the source dataset will vary based on the dataset type. The example above shows the configuration options for a delimited text dataset. Below are examples of configuration options for an Azure SQL table dataset and an OData dataset.





Supported capabilities

Note the following:

- The Lookup activity can return up to **5000 rows**; if the result set contains more records, the first 5000 rows will be returned.
- The Lookup activity output supports up to **4 MB** in size, activity will fail if the size exceeds the limit.
- The longest duration for Lookup activity before timeout is **24 hours**.

> **NOTE**
> When you use query or stored procedure to lookup data, make sure to return one and exact one result set. Otherwise, Lookup activity fails.

The following data sources are supported for Lookup activity.

CATEGORY	DATA STORE
Azure	Azure Blob storage
	Azure Cosmos DB (SQL API)
	Azure Data Explorer (Kusto)

CATEGORY	DATA STORE
	Azure Data Lake Storage Gen1
	Azure Data Lake Storage Gen2
	Azure Database for MariaDB
	Azure Database for MySQL
	Azure Database for PostgreSQL
	Azure Databricks Delta Lake
	Azure Files
	Azure SQL Database
	Azure SQL Database Managed Instance
	Azure Synapse Analytics
	Azure Table storage
Database	Amazon RDS for Oracle
	Amazon RDS for SQL Server
	Amazon Redshift
	Apache Impala
	Azure SQL Database Managed Instance
	DB2
	Drill
	Google AdWords
	Google BigQuery
	Greenplum
	HBase
	Hive
	Informix
	MariaDB

CATEGORY	DATA STORE
	Microsoft Access
	MySQL
	Netezza
	Oracle
	Phoenix
	PostgreSQL
	Presto
	SAP Business Warehouse Open Hub
	SAP Business Warehouse via MDX
	SAP HANA
	SAP Table
	SQL Server
	Spark
	Sybase
	Teradata
	Vertica
NoSQL	Cassandra
	Couchbase (Preview)
File	Amazon S3
	Amazon S3 Compatible Storage
	File System
	FTP
	Google Cloud Storage
	HDFS
	Generic HTTP

CATEGORY	DATA STORE
	Oracle Cloud Storage
	SFTP
Generic protocol	Generic OData
	Generic ODBC
	SharePoint Online List
Services and apps	Amazon Marketplace Web Service
	Concur (Preview)
	Dataverse
	Dynamics 365
	Dynamics AX
	Dynamics CRM
	HubSpot
	Jira
	Azure Data Explorer (Kusto)
	Magento (Preview)
	Marketo (Preview)
	Office 365
	Oracle Eloqua (Preview)
	Oracle Responsys (Preview)
	Oracle Service Cloud (Preview)
	PayPal (Preview)
	QuickBooks (Preview)
	SAP Cloud for Customer (C4C)
	SAP ECC
	Salesforce

CATEGORY	DATA STORE
	Salesforce Marketing Cloud
	Salesforce Service Cloud
	ServiceNow
	Shopify (Preview)
	Snowflake
	Square (Preview)
	Web Table (HTML table)
	Xero
	Zoho (Preview)

NOTE

Any connector marked as *Preview* means that you can try it out and give us feedback. If you want to take a dependency on preview connectors in your solution, please contact Azure support.

Syntax

```
{
    "name":"LookupActivity",
    "type":"Lookup",
    "typeProperties":{
        "source":{
            "type":"<source type>"
        },
        "dataset":{
            "referenceName":"<source dataset name>",
            "type":"DatasetReference"
        },
        "firstRowOnly":<true or false>
    }
}
```

Type properties

NAME	DESCRIPTION	TYPE	REQUIRED?
dataset	Provides the dataset reference for the lookup. Get details from the **Dataset properties** section in each corresponding connector article.	Key/value pair	Yes

NAME	DESCRIPTION	TYPE	REQUIRED?
source	Contains dataset-specific source properties, the same as the Copy Activity source. Get details from the **Copy Activity properties** section in each corresponding connector article.	Key/value pair	Yes
firstRowOnly	Indicates whether to return only the first row or all rows.	Boolean	No. The default is `true`.

NOTE

- Source columns with **ByteArray** type aren't supported.
- **Structure** isn't supported in dataset definitions. For text-format files, use the header row to provide the column name.
- If your lookup source is a JSON file, the `jsonPathDefinition` setting for reshaping the JSON object isn't supported. The entire objects will be retrieved.

Use the Lookup activity result

The lookup result is returned in the `output` section of the activity run result.

- **When `firstRowOnly` is set to `true` (default)**, the output format is as shown in the following code. The lookup result is under a fixed `firstRow` key. To use the result in subsequent activity, use the pattern of `@{activity('LookupActivity').output.firstRow.table}`.

```
{
    "firstRow":
    {
        "Id": "1",
        "schema":"dbo",
        "table":"Table1"
    }
}
```

- **When `firstRowOnly` is set to `false`**, the output format is as shown in the following code. A `count` field indicates how many records are returned. Detailed values are displayed under a fixed `value` array. In such a case, the Lookup activity is followed by a Foreach activity. You pass the `value` array to the ForEach activity `items` field by using the pattern of `@activity('MyLookupActivity').output.value`. To access elements in the `value` array, use the following syntax: `@{activity('lookupActivity').output.value[zero based index].propertyname}`. An example is `@{activity('lookupActivity').output.value[0].schema}`.

```
{
    "count": "2",
    "value": [
        {
            "Id": "1",
            "schema":"dbo",
            "table":"Table1"
        },
        {
            "Id": "2",
            "schema":"dbo",
            "table":"Table2"
        }
    ]
}
```

Example

In this example, the pipeline contains two activities: **Lookup** and **Copy**. The Copy Activity copies data from a SQL table in your Azure SQL Database instance to Azure Blob storage. The name of the SQL table is stored in a JSON file in Blob storage. The Lookup activity looks up the table name at runtime. JSON is modified dynamically by using this approach. You don't need to redeploy pipelines or datasets.

This example demonstrates lookup for the first row only. For lookup for all rows and to chain the results with ForEach activity, see the samples in Copy multiple tables in bulk.

Pipeline

- The Lookup activity is configured to use **LookupDataset**, which refers to a location in Azure Blob storage. The Lookup activity reads the name of the SQL table from a JSON file in this location.
- The Copy Activity uses the output of the Lookup activity, which is the name of the SQL table. The **tableName** property in the **SourceDataset** is configured to use the output from the Lookup activity. Copy Activity copies data from the SQL table to a location in Azure Blob storage. The location is specified by the **SinkDataset** property.

```
{
    "name": "LookupPipelineDemo",
    "properties": {
        "activities": [
            {
                "name": "LookupActivity",
                "type": "Lookup",
                "dependsOn": [],
                "policy": {
                    "timeout": "7.00:00:00",
                    "retry": 0,
                    "retryIntervalInSeconds": 30,
                    "secureOutput": false,
                    "secureInput": false
                },
                "userProperties": [],
                "typeProperties": {
                    "source": {
                        "type": "JsonSource",
                        "storeSettings": {
                            "type": "AzureBlobStorageReadSettings",
                            "recursive": true
                        },
                        "formatSettings": {
                            "type": "JsonReadSettings"
                        }
                    },
```

```
                    "dataset": {
                        "referenceName": "LookupDataset",
                        "type": "DatasetReference"
                    },
                    "firstRowOnly": true
                }
            },
            {
                "name": "CopyActivity",
                "type": "Copy",
                "dependsOn": [
                    {
                        "activity": "LookupActivity",
                        "dependencyConditions": [
                            "Succeeded"
                        ]
                    }
                ],
                "policy": {
                    "timeout": "7.00:00:00",
                    "retry": 0,
                    "retryIntervalInSeconds": 30,
                    "secureOutput": false,
                    "secureInput": false
                },
                "userProperties": [],
                "typeProperties": {
                    "source": {
                        "type": "AzureSqlSource",
                        "sqlReaderQuery": {
                            "value": "select * from [@{activity('LookupActivity').output.firstRow.schema}].
[@{activity('LookupActivity').output.firstRow.table}]",
                            "type": "Expression"
                        },
                        "queryTimeout": "02:00:00",
                        "partitionOption": "None"
                    },
                    "sink": {
                        "type": "DelimitedTextSink",
                        "storeSettings": {
                            "type": "AzureBlobStorageWriteSettings"
                        },
                        "formatSettings": {
                            "type": "DelimitedTextWriteSettings",
                            "quoteAllText": true,
                            "fileExtension": ".txt"
                        }
                    },
                    "enableStaging": false,
                    "translator": {
                        "type": "TabularTranslator",
                        "typeConversion": true,
                        "typeConversionSettings": {
                            "allowDataTruncation": true,
                            "treatBooleanAsNumber": false
                        }
                    }
                },
                "inputs": [
                    {
                        "referenceName": "SourceDataset",
                        "type": "DatasetReference",
                        "parameters": {
                            "schemaName": {
                                "value": "@activity('LookupActivity').output.firstRow.schema",
                                "type": "Expression"
                            },
                            "tableName": {
                                "value": "@activity('LookupActivity').output.firstRow.table",
```

```
                            "type": "Expression"
                        }
                    }
                }
            ],
            "outputs": [
                {
                    "referenceName": "SinkDataset",
                    "type": "DatasetReference",
                    "parameters": {
                        "schema": {
                            "value": "@activity('LookupActivity').output.firstRow.schema",
                            "type": "Expression"
                        },
                        "table": {
                            "value": "@activity('LookupActivity').output.firstRow.table",
                            "type": "Expression"
                        }
                    }
                }
            ]
        }
    ],
    "annotations": [],
    "lastPublishTime": "2020-08-17T10:48:25Z"
  }
}
```

Lookup dataset

The **lookup** dataset is the **sourcetable.json** file in the Azure Storage lookup folder specified by the **AzureBlobStorageLinkedService** type.

```
{
    "name": "LookupDataset",
    "properties": {
        "linkedServiceName": {
            "referenceName": "AzureBlobStorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "annotations": [],
        "type": "Json",
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "fileName": "sourcetable.json",
                "container": "lookup"
            }
        }
    }
}
```

Source dataset for Copy Activity

The **source** dataset uses the output of the Lookup activity, which is the name of the SQL table. Copy Activity copies data from this SQL table to a location in Azure Blob storage. The location is specified by the **sink** dataset.

```
{
    "name": "SourceDataset",
    "properties": {
        "linkedServiceName": {
            "referenceName": "AzureSqlDatabase",
            "type": "LinkedServiceReference"
        },
        "parameters": {
            "schemaName": {
                "type": "string"
            },
            "tableName": {
                "type": "string"
            }
        },
        "annotations": [],
        "type": "AzureSqlTable",
        "schema": [],
        "typeProperties": {
            "schema": {
                "value": "@dataset().schemaName",
                "type": "Expression"
            },
            "table": {
                "value": "@dataset().tableName",
                "type": "Expression"
            }
        }
    }
}
```

Sink dataset for Copy Activity

Copy Activity copies data from the SQL table to the **filebylookup.csv** file in the **csv** folder in Azure Storage. The file is specified by the **AzureBlobStorageLinkedService** property.

```
{
    "name": "SinkDataset",
    "properties": {
        "linkedServiceName": {
            "referenceName": "AzureBlobStorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "parameters": {
            "schema": {
                "type": "string"
            },
            "table": {
                "type": "string"
            }
        },
        "annotations": [],
        "type": "DelimitedText",
        "typeProperties": {
            "location": {
                "type": "AzureBlobStorageLocation",
                "fileName": {
                    "value": "@{dataset().schema}_@{dataset().table}.csv",
                    "type": "Expression"
                },
                "container": "csv"
            },
            "columnDelimiter": ",",
            "escapeChar": "\\",
            "quoteChar": "\""
        },
        "schema": []
    }
}
```

sourcetable.json

You can use following two kinds of formats for **sourcetable.json** file.

Set of objects

```
{
   "Id":"1",
   "schema":"dbo",
   "table":"Table1"
}
{
   "Id":"2",
   "schema":"dbo",
   "table":"Table2"
}
```

Array of objects

```
[
    {
        "Id": "1",
        "schema":"dbo",
        "table":"Table1"
    },
    {
        "Id": "2",
        "schema":"dbo",
        "table":"Table2"
    }
]
```

Limitations and workarounds

Here are some limitations of the Lookup activity and suggested workarounds.

LIMITATION	WORKAROUND
The Lookup activity has a maximum of 5,000 rows, and a maximum size of 4 MB.	Design a two-level pipeline where the outer pipeline iterates over an inner pipeline, which retrieves data that doesn't exceed the maximum rows or size.

Next steps

See other control flow activities supported by Azure Data Factory and Synapse pipelines:

- Execute Pipeline activity
- ForEach activity
- GetMetadata activity
- Web activity

Set Variable Activity in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

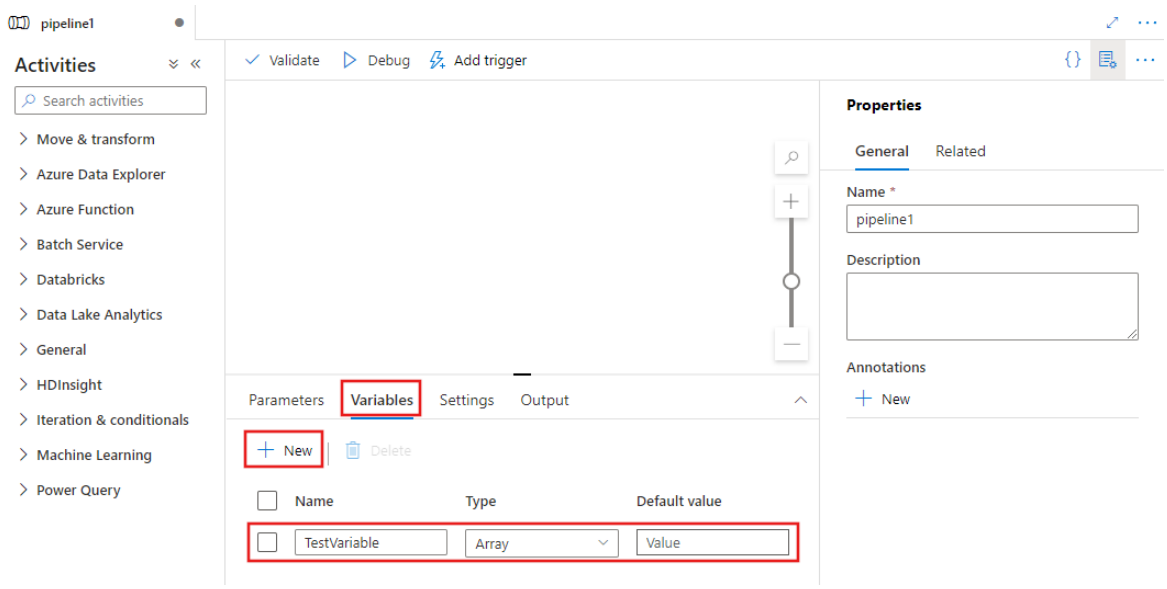
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Use the Set Variable activity to set the value of an existing variable of type String, Bool, or Array defined in a Data Factory or Synapse pipeline.

Create an Append Variable activity with UI

To use a Set Variable activity in a pipeline, complete the following steps:

1. Select the background of the pipeline canvas and use the Variables tab to add a variable:

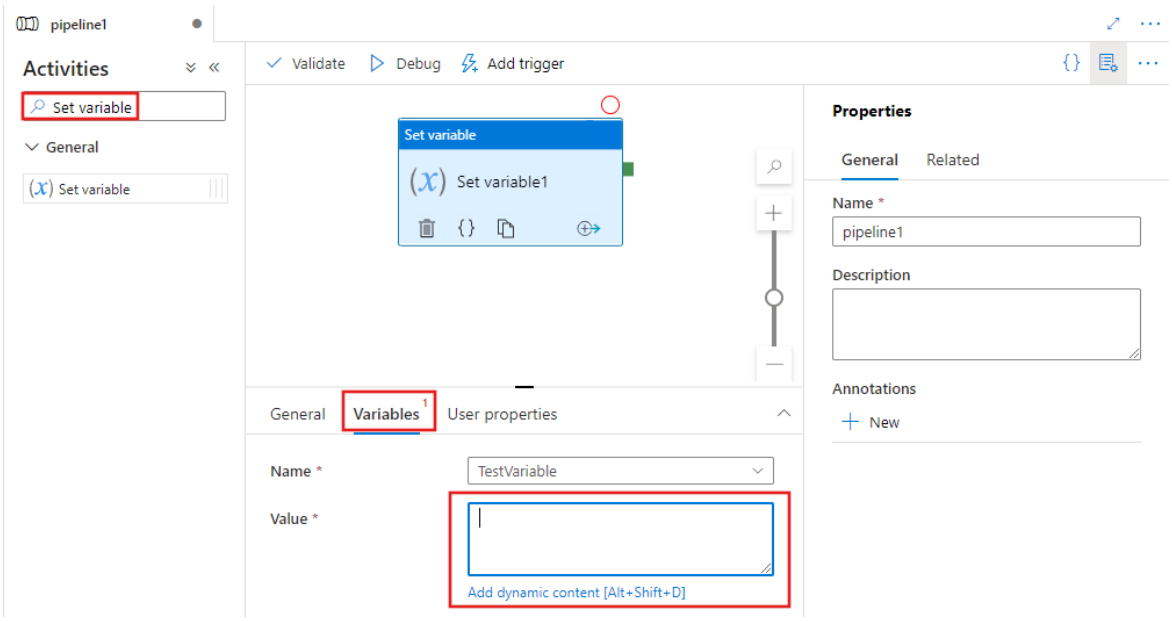


2. Search for *Set Variable* in the pipeline Activities pane, and drag a Set Variable activity to the pipeline canvas.

3. Select the Set Variable activity on the canvas if it is not already selected, and its **Variables** tab, to edit its details.

4. Select the variable for the Name property.

5. Enter an expression to set the value. This can be a literal string expression, or any combination of dynamic expressions, functions, system variables, or outputs from other activities.



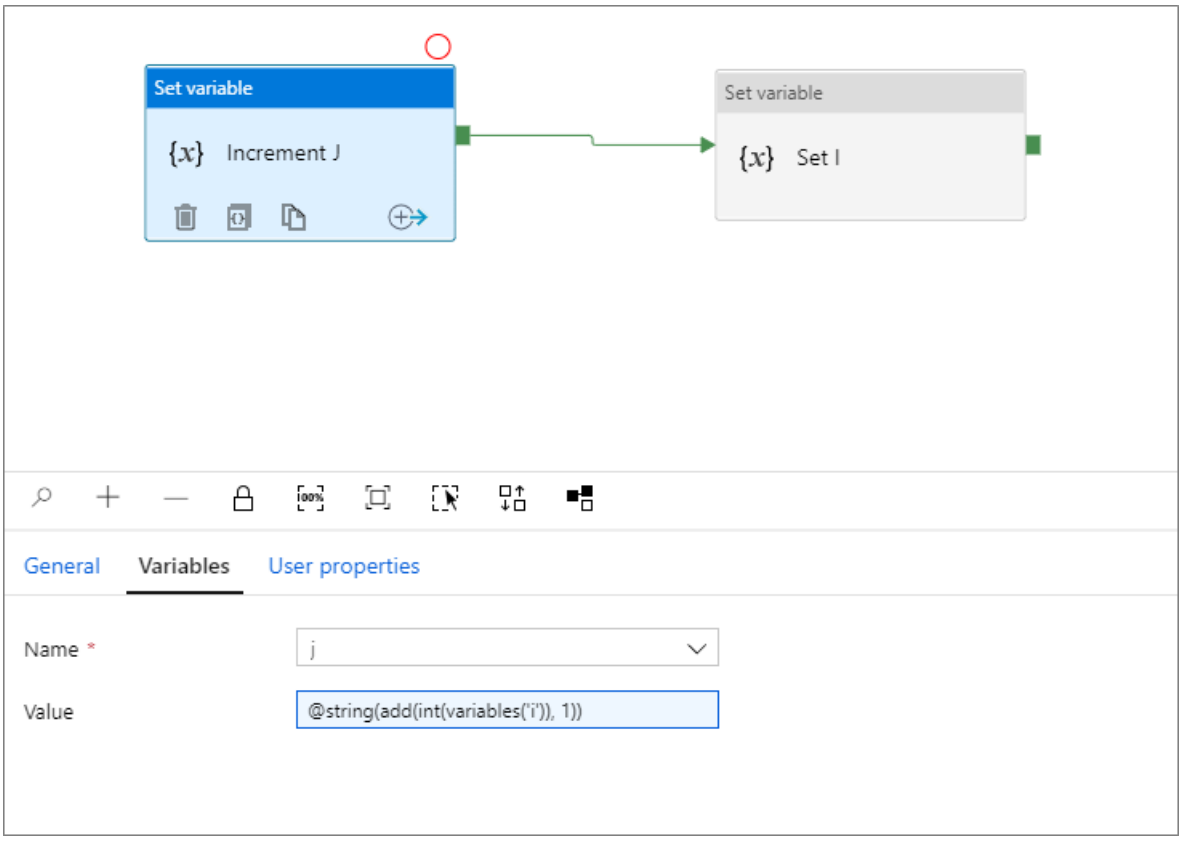
Type properties

PROPERTY	DESCRIPTION	REQUIRED
name	Name of the activity in pipeline	yes
description	Text describing what the activity does	no
type	Must be set to **SetVariable**	yes
value	String literal or expression object value that the variable is assigned to	yes
variableName	Name of the variable that is set by this activity	yes

Incrementing a variable

A common scenario involving variables is using a variable as an iterator within an until or foreach activity. In a set variable activity you cannot reference the variable being set in the `value` field. To workaround this limitation, set a temporary variable and then create a second set variable activity. The second set variable activity sets the value of the iterator to the temporary variable.

Below is an example of this pattern:


Set variable
{x} Increment J
Set variable
{x} Set I
General
Variables
User properties
Name *
j
Value
@string(add(int(variables('i')), 1))

```
{
    "name": "pipeline3",
    "properties": {
        "activities": [
            {
                "name": "Set I",
                "type": "SetVariable",
                "dependsOn": [
                    {
                        "activity": "Increment J",
                        "dependencyConditions": [
                            "Succeeded"
                        ]
                    }
                ],
                "userProperties": [],
                "typeProperties": {
                    "variableName": "i",
                    "value": {
                        "value": "@variables('j')",
                        "type": "Expression"
                    }
                }
            },
            {
                "name": "Increment J",
                "type": "SetVariable",
                "dependsOn": [],
                "userProperties": [],
                "typeProperties": {
                    "variableName": "j",
                    "value": {
                        "value": "@string(add(int(variables('i')), 1))",
                        "type": "Expression"
                    }
                }
            }
        ],
        "variables": {
            "i": {
                "type": "String",
                "defaultValue": "0"
            },
            "j": {
                "type": "String",
                "defaultValue": "0"
            }
        },
        "annotations": []
    }
}
```

Variables are currently scoped at the pipeline level. This means that they are not thread safe and can cause unexpected and undesired behavior if they are accessed from within a parallel iteration activity such as a foreach loop, especially when the value is also being modified within that foreach activity.

Next steps

Learn about another related control flow activity:

- Append Variable Activity

Until activity in Azure Data Factory and Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

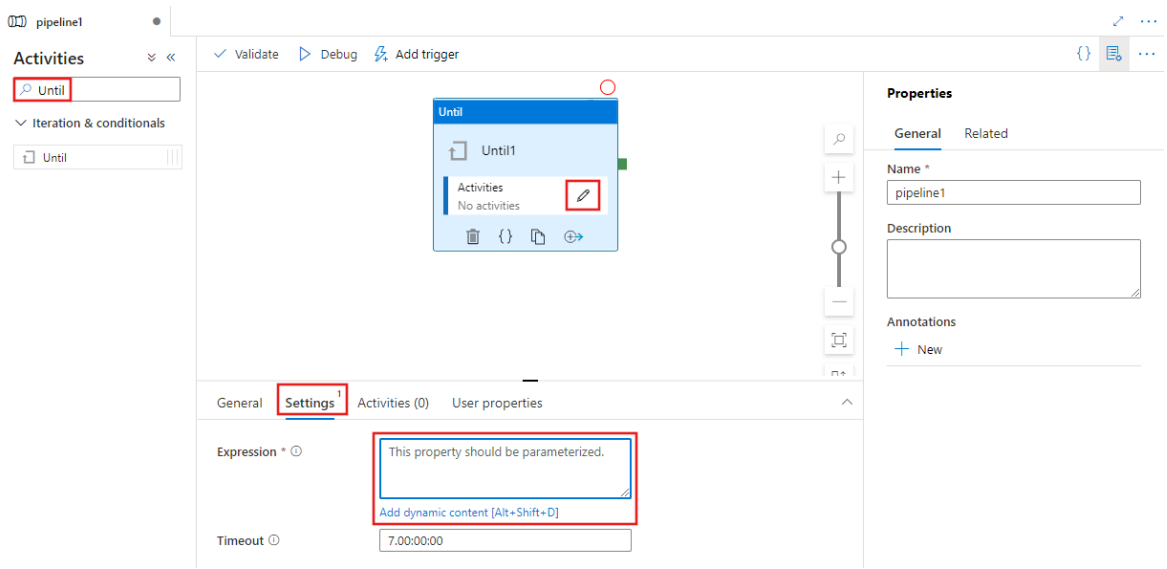
APPLIES TO: Azure Data Factory Azure Synapse Analytics

The Until activity provides the same functionality that a do-until looping structure provides in programming languages. It executes a set of activities in a loop until the condition associated with the activity evaluates to true. You can specify a timeout value for the until activity.

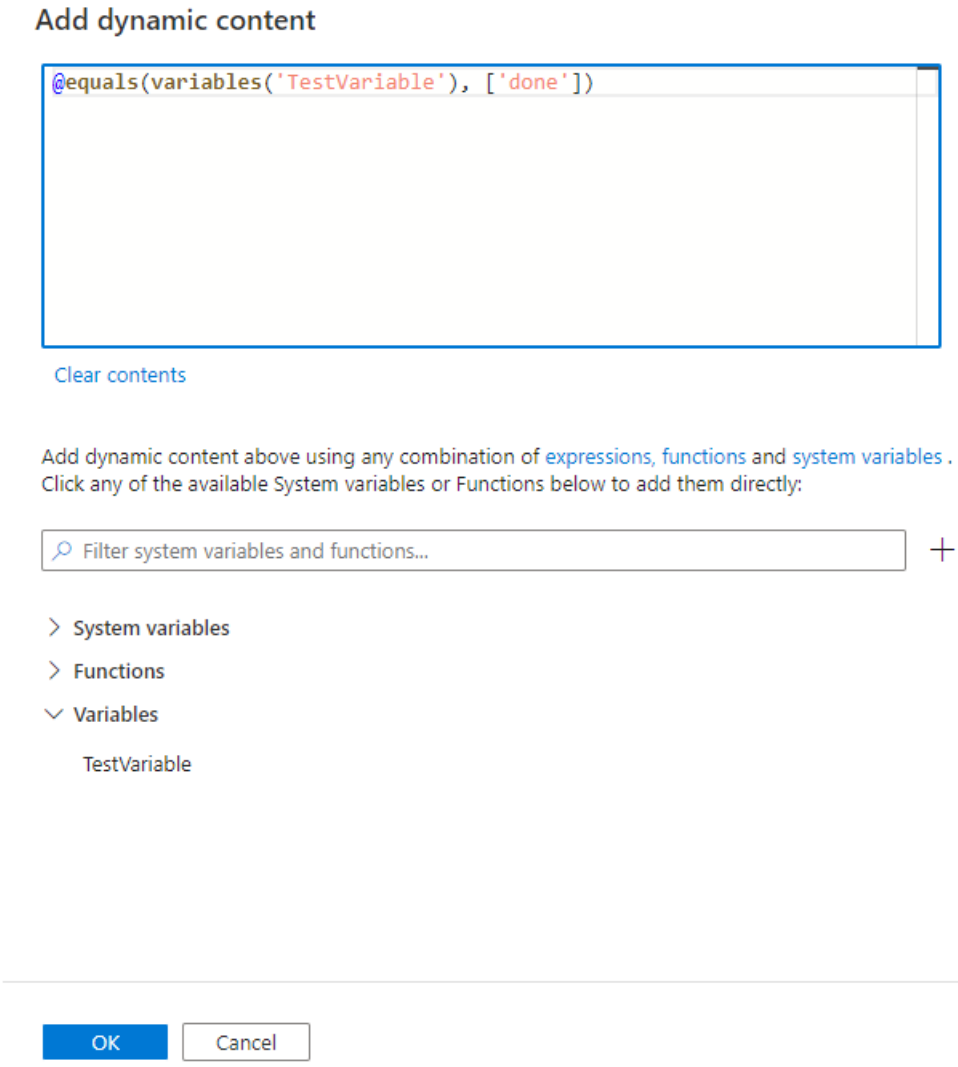
Create an Until activity with UI

To use an Until activity in a pipeline, complete the following steps:

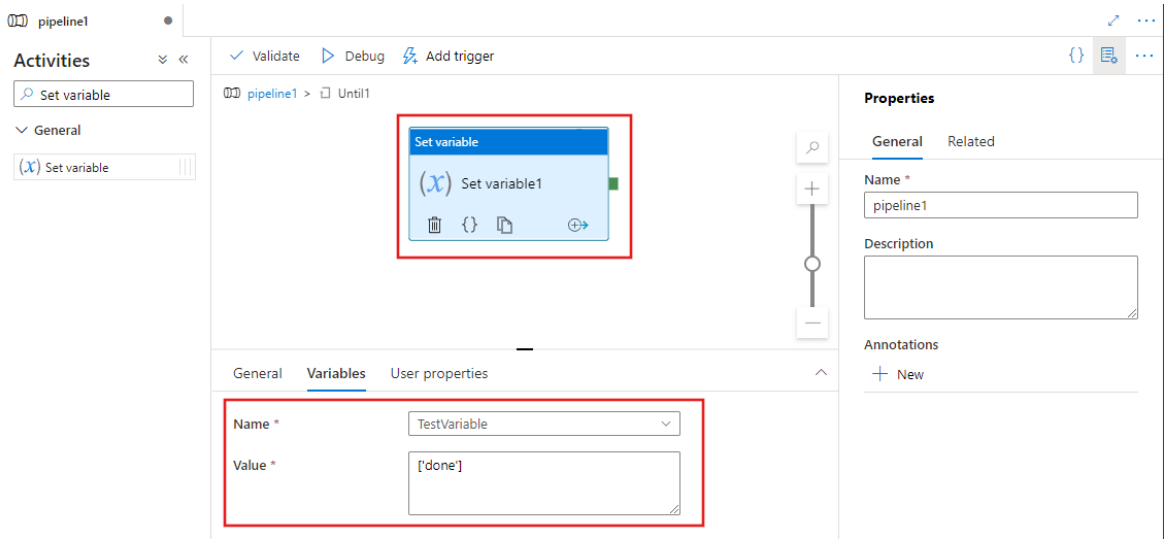
1. Search for *Until* in the pipeline Activities pane, and drag a Set Variable activity to the pipeline canvas.
2. Select the Until activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



3. Enter an expression that will be evaluated after all child activities defined in the Until activity are executed. If the expression evaluates to false, the Until activity will execute all its child activities again. When it evaluates to true, the Until activity will complete. The expression can be a literal string expression, or any combination of dynamic expressions, functions, system variables, or outputs from other activities. The example below checks the value of a previously defined pipeline array variable called TestVariable to see if it evaluates to ['done'].



4. Define activities that the Until activity will execute by selecting the Edit Activities button on the Until activity directly, or by selecting the Activities tab to select it there. A new activities editor pane will be displayed where you can add any activities for the Until activity to execute. In this example, a Set Variable activity simply sets the value of the variable referenced in the expression above to ['done'], so the Until activity's expression will be true the first time it is executed, and the Until activity will stop. In your real-world use, you can check any conditions required and the Until activity will continue to execute its child activities each time the expression is evaluated, until the conditions are met.



Syntax

```
{
    "type": "Until",
    "typeProperties": {
        "expression":  {
            "value":  "<expression that evaluates to true or false>",
            "type": "Expression"
        },
        "timeout": "<time out for the loop. for example: 00:01:00 (1 minute)>",
        "activities": [
            {
                "<Activity 1 definition>"
            },
            {
                "<Activity 2 definition>"
            },
            {
                "<Activity N definition>"
            }
        ]
    },
    "name": "MyUntilActivity"
}
```

Type properties

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
name	Name of the `Until` activity.	String	Yes
type	Must be set to **Until**.	String	Yes
expression	Expression that must evaluate to true or false	Expression.	Yes
timeout	The do-until loop times out after the specified time here.	String. `d.hh:mm:ss` (or) `hh:mm:ss`. The default value is 7 days. Maximum value is: 90 days.	No
Activities	Set of activities that are executed until expression evaluates to `true`.	Array of activities.	Yes

Example 1

> **NOTE**
> This section provides JSON definitions and sample PowerShell commands to run the pipeline. For a walkthrough with step-by-step instructions to create a pipeline by using Azure PowerShell and JSON definitions, see tutorial: create a data factory by using Azure PowerShell.

Pipeline with Until activity

In this example, the pipeline has two activities: **Until** and **Wait**. The Wait activity waits for the specified period of time before running the Web activity in the loop. To learn about expressions and functions, see Expression

language and functions.

```
{
    "name": "DoUntilPipeline",
    "properties": {
        "activities": [
            {
                "type": "Until",
                "typeProperties": {
                    "expression": {
                        "value": "@equals('Failed', coalesce(body('MyUnauthenticatedActivity')?.status,
actions('MyUnauthenticatedActivity')?.status, 'null'))",
                        "type": "Expression"
                    },
                    "timeout": "00:00:01",
                    "activities": [
                        {
                            "name": "MyUnauthenticatedActivity",
                            "type": "WebActivity",
                            "typeProperties": {
                                "method": "get",
                                "url": "https://www.fake.com/",
                                "headers": {
                                    "Content-Type": "application/json"
                                }
                            },
                            "dependsOn": [
                                {
                                    "activity": "MyWaitActivity",
                                    "dependencyConditions": [ "Succeeded" ]
                                }
                            ]
                        },
                        {
                            "type": "Wait",
                            "typeProperties": {
                                "waitTimeInSeconds": 1
                            },
                            "name": "MyWaitActivity"
                        }
                    ]
                },
                "name": "MyUntilActivity"
            }
        ]
    }
}
```

Example 2

The pipeline in this sample copies data from an input folder to an output folder in a loop. The loop terminates when the value for the repeat parameter is set to false or it times out after one minute.

Pipeline with Until activity (Adfv2QuickStartPipeline.json)

```
{
    "name": "Adfv2QuickStartPipeline",
    "properties": {
        "activities": [
            {
                "type": "Until",
                "typeProperties": {
                    "expression":  {
                        "value":  "@equals('false', pipeline().parameters.repeat)",
```

```
                        "type": "Expression"
                    },
                    "timeout": "00:01:00",
                    "activities": [
                        {
                            "name": "CopyFromBlobToBlob",
                            "type": "Copy",
                            "inputs": [
                                {
                                    "referenceName": "BlobDataset",
                                    "parameters": {
                                        "path": "@pipeline().parameters.inputPath"
                                    },
                                    "type": "DatasetReference"
                                }
                            ],
                            "outputs": [
                                {
                                    "referenceName": "BlobDataset",
                                    "parameters": {
                                        "path": "@pipeline().parameters.outputPath"
                                    },
                                    "type": "DatasetReference"
                                }
                            ],
                            "typeProperties": {
                                "source": {
                                    "type": "BlobSource"
                                },
                                "sink": {
                                    "type": "BlobSink"
                                }
                            },
                            "policy": {
                                "retry": 1,
                                "timeout": "00:10:00",
                                "retryIntervalInSeconds": 60
                            }
                        }
                    ]
                },
                "name": "MyUntilActivity"
            }
        ],
        "parameters": {
            "inputPath": {
                "type": "String"
            },
            "outputPath": {
                "type": "String"
            },
            "repeat": {
                "type": "String"
            }
        }
    }
}
```

Azure Storage linked service (AzureStorageLinkedService.json)

```
{
    "name": "AzureStorageLinkedService",
    "properties": {
        "type": "AzureStorage",
        "typeProperties": {
            "connectionString": "DefaultEndpointsProtocol=https;AccountName=<Azure Storage account
name>;AccountKey=<Azure Storage account key>"
        }
    }
}
```

Parameterized Azure Blob dataset (BlobDataset.json)

The pipeline sets the **folderPath** to the value of either **outputPath1** or **outputPath2** parameter of the pipeline.

```
{
    "name": "BlobDataset",
    "properties": {
        "type": "AzureBlob",
        "typeProperties": {
            "folderPath": {
                "value": "@{dataset().path}",
                "type": "Expression"
            }
        },
        "linkedServiceName": {
            "referenceName": "AzureStorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "parameters": {
            "path": {
                "type": "String"
            }
        }
    }
}
```

Pipeline parameter JSON (PipelineParameters.json)

```
{
    "inputPath": "adftutorial/input",
    "outputPath": "adftutorial/outputUntil",
    "repeat": "true"
}
```

PowerShell commands

> **NOTE**
>
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

These commands assume that you have saved the JSON files into the folder: C:\ADF.

```
Connect-AzAccount
Select-AzSubscription "<Your subscription name>"

$resourceGroupName = "<Resource Group Name>"
$dataFactoryName = "<Data Factory Name. Must be globally unique>";
Remove-AzDataFactoryV2 $dataFactoryName -ResourceGroupName $resourceGroupName -force


Set-AzDataFactoryV2 -ResourceGroupName $resourceGroupName -Location "East US" -Name $dataFactoryName
Set-AzDataFactoryV2LinkedService -DataFactoryName $dataFactoryName -ResourceGroupName $resourceGroupName -
Name "AzureStorageLinkedService" -DefinitionFile "C:\ADF\AzureStorageLinkedService.json"
Set-AzDataFactoryV2Dataset -DataFactoryName $dataFactoryName -ResourceGroupName $resourceGroupName -Name
"BlobDataset" -DefinitionFile "C:\ADF\BlobDataset.json"
Set-AzDataFactoryV2Pipeline -DataFactoryName $dataFactoryName -ResourceGroupName $resourceGroupName -Name
"Adfv2QuickStartPipeline" -DefinitionFile "C:\ADF\Adfv2QuickStartPipeline.json"
$runId = Invoke-AzDataFactoryV2Pipeline -DataFactoryName $dataFactoryName -ResourceGroupName
$resourceGroupName -PipelineName "Adfv2QuickStartPipeline" -ParameterFile C:\ADF\PipelineParameters.json

while ($True) {
    $run = Get-AzDataFactoryV2PipelineRun -ResourceGroupName $resourceGroupName -DataFactoryName
$DataFactoryName -PipelineRunId $runId

    if ($run) {
        if ($run.Status -ne 'InProgress') {
            Write-Host "Pipeline run finished. The status is: " $run.Status -foregroundcolor "Yellow"
            $run
            break
        }
        Write-Host  "Pipeline is running...status: InProgress" -foregroundcolor "Yellow"
        Write-Host "Activity run details:" -foregroundcolor "Yellow"
        $result = Get-AzDataFactoryV2ActivityRun -DataFactoryName $dataFactoryName -ResourceGroupName
$resourceGroupName -PipelineRunId $runId -RunStartedAfter (Get-Date).AddMinutes(-30) -RunStartedBefore (Get-
Date).AddMinutes(30)
        $result

        Write-Host "Activity 'Output' section:" -foregroundcolor "Yellow"
        $result.Output -join "`r`n"
    }

    Start-Sleep -Seconds 15
}
```

Next steps

See other supported control flow activities:

- If Condition Activity
- Execute Pipeline Activity
- For Each Activity
- Get Metadata Activity
- Lookup Activity
- Web Activity

Validation activity in Azure Data Factory and Synapse Analytics pipelines

8/19/2022 • 2 minutes to read • Edit Online

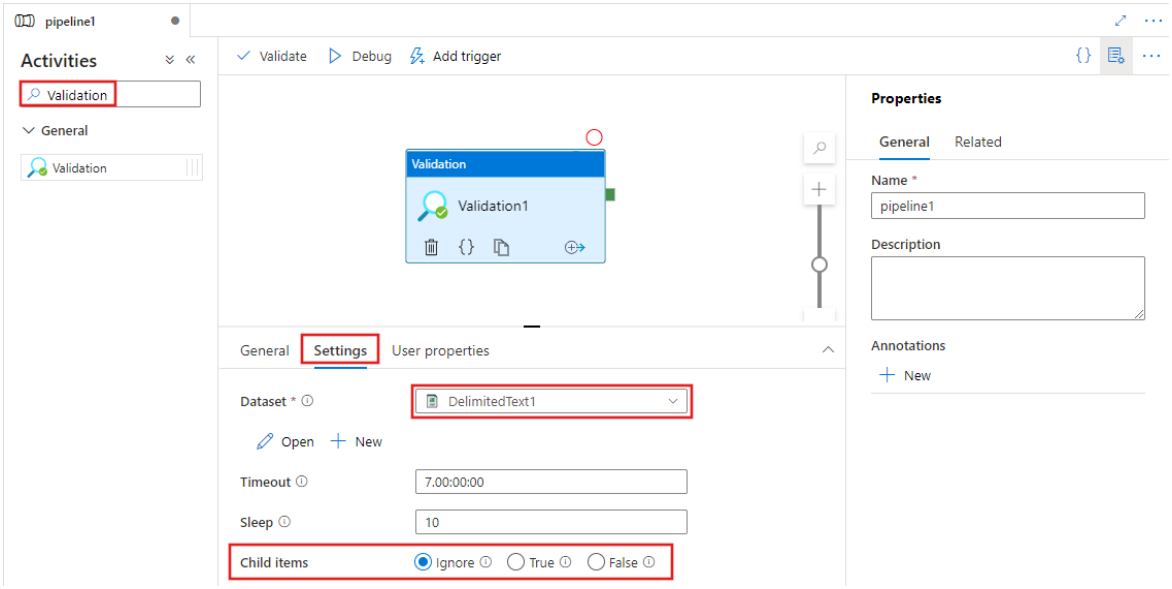
APPLIES TO: Azure Data Factory Azure Synapse Analytics

You can use a Validation in a pipeline to ensure the pipeline only continues execution once it has validated the attached dataset reference exists, that it meets the specified criteria, or timeout has been reached.

Create a Validation activity with UI

To use a Validation activity in a pipeline, complete the following steps:

1. Search for *Validation* in the pipeline Activities pane, and drag a Validation activity to the pipeline canvas.
2. Select the new Validation activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



3. Select a dataset, or define a new one by selecting the New button. For file based datasets like the delimited text example above, you can select either a specific file, or a folder. When a folder is selected, the Validation activity allows you to ignore validation of the existence of child items in the folder, or require whether child items exist or not.
4. The output of the Validation activity can be used as an input to any other activities, and referenced within those activities for any of their properties using dynamic expressions.

Syntax

```
{
    "name": "Validation_Activity",
    "type": "Validation",
    "typeProperties": {
        "dataset": {
            "referenceName": "Storage_File",
            "type": "DatasetReference"
        },
        "timeout": "7.00:00:00",
        "sleep": 10,
        "minimumSize": 20
    }
},
{
    "name": "Validation_Activity_Folder",
    "type": "Validation",
    "typeProperties": {
        "dataset": {
            "referenceName": "Storage_Folder",
            "type": "DatasetReference"
        },
        "timeout": "7.00:00:00",
        "sleep": 10,
        "childItems": true
    }
}
```

Type properties

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
name	Name of the 'Validation' activity	String	Yes
type	Must be set to **Validation**.	String	Yes
dataset	Activity will block execution until it has validated this dataset reference exists and that it meets the specified criteria, or timeout has been reached. Dataset provided should support "MinimumSize" or "ChildItems" property.	Dataset reference	Yes
timeout	Specifies the timeout for the activity to run. If no value is specified, default value is 7 days ("7.00:00:00"). Format is d.hh:mm:ss	String	No
sleep	A delay in seconds between validation attempts. If no value is specified, default value is 10 seconds.	Integer	No

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
childItems	Checks if the folder has child items. Can be set to-true : Validate that the folder exists and that it has items. Blocks until at least one item is present in the folder or timeout value is reached.-false: Validate that the folder exists and that it is empty. Blocks until folder is empty or until timeout value is reached. If no value is specified, activity will block until the folder exists or until timeout is reached.	Boolean	No
minimumSize	Minimum size of a file in bytes. If no value is specified, default value is 0 bytes	Integer	No

Next steps

See other supported control flow activities:

- If Condition Activity
- Execute Pipeline Activity
- For Each Activity
- Get Metadata Activity
- Lookup Activity
- Web Activity
- Until Activity

Execute Wait activity in Azure Data Factory and Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

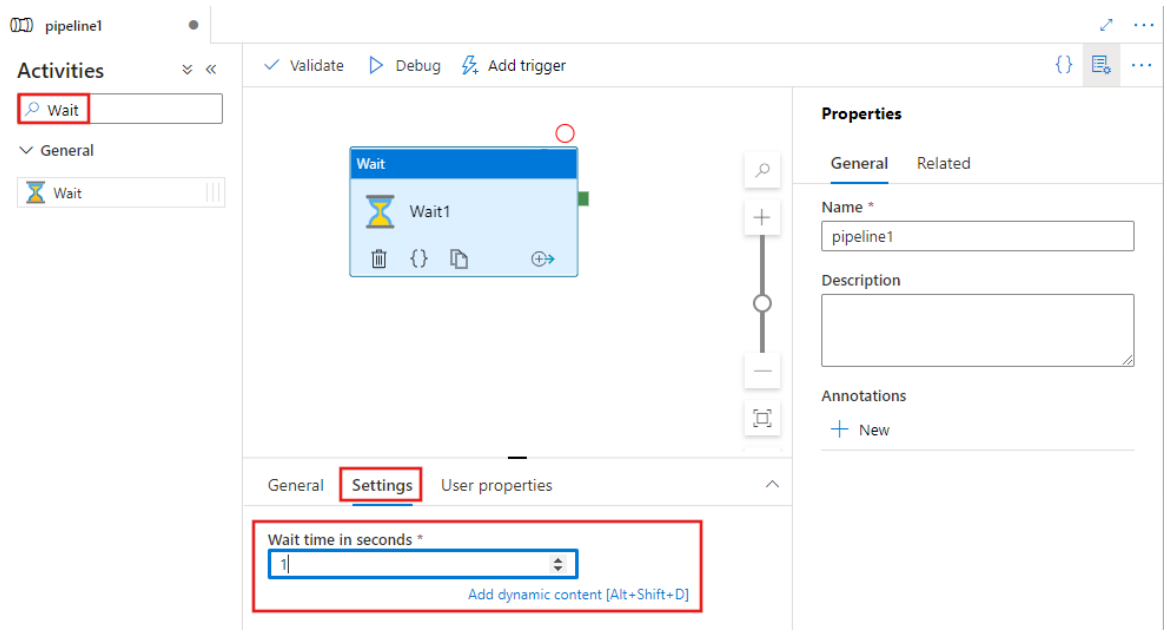
When you use a Wait activity in a pipeline, the pipeline waits for the specified period of time before continuing with execution of subsequent activities.

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Create a Wait activity with UI

To use a Wait activity in a pipeline, complete the following steps:

1. Search for *Wait* in the pipeline Activities pane, and drag a Wait activity to the pipeline canvas.
2. Select the new Wait activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



3. Enter a number of seconds for the activity to wait. This can be a literal number, or any combination of dynamic expressions, functions, system variables, or outputs from other activities.

Syntax

```
{
    "name": "MyWaitActivity",
    "type": "Wait",
    "typeProperties": {
        "waitTimeInSeconds": 1
    }
}
```

Type properties

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
name	Name of the `Wait` activity.	String	Yes
type	Must be set to **Wait**.	String	Yes
waitTimeInSeconds	The number of seconds that the pipeline waits before continuing with the processing.	Integer	Yes

Example

> **NOTE**
> This section provides JSON definitions and sample PowerShell commands to run the pipeline. For a walkthrough with step-by-step instructions to create a pipeline by using Azure PowerShell and JSON definitions, see tutorial: create a data factory by using Azure PowerShell.

Pipeline with Wait activity

In this example, the pipeline has two activities: **Until** and **Wait**. The Wait activity is configured to wait for one second. The pipeline runs the Web activity in a loop with one second waiting time between each run.

```
{
    "name": "DoUntilPipeline",
    "properties": {
        "activities": [
            {
                "type": "Until",
                "typeProperties": {
                    "expression": {
                        "value": "@equals('Failed', coalesce(body('MyUnauthenticatedActivity')?.status,
actions('MyUnauthenticatedActivity')?.status, 'null'))",
                        "type": "Expression"
                    },
                    "timeout": "00:00:01",
                    "activities": [
                        {
                            "name": "MyUnauthenticatedActivity",
                            "type": "WebActivity",
                            "typeProperties": {
                                "method": "get",
                                "url": "https://www.fake.com/",
                                "headers": {
                                    "Content-Type": "application/json"
                                }
                            },
                            "dependsOn": [
                                {
                                    "activity": "MyWaitActivity",
                                    "dependencyConditions": [ "Succeeded" ]
                                }
                            ]
                        },
                        {
                            "type": "Wait",
                            "typeProperties": {
                                "waitTimeInSeconds": 1
                            },
                            "name": "MyWaitActivity"
                        }
                    ]
                },
                "name": "MyUntilActivity"
            }
        ]
    }
}
```

Next steps

See other supported control flow activities:

- If Condition Activity
- Execute Pipeline Activity
- For Each Activity
- Get Metadata Activity
- Lookup Activity
- Web Activity
- Until Activity

Web activity in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Web Activity can be used to call a custom REST endpoint from an Azure Data Factory or Synapse pipeline. You can pass datasets and linked services to be consumed and accessed by the activity.

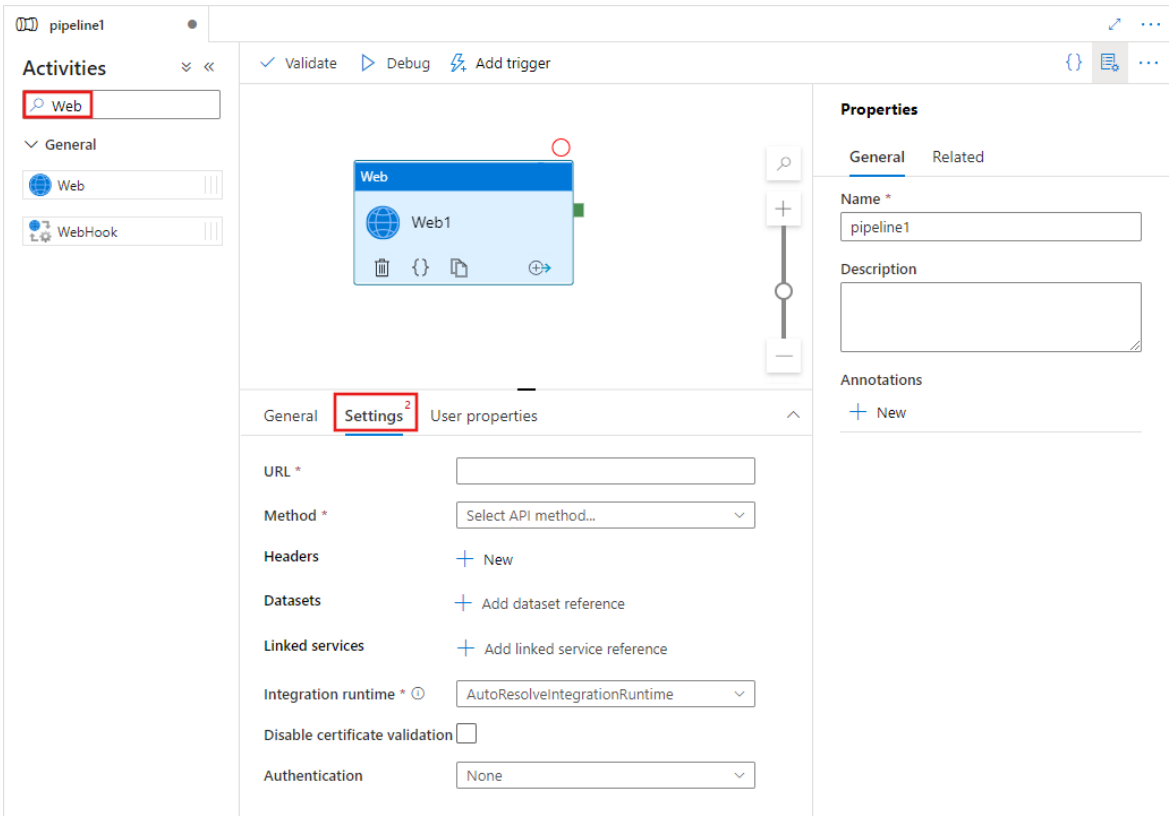
> **NOTE**
> Web Activity is supported for invoking URLs that are hosted in a private virtual network as well by leveraging self-hosted integration runtime. The integration runtime should have a line of sight to the URL endpoint.

> **NOTE**
> The maximum supported output response payload size is 4 MB.

Create a Web activity with UI

To use a Web activity in a pipeline, complete the following steps:

1. Search for *Web* in the pipeline Activities pane, and drag a Web activity to the pipeline canvas.
2. Select the new Web activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



3. Specify a URL, which can be a literal URL string, or any combination of dynamic expressions, functions, system variables, or outputs from other activities. Provide other details to be submitted with the request.

4. Use the output from the activity as the input to any other activity, and reference the output anywhere dynamic content is supported in the destination activity.

Syntax

```
{
   "name":"MyWebActivity",
   "type":"WebActivity",
   "typeProperties":{
      "method":"Post",
      "url":"<URLEndpoint>",
      "httpRequestTimeout": "00:01:00"
      "connectVia": {
          "referenceName": "<integrationRuntimeName>",
          "type": "IntegrationRuntimeReference"
      }
      "headers":{
         "Content-Type":"application/json"
      },
      "authentication":{
         "type":"ClientCertificate",
         "pfx":"****",
         "password":"****"
      },
      "datasets":[
         {
            "referenceName":"<ConsumedDatasetName>",
            "type":"DatasetReference",
            "parameters":{
               ...
            }
         }
      ],
      "linkedServices":[
         {
            "referenceName":"<ConsumedLinkedServiceName>",
            "type":"LinkedServiceReference"
         }
      ]
   }
}
```

Type properties

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
name	Name of the web activity	String	Yes
type	Must be set to **WebActivity**.	String	Yes
method	REST API method for the target endpoint.	String. Supported Types: "GET", "POST", "PUT", "PATCH", "DELETE"	Yes

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
url	Target endpoint and path	String (or expression with resultType of string). The activity will timeout at 1 minute with an error if it does not receive a response from the endpoint. You can increase this response timeout up to 10 mins by updating the httpRequestTimeout property	Yes
httpRequestTimeout	Response timeout duration	hh:mm:ss with the max value as 00:10:00. If not explicitly specified defaults to 00:01:00	No
headers	Headers that are sent to the request. For example, to set the language and type on a request: `"headers" : { "Accept-Language": "en-us", "Content-Type": "application/json" }` .	String (or expression with resultType of string)	No
body	Represents the payload that is sent to the endpoint.	String (or expression with resultType of string). See the schema of the request payload in Request payload schema section.	Required for POST/PUT/PATCH methods.
authentication	Authentication method used for calling the endpoint. Supported Types are "Basic, Client Certificate, System-assigned Managed Identity, User-assigned Managed Identity, Service Principal." For more information, see Authentication section. If authentication is not required, exclude this property.	String (or expression with resultType of string)	No
turnOffAsync	Option to disable invoking HTTP GET on location field in the response header of a HTTP 202 Response. If set true, it stops invoking HTTP GET on http location given in response header. If set false then it continues to invoke HTTP GET call on location given in http response headers.	Allowed values are false (default) and true.	No

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
disableCertValidation	Removes server side certificate validation (not recommended unless you are connecting to a trusted server that does not use a standard CA cert).	Allowed values are false (default) and true.	No
datasets	List of datasets passed to the endpoint.	Array of dataset references. Can be an empty array.	Yes
linkedServices	List of linked services passed to endpoint.	Array of linked service references. Can be an empty array.	Yes
connectVia	The integration runtime to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	The integration runtime reference.	No

NOTE

REST endpoints that the web activity invokes must return a response of type JSON. The activity will timeout at 1 minute with an error if it does not receive a response from the endpoint. For endpoints that support Asynchronous Request-Reply pattern, the web activity will continue to wait without timeing out (upto 7 day) or till the endpoints signals completion of the job.

The following table shows the requirements for JSON content:

VALUE TYPE	REQUEST BODY	RESPONSE BODY
JSON object	Supported	Supported
JSON array	Supported (At present, JSON arrays don't work as a result of a bug. A fix is in progress.)	Unsupported
JSON value	Supported	Unsupported
Non-JSON type	Unsupported	Unsupported

Authentication

Below are the supported authentication types in the web activity.

None

If authentication is not required, do not include the "authentication" property.

Basic

Specify user name and password to use with the basic authentication.

```
"authentication":{
   "type":"Basic",
   "username":"****",
   "password":"****"
}
```

Client certificate

Specify base64-encoded contents of a PFX file and the password.

```
"authentication":{
   "type":"ClientCertificate",
   "pfx":"****",
   "password":"****"
}
```

Certificate needs to be an x509 certificate. For conversion to PFX file, you can use your favorite utility. For base-64 encoding, you may use following PowerShell snippet.

```
$fileContentBytes = get-content 'enr.dev.webactivity.pfx' -AsByteStream

[System.Convert]::ToBase64String($fileContentBytes) | Out-File ‘pfx-encoded-bytes.txt’
```

Managed Identity

Specify the resource uri for which the access token will be requested using the managed identity for the data factory or Synapse workspace instance. To call the Azure Resource Management API, use `https://management.azure.com/`. For more information about how managed identities works see the managed identities for Azure resources overview page.

```
"authentication": {
 "type": "MSI",
 "resource": "https://management.azure.com/"
}
```

> **NOTE**
> If your data factory or Synapse workspace is configured with a git repository, you must store your credentials in Azure Key Vault to use basic or client certificate authentication. The service does not store passwords in git.

Request payload schema

When you use the POST/PUT method, the body property represents the payload that is sent to the endpoint. You can pass linked services and datasets as part of the payload. Here is the schema for the payload:

```
{
    "body": {
        "myMessage": "Sample",
        "datasets": [{
            "name": "MyDataset1",
            "properties": {
                ...
            }
        }],
        "linkedServices": [{
            "name": "MyStorageLinkedService1",
            "properties": {
                ...
            }
        }]
    }
}
```

Example

In this example, the web activity in the pipeline calls a REST end point. It passes an Azure SQL linked service and an Azure SQL dataset to the endpoint. The REST end point uses the Azure SQL connection string to connect to the logical SQL server and returns the name of the instance of SQL server.

Pipeline definition

```
{
    "name": "<MyWebActivityPipeline>",
    "properties": {
        "activities": [
            {
                "name": "<MyWebActivity>",
                "type": "WebActivity",
                "typeProperties": {
                    "method": "Post",
                    "url": "@pipeline().parameters.url",
                    "headers": {
                        "Content-Type": "application/json"
                    },
                    "authentication": {
                        "type": "ClientCertificate",
                        "pfx": "*****",
                        "password": "*****"
                    },
                    "datasets": [
                        {
                            "referenceName": "MySQLDataset",
                            "type": "DatasetReference",
                            "parameters": {
                                "SqlTableName": "@pipeline().parameters.sqlTableName"
                            }
                        }
                    ],
                    "linkedServices": [
                        {
                            "referenceName": "SqlLinkedService",
                            "type": "LinkedServiceReference"
                        }
                    ]
                }
            }
        ],
        "parameters": {
            "sqlTableName": {
                "type": "String"
            },
            "url": {
                "type": "String"
            }
        }
    }
}
```

Pipeline parameter values

```
{
    "sqlTableName": "department",
    "url": "https://adftes.azurewebsites.net/api/execute/running"
}
```

Web service endpoint code

```
[HttpPost]
public HttpResponseMessage Execute(JObject payload)
{
    Trace.TraceInformation("Start Execute");

    JObject result = new JObject();
    result.Add("status", "complete");

    JArray datasets = payload.GetValue("datasets") as JArray;
    result.Add("sinktable", datasets[0]["properties"]["typeProperties"]["tableName"].ToString());

    JArray linkedServices = payload.GetValue("linkedServices") as JArray;
    string connString = linkedServices[0]["properties"]["typeProperties"]["connectionString"].ToString();

    System.Data.SqlClient.SqlConnection sqlConn = new System.Data.SqlClient.SqlConnection(connString);

    result.Add("sinkServer", sqlConn.DataSource);

    Trace.TraceInformation("Stop Execute");

    return this.Request.CreateResponse(HttpStatusCode.OK, result);
}
```

Next steps

See other supported control flow activities:

- Execute Pipeline Activity
- For Each Activity
- Get Metadata Activity
- Lookup Activity

Webhook activity in Azure Data Factory

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

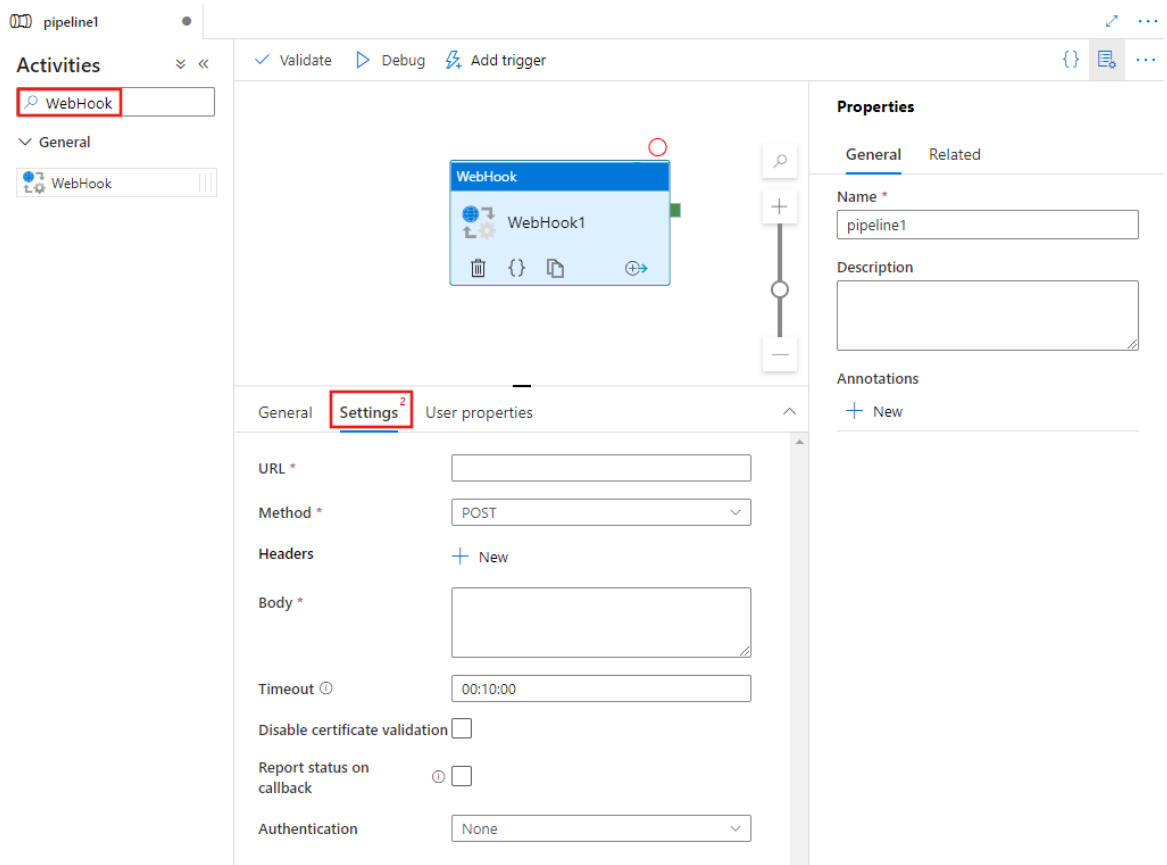
A webhook activity can control the execution of pipelines through custom code. With the webhook activity, code can call an endpoint and pass it a callback URL. The pipeline run waits for the callback invocation before it proceeds to the next activity.

> **IMPORTANT**
> WebHook activity now allows you to surface error status and custom messages back to activity and pipeline. Set *reportStatusOnCallBack* to true, and include *StatusCode* and *Error* in callback payload. For more information, see Additional Notes section.

Create a Webhook activity with UI

To use a Webhook activity in a pipeline, complete the following steps:

1. Search for *Webhook* in the pipeline Activities pane, and drag a Webhook activity to the pipeline canvas.
2. Select the new Fail activity on the canvas if it is not already selected, and its **Settings** tab, to edit its details.



3. Specify a URL for the webhook, which can be a literal URL string, or any combination of dynamic expressions, functions, system variables, or outputs from other activities. Provide other details to be submitted with the request.

4. Use the output from the activity as the input to any other activity, and reference the output anywhere dynamic content is supported in the destination activity.

Syntax

```
{
    "name": "MyWebHookActivity",
    "type": "WebHook",
    "typeProperties": {
        "method": "POST",
        "url": "<URLEndpoint>",
        "headers": {
            "Content-Type": "application/json"
        },
        "body": {
            "key": "value"
        },
        "timeout": "00:03:00",
        "reportStatusOnCallBack": false,
        "authentication": {
            "type": "ClientCertificate",
            "pfx": "****",
            "password": "****"
        }
    }
}
```

Type properties

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
name	The name of the webhook activity.	String	Yes
type	Must be set to "WebHook".	String	Yes
method	The REST API method for the target endpoint.	String. The supported type is "POST".	Yes
url	The target endpoint and path.	A string or an expression with the **resultType** value of a string.	Yes
headers	Headers that are sent to the request. Here's an example that sets the language and type on a request: `"headers" : { "Accept-Language": "en-us", "Content-Type": "application/json" }` .	A string or an expression with the **resultType** value of a string.	Yes. A `Content-Type` header like `"headers":{ "Content-Type":"application/json"}` is required.

PROPERTY	DESCRIPTION	ALLOWED VALUES	REQUIRED
body	Represents the payload that is sent to the endpoint.	Valid JSON or an expression with the **resultType** value of JSON. See Request payload schema for the schema of the request payload.	Yes
authentication	The authentication method used to call the endpoint. Supported types are "Basic" and "ClientCertificate". For more information, see Authentication. If authentication isn't required, exclude this property.	A string or an expression with the **resultType** value of a string.	No
timeout	How long the activity waits for the callback specified by **callBackUri** to be invoked. The default value is 10 minutes ("00:10:00"). Values have the TimeSpan format *d.hh:mm:ss*.	String	No
Report status on callback	Lets a user report the failed status of a webhook activity.	Boolean	No

Authentication

A webhook activity supports the following authentication types.

None

If authentication isn't required, don't include the **authentication** property.

Basic

Specify the username and password to use with basic authentication.

```
"authentication":{
   "type":"Basic",
   "username":"****",
   "password":"****"
}
```

Client certificate

Specify the Base64-encoded contents of a PFX file and a password.

```
"authentication":{
   "type":"ClientCertificate",
   "pfx":"****",
   "password":"****"
}
```

Managed identity

Use the managed identity for your data factory or Synapse workspace to specify the resource URI for which the access token is requested. To call the Azure Resource Management API, use `https://management.azure.com/`. For more information about how managed identities work, see the managed identities for Azure resources overview.

```
"authentication": {
    "type": "MSI",
    "resource": "https://management.azure.com/"
}
```

> **NOTE**
> If the service is configured with a Git repository, you must store your credentials in Azure Key Vault to use basic or client-certificate authentication. The service doesn't store passwords in Git.

Additional notes

The service passes the additional property **callBackUri** in the body sent to the URL endpoint. The service expects this URI to be invoked before the specified timeout value. If the URI isn't invoked, the activity fails with the status "TimedOut".

The webhook activity fails when the call to the custom endpoint fails. Any error message can be added to the callback body and used in a later activity.

For every REST API call, the client times out if the endpoint doesn't respond within one minute. This behavior is standard HTTP best practice. To fix this problem, implement a 202 pattern. In the current case, the endpoint returns 202 (Accepted) and the client polls.

The one-minute timeout on the request has nothing to do with the activity timeout. The latter is used to wait for the callback specified by **callbackUri**.

The body passed back to the callback URI must be valid JSON. Set the `Content-Type` header to `application/json`.

When you use the **Report status on callback** property, you must add the following code to the body when you make the callback:

```
{
    "Output": {
        // output object is used in activity output
        "testProp": "testPropValue"
    },
    "Error": {
        // Optional, set it when you want to fail the activity
        "ErrorCode": "testErrorCode",
        "Message": "error message to show in activity error"
    },
    "StatusCode": "403" // when status code is >=400, activity is marked as failed
}
```

Next steps

See the following supported control flow activities:

- If Condition Activity
- Execute Pipeline Activity

- For Each Activity
- Get Metadata Activity
- Lookup Activity
- Web Activity
- Until Activity

Mapping data flow transformation overview

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

Below is a list of the transformations currently supported in mapping data flow. Click on each transformations to learn its configuration details.

NAME	CATEGORY	DESCRIPTION
Aggregate	Schema modifier	Define different types of aggregations such as SUM, MIN, MAX, and COUNT grouped by existing or computed columns.
Alter row	Row modifier	Set insert, delete, update, and upsert policies on rows.
Assert	Row modifier	Set assert rules for each row.
Cast	Schema modifier	Change column data types with type checking.
Conditional split	Multiple inputs/outputs	Route rows of data to different streams based on matching conditions.
Derived column	Schema modifier	Generate new columns or modify existing fields using the data flow expression language.
External call	Schema modifier	Call external endpoints inline row-by-row.
Exists	Multiple inputs/outputs	Check whether your data exists in another source or stream.
Filter	Row modifier	Filter a row based upon a condition.
Flatten	Formatters	Take array values inside hierarchical structures such as JSON and unroll them into individual rows.
Flowlet	Flowlets	Build and include custom re-usable transformation logic.
Join	Multiple inputs/outputs	Combine data from two sources or streams.

NAME	CATEGORY	DESCRIPTION
Lookup	Multiple inputs/outputs	Reference data from another source.
New branch	Multiple inputs/outputs	Apply multiple sets of operations and transformations against the same data stream.
Parse	Formatters	Parse text columns in your data stream that are strings of JSON, delimited text, or XML formatted text.
Pivot	Schema modifier	An aggregation where one or more grouping columns has its distinct row values transformed into individual columns.
Rank	Schema modifier	Generate an ordered ranking based upon sort conditions
Select	Schema modifier	Alias columns and stream names, and drop or reorder columns
Sink	-	A final destination for your data
Sort	Row modifier	Sort incoming rows on the current data stream
Source	-	A data source for the data flow
Stringify	Formatters	Turn complex types into plain strings
Surrogate key	Schema modifier	Add an incrementing non-business arbitrary key value
Union	Multiple inputs/outputs	Combine multiple data streams vertically
Unpivot	Schema modifier	Pivot columns into row values
Window	Schema modifier	Define window-based aggregations of columns in your data streams.

Aggregate transformation in mapping data flow

8/19/2022 • 3 minutes to read • Edit Online

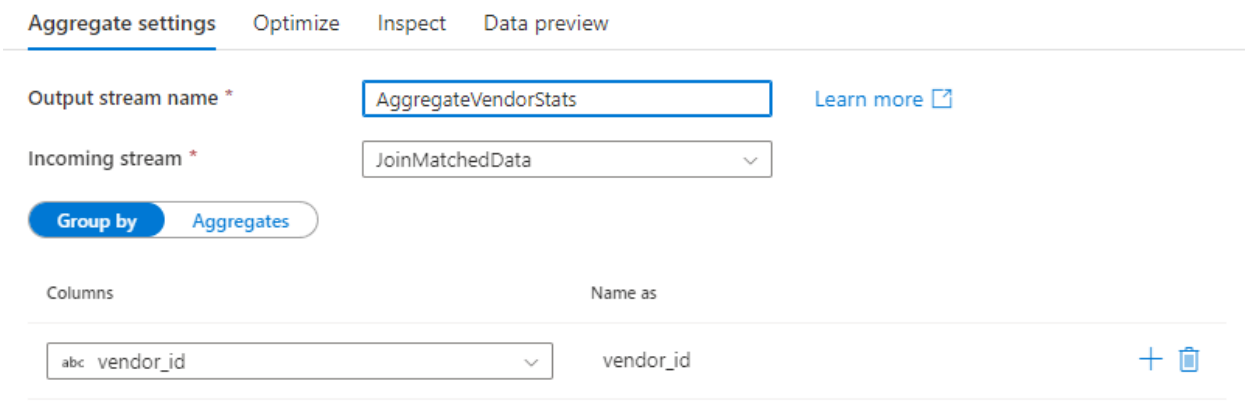
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

The Aggregate transformation defines aggregations of columns in your data streams. Using the Expression Builder, you can define different types of aggregations such as SUM, MIN, MAX, and COUNT grouped by existing or computed columns.

Group by

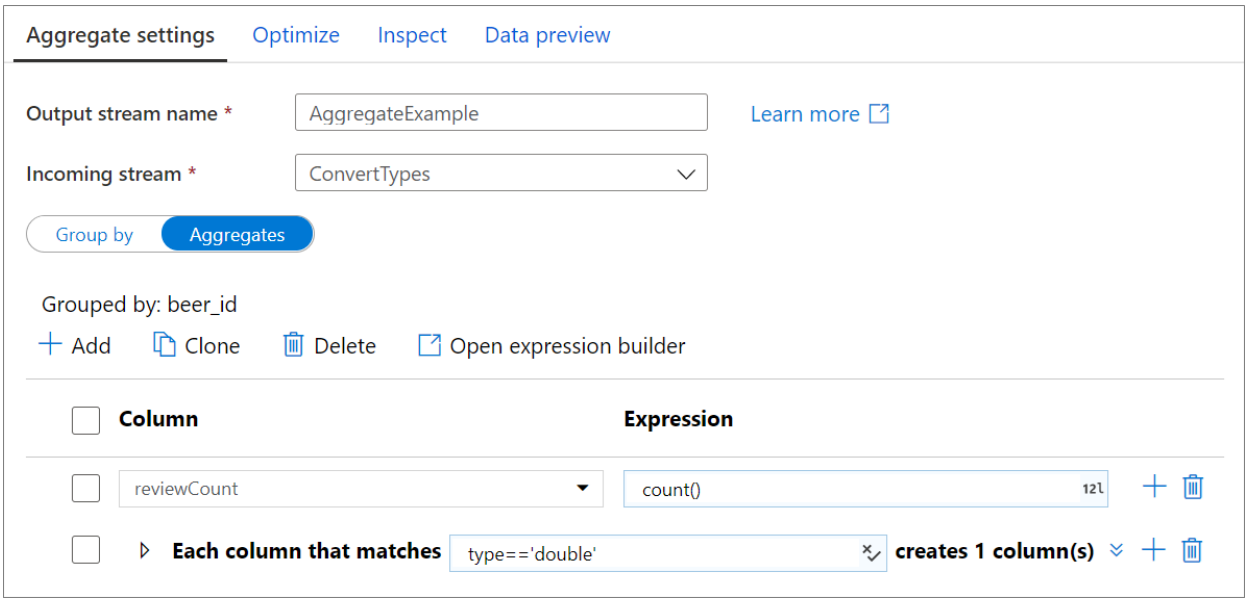
Select an existing column or create a new computed column to use as a group by clause for your aggregation. To use an existing column, select it from the dropdown. To create a new computed column, hover over the clause and click **Computed column**. This opens the data flow expression builder. Once you create your computed column, enter the output column name under the **Name as** field. If you wish to add an additional group by clause, hover over an existing clause and click the plus icon.



A group by clause is optional in an Aggregate transformation.

Aggregate columns

Go to the **Aggregates** tab to build aggregation expressions. You can either overwrite an existing column with an aggregation, or create a new field with a new name. The aggregation expression is entered in the right-hand box next to the column name selector. To edit the expression, click on the text box and open the expression builder. To add more aggregate columns, click on **Add** above the column list or the plus icon next to an existing aggregate column. Choose either **Add column** or **Add column pattern**. Each aggregation expression must contain at least one aggregate function.



NOTE

In Debug mode, the expression builder cannot produce data previews with aggregate functions. To view data previews for aggregate transformations, close the expression builder and view the data via the 'Data Preview' tab.

Column patterns

Use column patterns to apply the same aggregation to a set of columns. This is useful if you wish to persist many columns from the input schema as they are dropped by default. Use a heuristic such as `first()` to persist input columns through the aggregation.

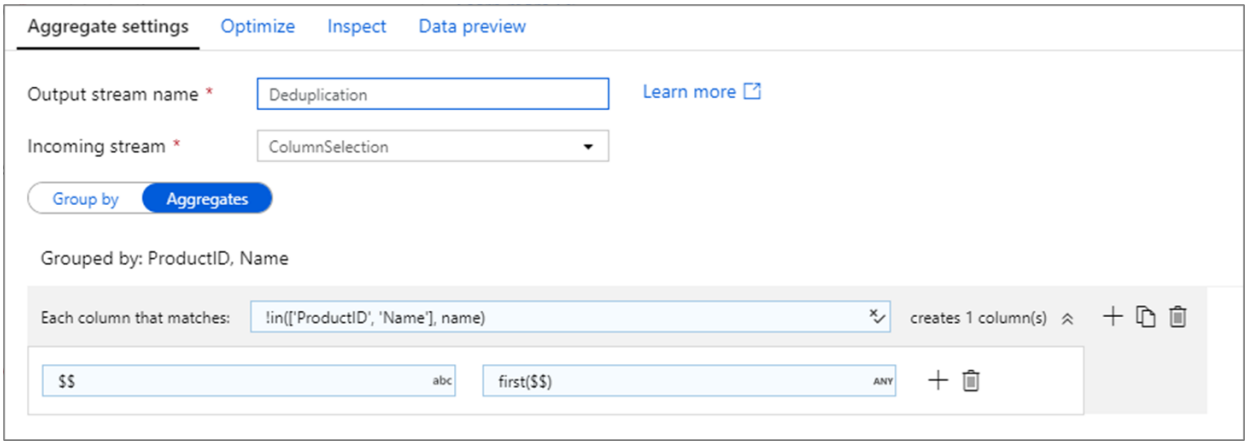
Reconnect rows and columns

Aggregate transformations are similar to SQL aggregate select queries. Columns that aren't included in your group by clause or aggregate functions won't flow through to the output of your aggregate transformation. If you wish to include other columns in your aggregated output, do one of the following methods:

- Use an aggregate function such as `last()` or `first()` to include that additional column.
- Rejoin the columns to your output stream using the self join pattern.

Removing duplicate rows

A common use of the aggregate transformation is removing or identifying duplicate entries in source data. This process is known as deduplication. Based upon a set of group by keys, use a heuristic of your choosing to determine which duplicate row to keep. Common heuristics are `first()`, `last()`, `max()`, and `min()`. Use column patterns to apply the rule to every column except for the group by columns.



In the above example, columns `ProductID` and `Name` are being use for grouping. If two rows have the same values for those two columns, they're considered duplicates. In this aggregate transformation, the values of the first row matched will be kept and all others will be dropped. Using column pattern syntax, all columns whose names aren't `ProductID` and `Name` are mapped to their existing column name and given the value of the first matched rows. The output schema is the same as the input schema.

For data validation scenarios, the `count()` function can be used to count how many duplicates there are.

Data flow script

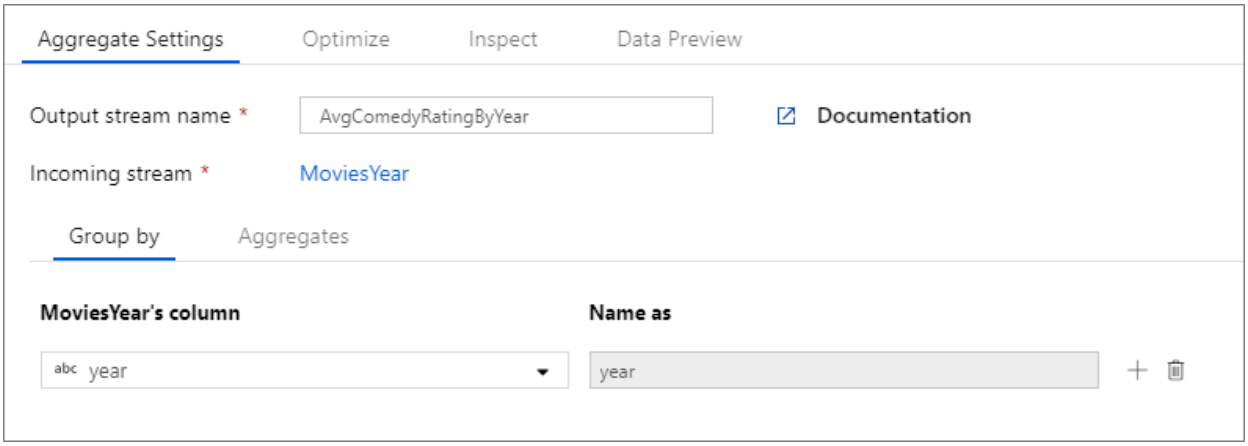
Syntax

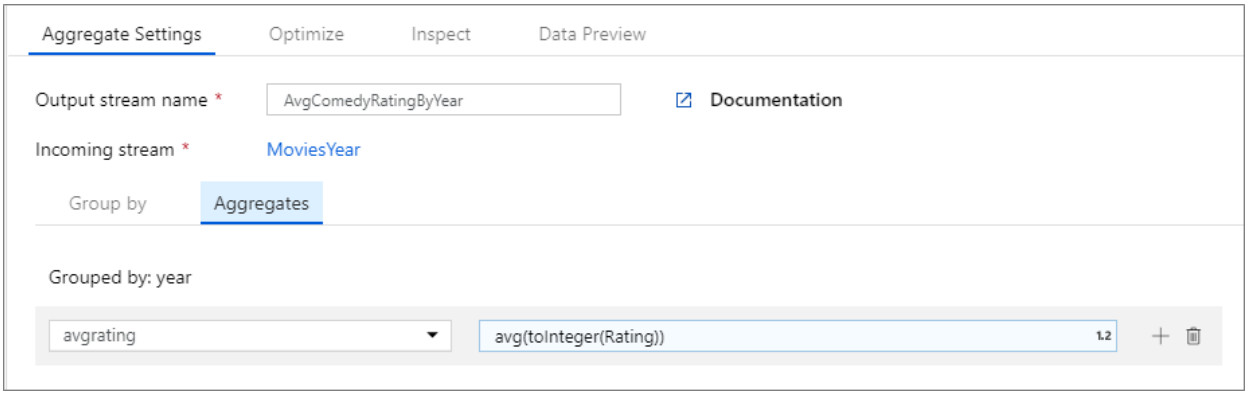
```
<incomingStream>
    aggregate(
           groupBy(
                <groupByColumnName> = <groupByExpression1>,
                <groupByExpression2>
               ),
           <aggregateColumn1> = <aggregateExpression1>,
           <aggregateColumn2> = <aggregateExpression2>,
           each(
                match(matchExpression),
                <metadataColumn1> = <metadataExpression1>,
                <metadataColumn2> = <metadataExpression2>
               )
          ) ~> <aggregateTransformationName>
```

Example

The below example takes an incoming stream `MoviesYear` and groups rows by column `year`. The transformation creates an aggregate column `avgrating` that evaluates to the average of column `Rating`. This aggregate transformation is named `AvgComedyRatingsByYear`.

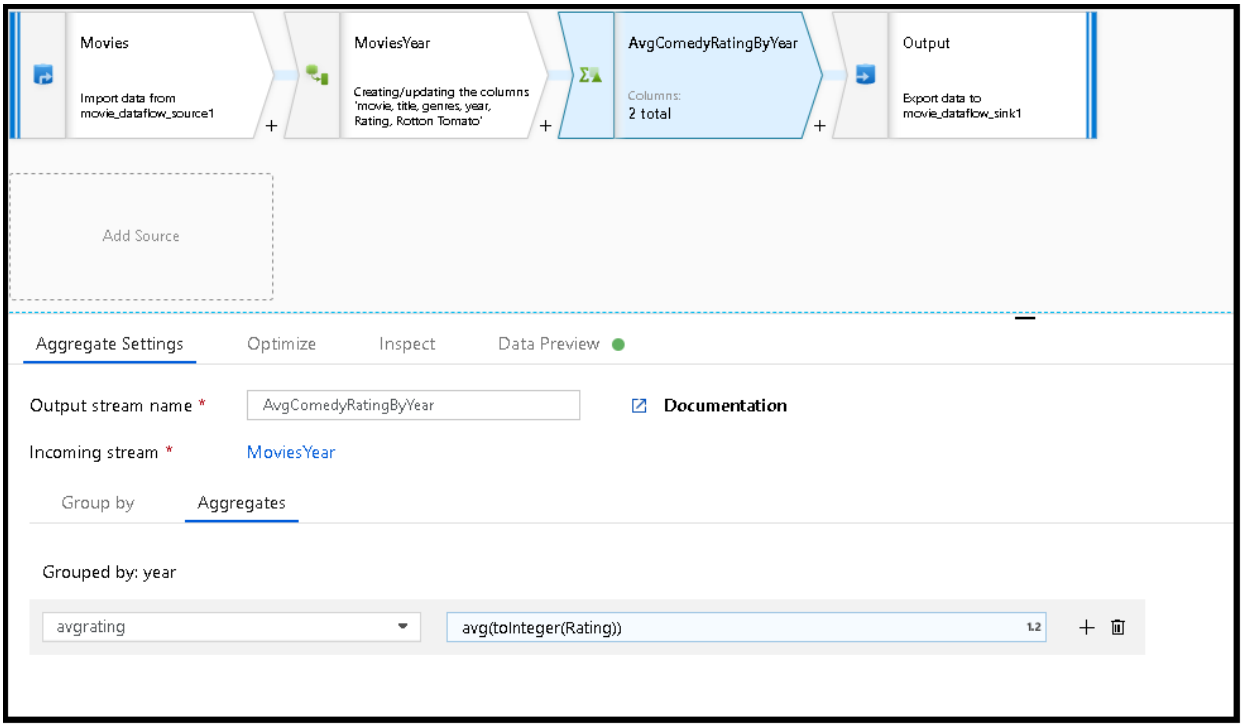
In the UI, this transformation looks like the below image:





The data flow script for this transformation is in the snippet below.

```
MoviesYear aggregate(
                groupBy(year),
                avgrating = avg(toInteger(Rating))
            ) ~> AvgComedyRatingByYear
```



`MoviesYear`: Derived Column defining year and title columns `AvgComedyRatingByYear`: Aggregate transformation for average rating of comedies grouped by year `avgrating`: Name of new column being created to hold the aggregated value

```
MoviesYear aggregate(groupBy(year),
 avgrating = avg(toInteger(Rating))) ~> AvgComedyRatingByYear
```

Next steps

- Define window-based aggregation using the Window transformation

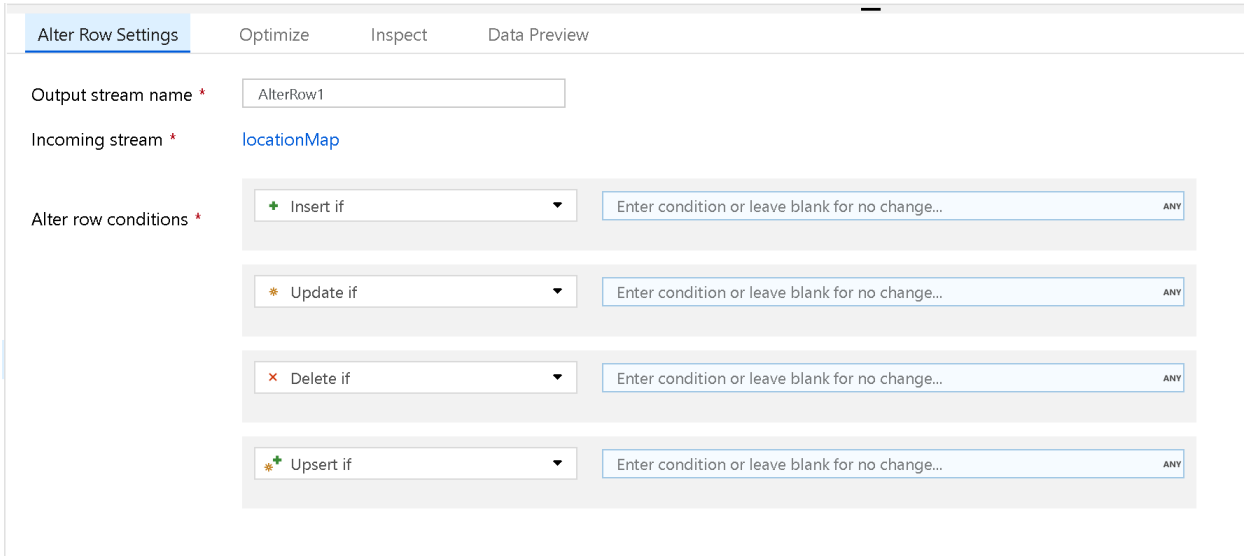
Alter row transformation in mapping data flow

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

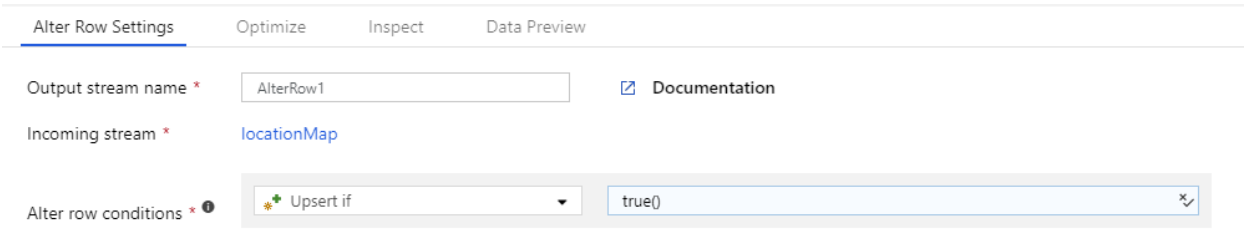
Use the Alter Row transformation to set insert, delete, update, and upsert policies on rows. You can add one-to-many conditions as expressions. These conditions should be specified in order of priority, as each row will be marked with the policy corresponding to the first-matching expression. Each of those conditions can result in a row (or rows) being inserted, updated, deleted, or upserted. Alter Row can produce both DDL & DML actions against your database.



Alter Row transformations will only operate on database, REST, or CosmosDB sinks in your data flow. The actions that you assign to rows (insert, update, delete, upsert) won't occur during debug sessions. Run an Execute Data Flow activity in a pipeline to enact the alter row policies on your database tables.

Specify a default row policy

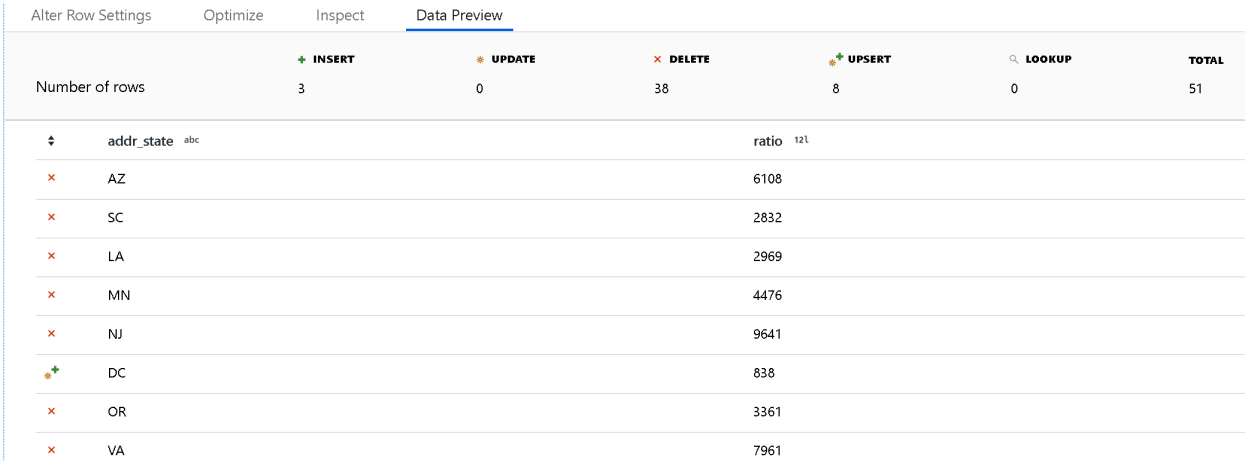
Create an Alter Row transformation and specify a row policy with a condition of `true()`. Each row that doesn't match any of the previously defined expressions will be marked for the specified row policy. By default, each row that doesn't match any conditional expression will be marked for `Insert`.



> **NOTE**
>
> To mark all rows with one policy, you can create a condition for that policy and specify the condition as `true()`.

View policies in data preview

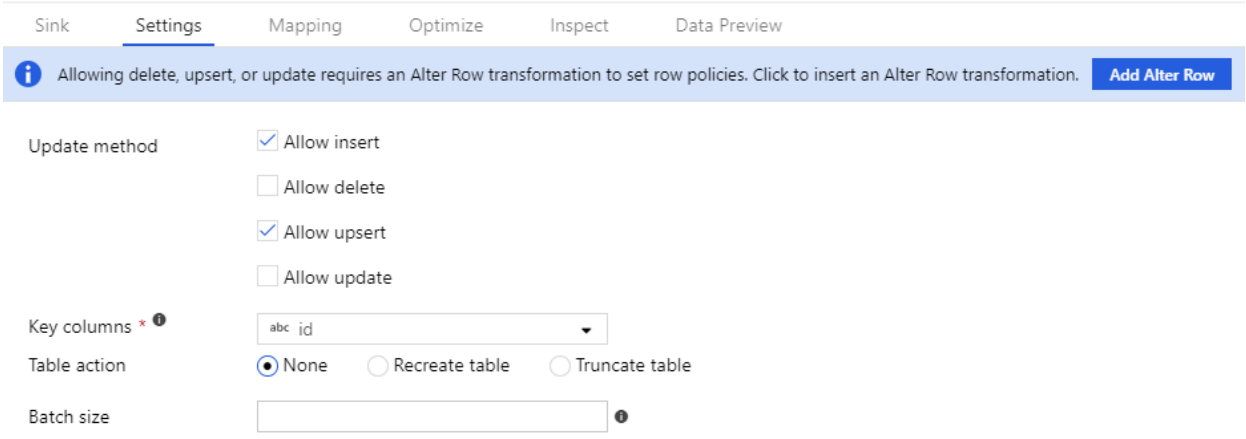
Use debug mode to view the results of your alter row policies in the data preview pane. A data preview of an alter row transformation won't produce DDL or DML actions against your target.



Each alter row policy is represented by an icon that indicates whether an insert, update, upsert, or deleted action will occur. The top header shows how many rows are affected by each policy in the preview.

Allow alter row policies in sink

For the alter row policies to work, the data stream must write to a database or Cosmos sink. In the **Settings** tab in your sink, enable which alter row policies are allowed for that sink.



The default behavior is to only allow inserts. To allow updates, upserts, or deletes, check the box in the sink corresponding to that condition. If updates, upserts, or, deletes are enabled, you must specify which key columns in the sink to match on.

> **NOTE**
>
> If your inserts, updates, or upserts modify the schema of the target table in the sink, the data flow will fail. To modify the target schema in your database, choose **Recreate table** as the table action. This will drop and recreate your table with the new schema definition.

The sink transformation requires either a single key or a series of keys for unique row identification in your

target database. For SQL sinks, set the keys in the sink settings tab. For CosmosDB, set the partition key in the settings and also set the CosmosDB system field "id" in your sink mapping. For CosmosDB, it is mandatory to include the system column "id" for updates, upserts, and deletes.

Merges and upserts with Azure SQL Database and Azure Synapse

Data Flows support merges against Azure SQL Database and Azure Synapse database pool (data warehouse) with the upsert option.

However, you may run into scenarios where your target database schema utilized the identity property of key columns. The service requires you to identify the keys that you will use to match the row values for updates and upserts. But if the target column has the identity property set and you are using the upsert policy, the target database will not allow you to write to the column. You may also run into errors when you try to upsert against a distributed table's distribution column.

Here are ways to fix that:

1. Go to the Sink transformation Settings and set "Skip writing key columns". This will tell the service to not write the column that you have selected as the key value for your mapping.
2. If that key column is not the column that is causing the issue for identity columns, then you can use the Sink transformation pre-processing SQL option: `SET IDENTITY_INSERT tbl_content ON`. Then, turn it off with the post-processing SQL property: `SET IDENTITY_INSERT tbl_content OFF`.
3. For both the identity case and the distribution column case, you can switch your logic from Upsert to using a separate update condition and a separate insert condition using a Conditional Split transformation. This way, you can set the mapping on the update path to ignore the key column mapping.

Data flow script

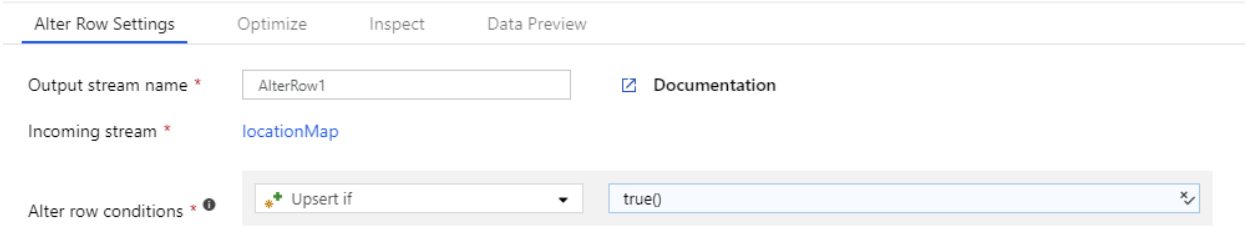
Syntax

```
<incomingStream>
    alterRow(
           insertIf(<condition>?),
           updateIf(<condition>?),
           deleteIf(<condition>?),
           upsertIf(<condition>?),
        ) ~> <alterRowTransformationName>
```

Example

The below example is an alter row transformation named `CleanData` that takes an incoming stream `SpecifyUpsertConditions` and creates three alter row conditions. In the previous transformation, a column named `alterRowCondition` is calculated that determines whether or not a row is inserted, updated, or deleted in the database. If the value of the column has a string value that matches the alter row rule, it is assigned that policy.

In the UI, this transformation looks like the below image:



The data flow script for this transformation is in the snippet below:

```
SpecifyUpsertConditions alterRow(insertIf(alterRowCondition == 'insert'),
 updateIf(alterRowCondition == 'update'),
 deleteIf(alterRowCondition == 'delete')) ~> AlterRow
```

Next steps

After the Alter Row transformation, you may want to sink your data into a destination data store.

Conditional split transformation in mapping data flow

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

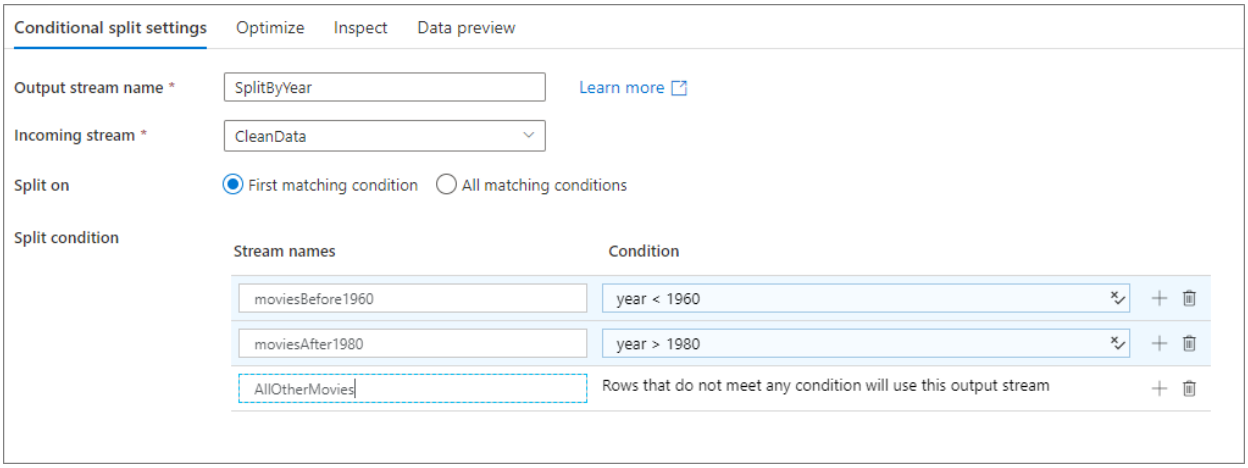
Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

The conditional split transformation routes data rows to different streams based on matching conditions. The conditional split transformation is similar to a CASE decision structure in a programming language. The transformation evaluates expressions, and based on the results, directs the data row to the specified stream.

Configuration

The **Split on** setting determines whether the row of data flows to the first matching stream or every stream it matches to.

Use the data flow expression builder to enter an expression for the split condition. To add a new condition, click on the plus icon in an existing row. A default stream can be added as well for rows that don't match any condition.



Data flow script

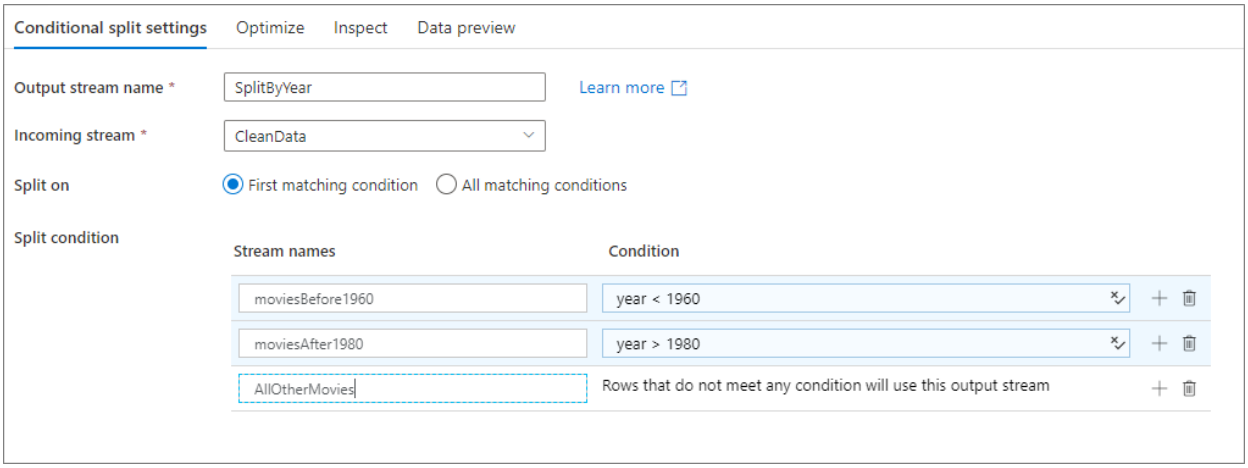
Syntax

```
<incomingStream>
    split(
        <conditionalExpression1>
        <conditionalExpression2>
        ...
        disjoint: {true | false}
    ) ~> <splitTx>@(stream1, stream2, ..., <defaultStream>)
```

Example

The below example is a conditional split transformation named `SplitByYear` that takes in incoming stream `CleanData`. This transformation has two split conditions `year < 1960` and `year > 1980`. `disjoint` is false because the data goes to the first matching condition rather than all matching conditions. Every row matching the first condition goes to output stream `moviesBefore1960`. All remaining rows matching the second condition go to output stream `moviesAFter1980`. All other rows flow through the default stream `AllOtherMovies`.

In the service UI, this transformation looks like the below image:



The data flow script for this transformation is in the snippet below:

```
CleanData
    split(
        year < 1960,
     year > 1980,
     disjoint: false
    ) ~> SplitByYear@(moviesBefore1960, moviesAfter1980, AllOtherMovies)
```

Next steps

Common data flow transformations used with conditional split are the join transformation, lookup transformation, and the select transformation

Derived column transformation in mapping data flow

8/19/2022 • 2 minutes to read • Edit Online

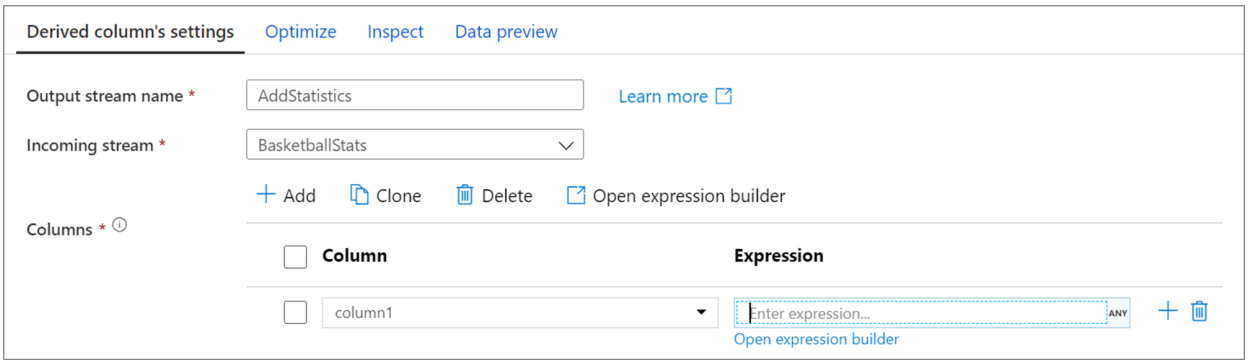
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

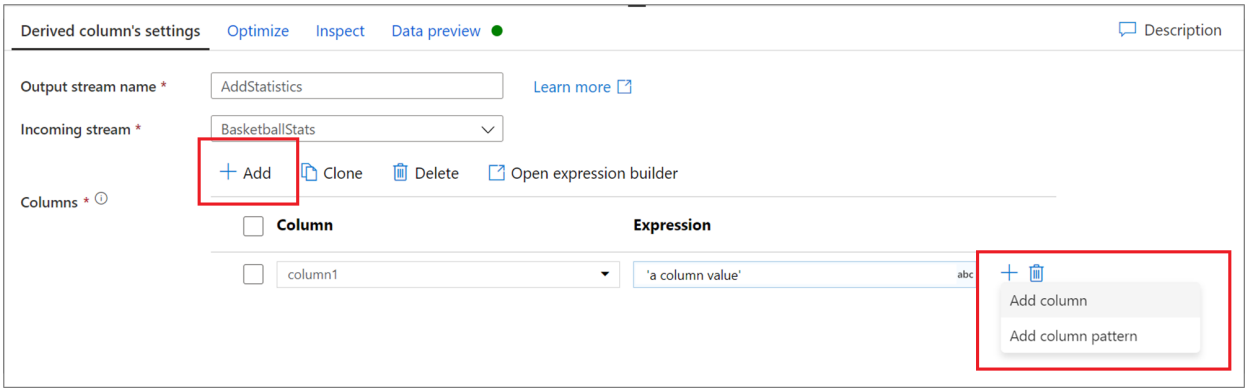
Use the derived column transformation to generate new columns in your data flow or to modify existing fields.

Create and update columns

When creating a derived column, you can either generate a new column or update an existing one. In the **Column** textbox, enter in the column you are creating. To override an existing column in your schema, you can use the column dropdown. To build the derived column's expression, click on the **Enter expression** textbox. You can either start typing your expression or open up the expression builder to construct your logic.

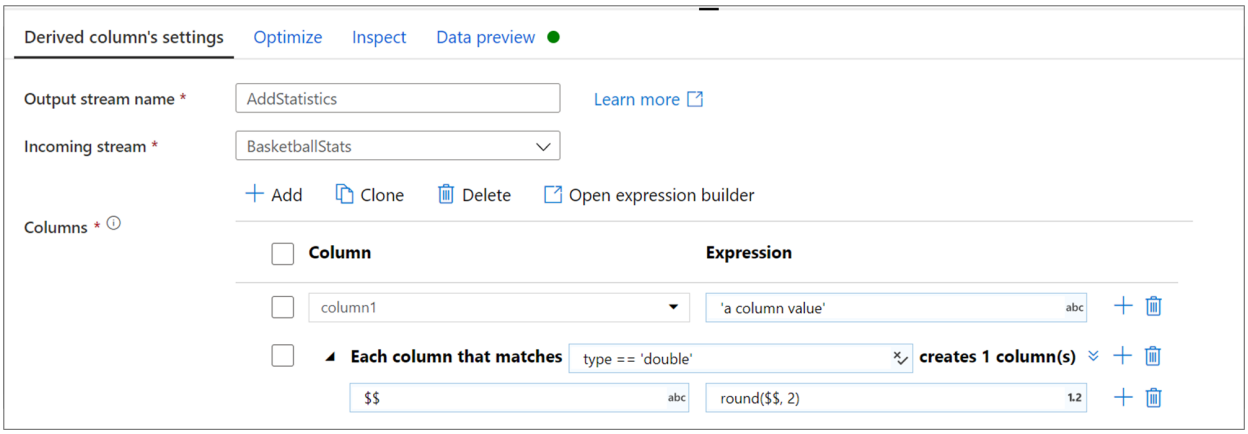


To add more derived columns, click on **Add** above the column list or the plus icon next to an existing derived column. Choose either **Add column** or **Add column pattern**.



Column patterns

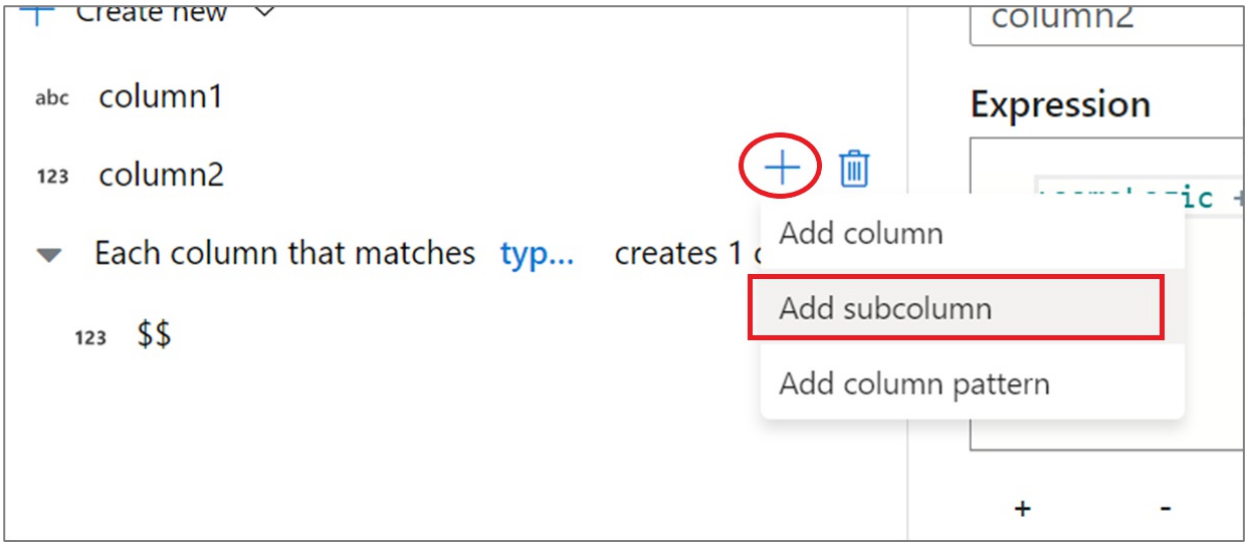
In cases where your schema is not explicitly defined or if you want to update a set of columns in bulk, you will want to create column patterns. Column patterns allow for you to match columns using rules based upon the column metadata and create derived columns for each matched column. For more information, learn how to build column patterns in the derived column transformation.



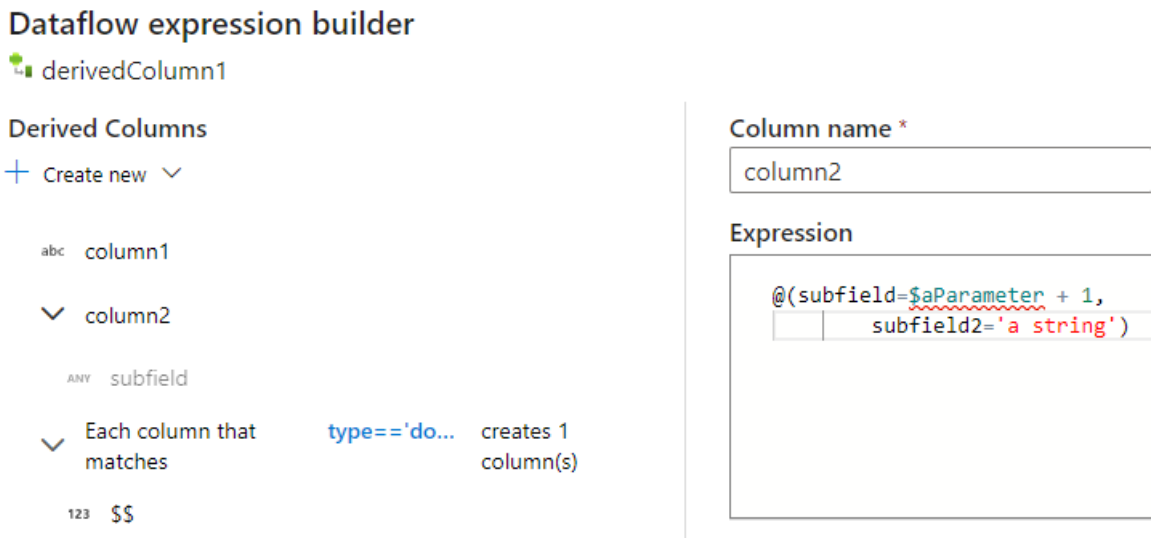
Building schemas using the expression builder

When using the mapping data flow expression builder, you can create, edit, and manage your derived columns in the **Derived Columns** section. All columns that are created or changed in the transformation are listed. Interactively choose which column or pattern you are editing by clicking on the column name. To add an additional column select **Create new** and choose whether you wish to add a single column or a pattern.

When working with complex columns, you can create subcolumns. To do this, click on the plus icon next to any column and select **Add subcolumn**. For more information on handling complex types in data flow, see JSON handling in mapping data flow.



For more information on handling complex types in data flow, see JSON handling in mapping data flow.



Data flow script

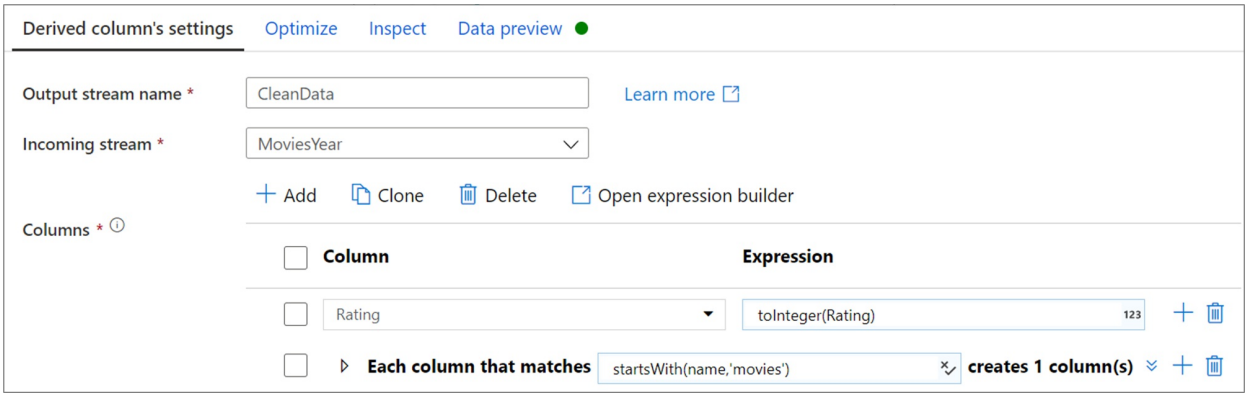
Syntax

```
<incomingStream>
    derive(
           <columnName1> = <expression1>,
           <columnName2> = <expression2>,
           each(
                match(matchExpression),
                <metadataColumn1> = <metadataExpression1>,
                <metadataColumn2> = <metadataExpression2>
               )
          ) ~> <deriveTransformationName>
```

Example

The below example is a derived column named `CleanData` that takes an incoming stream `MoviesYear` and creates two derived columns. The first derived column replaces column `Rating` with Rating's value as an integer type. The second derived column is a pattern that matches each column whose name starts with 'movies'. For each matched column, it creates a column `movie` that is equal to the value of the matched column prefixed with 'movie_'.

In the UI, this transformation looks like the below image:



The data flow script for this transformation is in the snippet below:

```
MoviesYear derive(
                Rating = toInteger(Rating),
          each(
                    match(startsWith(name,'movies')),
                    'movie' = 'movie_' + toString($$)
                )
            ) ~> CleanData
```

Next steps

- Learn more about the Mapping Data Flow expression language.

Exists transformation in mapping data flow

8/19/2022 • 2 minutes to read • Edit Online

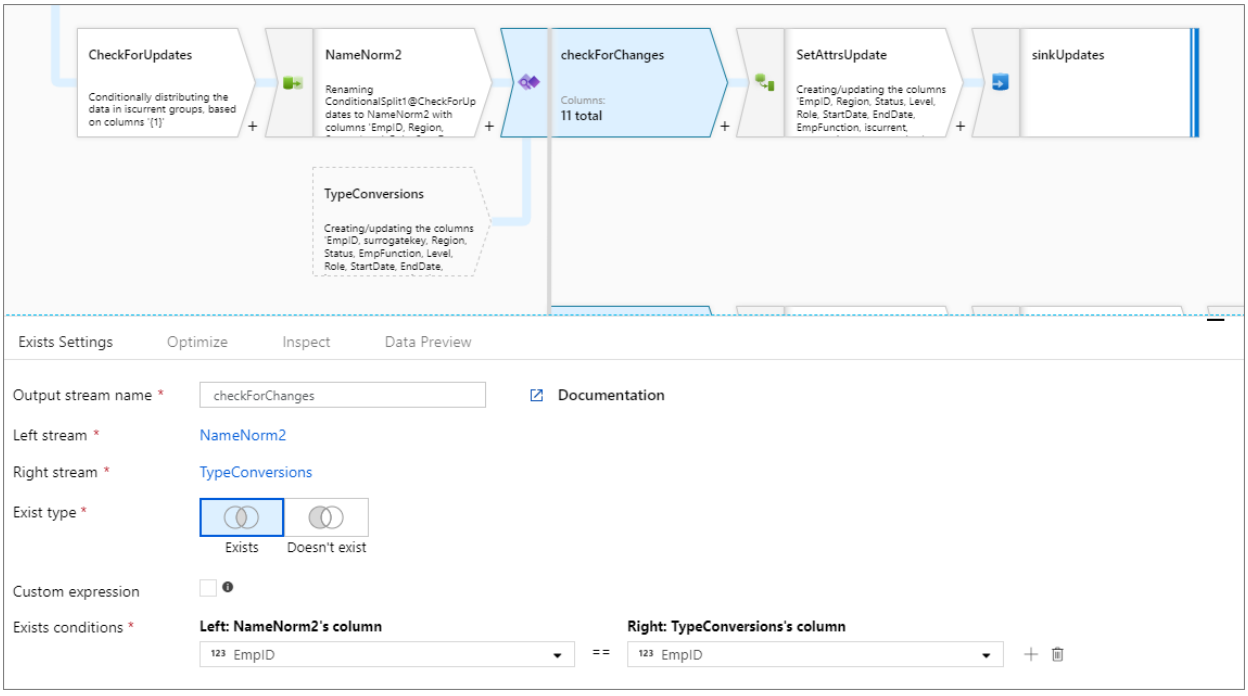
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

The exists transformation is a row filtering transformation that checks whether your data exists in another source or stream. The output stream includes all rows in the left stream that either exist or don't exist in the right stream. The exists transformation is similar to `SQL WHERE EXISTS` and `SQL WHERE NOT EXISTS`.

Configuration

1. Choose which data stream you're checking for existence in the **Right stream** dropdown.
2. Specify whether you're looking for the data to exist or not exist in the **Exist type** setting.
3. Select whether or not your want a **Custom expression**.
4. Choose which key columns you want to compare as your exists conditions. By default, data flow looks for equality between one column in each stream. To compare via a computed value, hover over the column dropdown and select **Computed column**.



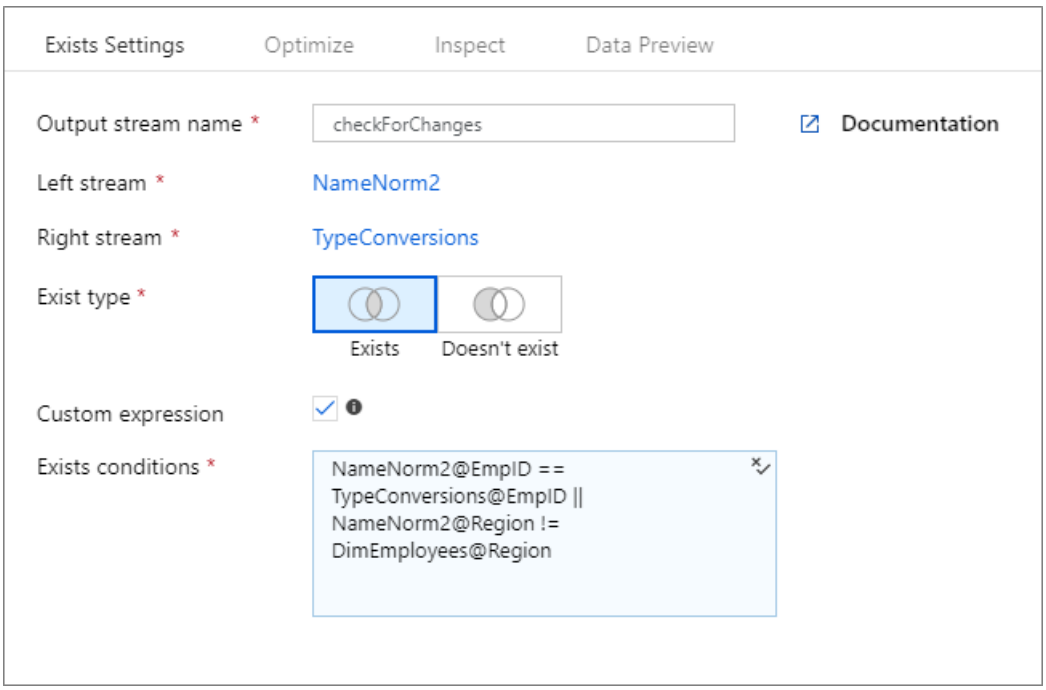
Multiple exists conditions

To compare multiple columns from each stream, add a new exists condition by clicking the plus icon next to an existing row. Each additional condition is joined by an "and" statement. Comparing two columns is the same as the following expression:

```
source1@column1 == source2@column1 && source1@column2 == source2@column2
```

Custom expression

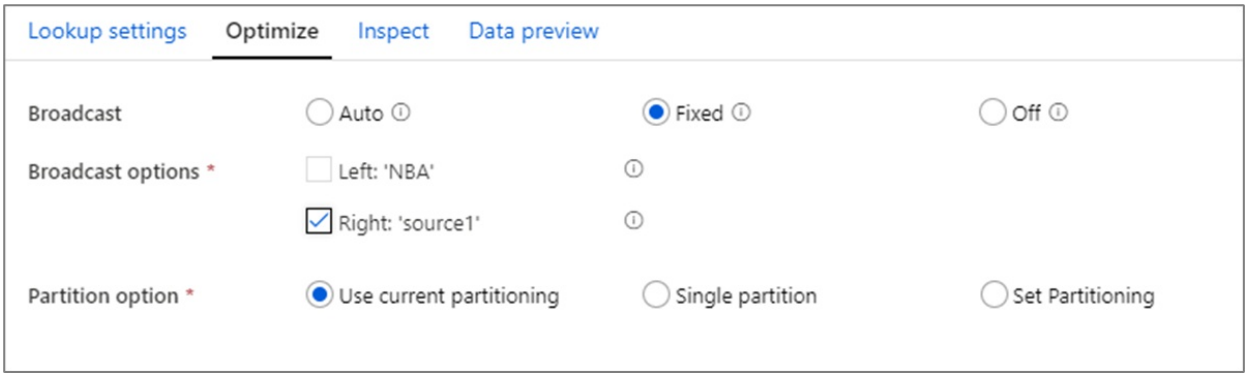
To create a free-form expression that contains operators other than "and" and "equals to", select the **Custom expression** field. Enter a custom expression via the data flow expression builder by clicking on the blue box.



If you are building dynamic patterns in your data flows by using "late binding" of columns via schema drift, you can use the `byName()` expression function to use the exists transformation without hardcoding (i.e. early binding) the column names. Example:

```
toString(byName('ProductNumber','source1')) == toString(byName('ProductNumber','source2'))
```

Broadcast optimization



In joins, lookups and exists transformation, if one or both data streams fit into worker node memory, you can optimize performance by enabling **Broadcasting**. By default, the spark engine will automatically decide whether or not to broadcast one side. To manually choose which side to broadcast, select **Fixed**.

It's not recommended to disable broadcasting via the **Off** option unless your joins are running into timeout errors.

Data flow script

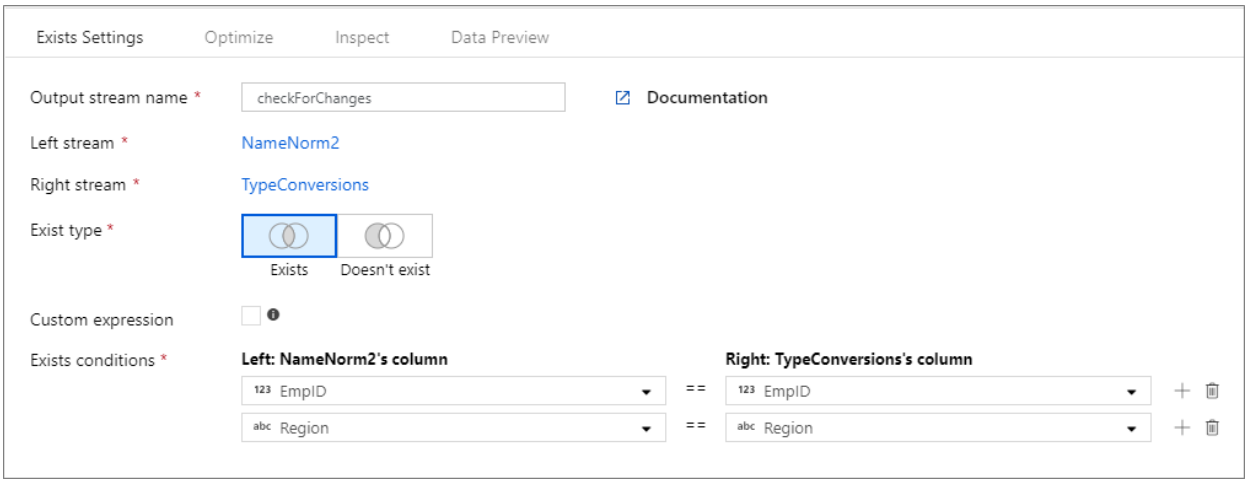
Syntax

```
<leftStream>, <rightStream>
    exists(
        <conditionalExpression>,
        negate: { true | false },
        broadcast: { 'auto' | 'left' | 'right' | 'both' | 'off' }
    ) ~> <existsTransformationName>
```

Example

The below example is an exists transformation named `checkForChanges` that takes left stream `NameNorm2` and right stream `TypeConversions`. The exists condition is the expression `NameNorm2@EmpID == TypeConversions@EmpID && NameNorm2@Region == DimEmployees@Region` that returns true if both the `EMPID` and `Region` columns in each stream matches. As we're checking for existence, `negate` is false. We aren't enabling any broadcasting in the optimize tab so `broadcast` has value `'none'`.

In the UI experience, this transformation looks like the below image:



The data flow script for this transformation is in the snippet below:

```
NameNorm2, TypeConversions
    exists(
        NameNorm2@EmpID == TypeConversions@EmpID && NameNorm2@Region == DimEmployees@Region,
     negate:false,
     broadcast: 'auto'
    ) ~> checkForChanges
```

Next steps

Similar transformations are Lookup and Join.

Filter transformation in mapping data flow

8/19/2022 • 2 minutes to read • Edit Online

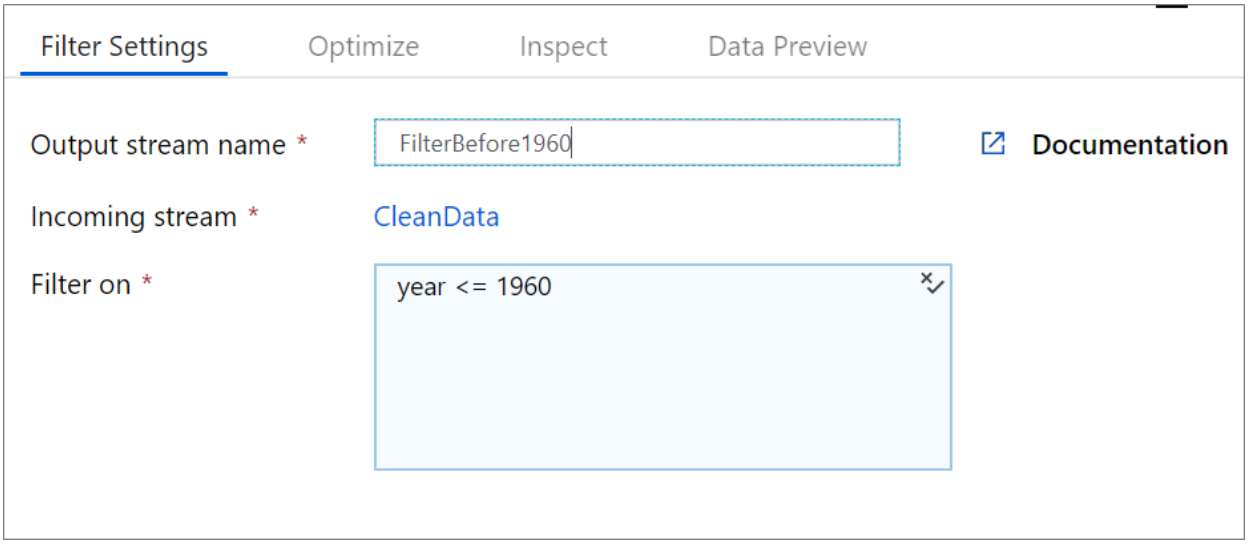
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

The Filter transforms allows row filtering based upon a condition. The output stream includes all rows that matching the filtering condition. The filter transformation is similar to a WHERE clause in SQL.

Configuration

Use the data flow expression builder to enter an expression for the filter condition. To open the expression builder, click on the blue box. The filter condition must be of type boolean. For more information on how to create an expression, see the expression builder documentation.



Data flow script

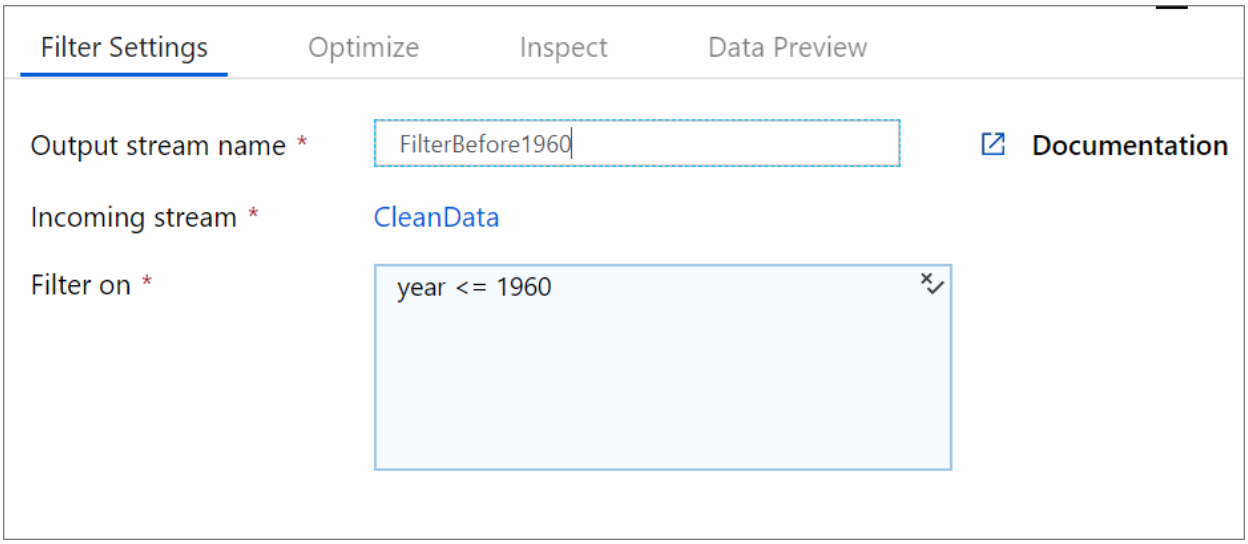
Syntax

```
<incomingStream>
    filter(
        <conditionalExpression>
    ) ~> <filterTransformationName>
```

Example

The below example is a filter transformation named `FilterBefore1960` that takes in incoming stream `CleanData`. The filter condition is the expression `year <= 1960`.

In the UI, this transformation looks like the below image:



The data flow script for this transformation is in the snippet below:

```
CleanData
    filter(
        year <= 1960
    ) ~> FilterBefore1960
```

Next steps

Filter out columns with the select transformation

Flatten transformation in mapping data flow

8/19/2022 • 3 minutes to read • Edit Online

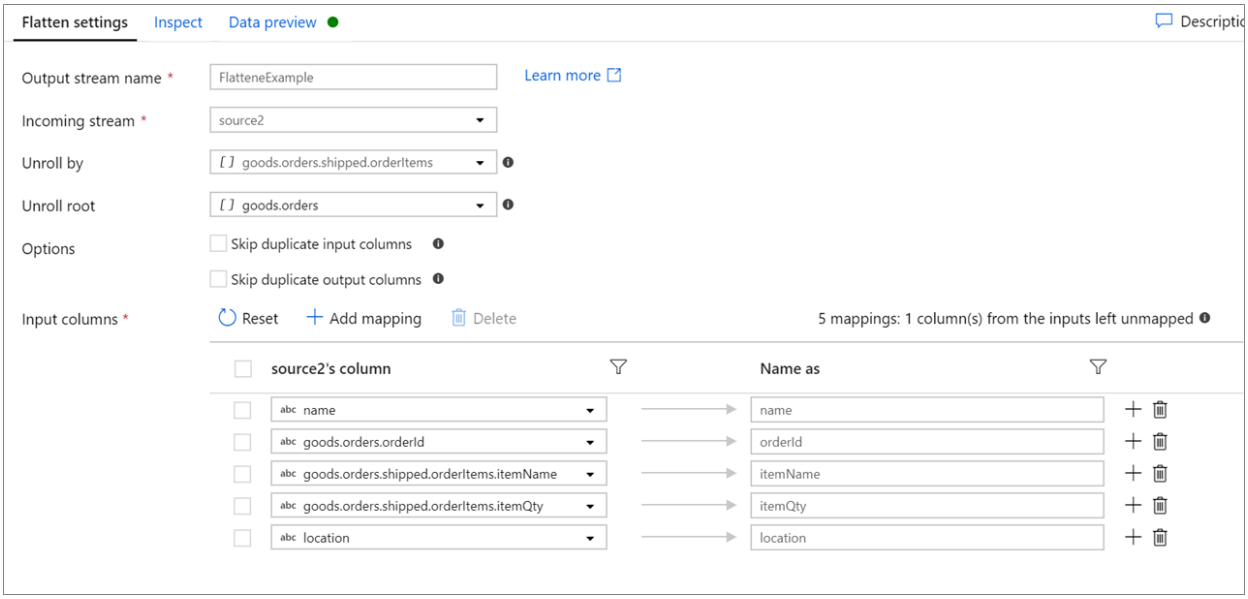
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

Use the flatten transformation to take array values inside hierarchical structures such as JSON and unroll them into individual rows. This process is known as denormalization.

Configuration

The flatten transformation contains the following configuration settings



Unroll by

Select an array to unroll. The output data will have one row per item in each array. If the unroll by array in the input row is null or empty, there will be one output row with unrolled values as null.

Unroll root

By default, the flatten transformation unrolls an array to the top of the hierarchy it exists in. You can optionally select an array as your unroll root. The unroll root must be an array of complex objects that either is or contains the unroll by array. If an unroll root is selected, the output data will contain at least one row per items in the unroll root. If the input row doesn't have any items in the unroll root, it will be dropped from the output data. Choosing an unroll root will always output a less than or equal number of rows than the default behavior.

Flatten mapping

Similar to the select transformation, choose the projection of the new structure from incoming fields and the denormalized array. If a denormalized array is mapped, the output column will be the same data type as the array. If the unroll by array is an array of complex objects that contains subarrays, mapping an item of that subarry will output an array.

Refer to the inspect tab and data preview to verify your mapping output.

Rule-based mapping

The flatten transformation supports rule-based mapping allowing you to create dynamic and flexible transformations that will flatten arrays based on rules and flatten structures based on hierarchy levels.

Matching condition

Enter a pattern matching condition for the column or columns that you wish to flatten using either exact matching or patterns. Example: `like(name,'cust%')`

Deep column traversal

Optional setting that tells the service to handle all subcolumns of a complex object individually instead of handling the complex object as a whole column.

Hierarchy level

Choose the level of the hierarchy that you would like expand.

Name matches (regex)

Optionally choose to express your name matching as a regular expression in this box, instead of using the matching condition above.

Examples

Refer to the following JSON object for the below examples of the flatten transformation

```
{
  "name":"MSFT","location":"Redmond", "satellites": ["Bay Area", "Shanghai"],
  "goods": {
    "trade":true, "customers":["government", "distributer", "retail"],
    "orders":[
        {"orderId":1,"orderTotal":123.34,"shipped":{"orderItems":[{"itemName":"Laptop","itemQty":20},
{"itemName":"Charger","itemQty":2}]}},
        {"orderId":2,"orderTotal":323.34,"shipped":{"orderItems":[{"itemName":"Mice","itemQty":2},
{"itemName":"Keyboard","itemQty":1}]}}
    ]}}
{"name":"Company1","location":"Seattle", "satellites": ["New York"],
  "goods":{"trade":false, "customers":["store1", "store2"],
  "orders":[
      {"orderId":4,"orderTotal":123.34,"shipped":{"orderItems":[{"itemName":"Laptop","itemQty":20},
{"itemName":"Charger","itemQty":3}]}},
      {"orderId":5,"orderTotal":343.24,"shipped":{"orderItems":[{"itemName":"Chair","itemQty":4},
{"itemName":"Lamp","itemQty":2}]}}
    ]}}
{"name": "Company2", "location": "Bellevue",
  "goods": {"trade": true, "customers":["Bank"], "orders": [{"orderId": 4, "orderTotal": 123.34}]}}
{"name": "Company3", "location": "Kirkland"}
```

No unroll root with string array

UNROLL BY	UNROLL ROOT	PROJECTION
goods.customers	None	name customer = goods.customer

Output

```
{ 'MSFT', 'government'}
{ 'MSFT', 'distributer'}
{ 'MSFT', 'retail'}
{ 'Company1', 'store'}
{ 'Company1', 'store2'}
{ 'Company2', 'Bank'}
{ 'Company3', null}
```

No unroll root with complex array

UNROLL BY	UNROLL ROOT	PROJECTION
goods.orders.shipped.orderItems	None	name orderId = goods.orders.orderId itemName = goods.orders.shipped.orderItems.itemName itemQty = goods.orders.shipped.orderItems.itemQty location = location

Output

```
{ 'MSFT', 1, 'Laptop', 20, 'Redmond'}
{ 'MSFT', 1, 'Charger', 2, 'Redmond'}
{ 'MSFT', 2, 'Mice', 2, 'Redmond'}
{ 'MSFT', 2, 'Keyboard', 1, 'Redmond'}
{ 'Company1', 4, 'Laptop', 20, 'Seattle'}
{ 'Company1', 4, 'Charger', 3, 'Seattle'}
{ 'Company1', 5, 'Chair', 4, 'Seattle'}
{ 'Company1', 5, 'Lamp', 2, 'Seattle'}
{ 'Company2', 4, null, null, 'Bellevue'}
{ 'Company3', null, null, null, 'Kirkland'}
```

Same root as unroll array

UNROLL BY	UNROLL ROOT	PROJECTION
goods.orders	goods.orders	name goods.orders.shipped.orderItems.itemName goods.customers location

Output

```
{ 'MSFT', ['Laptop','Charger'], ['government','distributer','retail'], 'Redmond'}
{ 'MSFT', ['Mice', 'Keyboard'], ['government','distributer','retail'], 'Redmond'}
{ 'Company1', ['Laptop','Charger'], ['store', 'store2'], 'Seattle'}
{ 'Company1', ['Chair', 'Lamp'], ['store', 'store2'], 'Seattle'}
{ 'Company2', null, ['Bank'], 'Bellevue'}
```

Unroll root with complex array

UNROLL BY	UNROLL ROOT	PROJECTION
goods.orders.shipped.orderItem	goods.orders	name orderId = goods.orders.orderId itemName = goods.orders.shipped.orderItems.itemName itemQty = goods.orders.shipped.orderItems.itemQty location = location

Output

```
{ 'MSFT', 1, 'Laptop', 20, 'Redmond'}
{ 'MSFT', 1, 'Charger', 2, 'Redmond'}
{ 'MSFT', 2, 'Mice', 2, 'Redmond'}
{ 'MSFT', 2, 'Keyboard', 1, 'Redmond'}
{ 'Company1', 4, 'Laptop', 20, 'Seattle'}
{ 'Company1', 4, 'Charger', 3, 'Seattle'}
{ 'Company1', 5, 'Chair', 4, 'Seattle'}
{ 'Company1', 5, 'Lamp', 2, 'Seattle'}
{ 'Company2', 4, null, null, 'Bellevue'}
```

Data flow script

Syntax

```
<incomingStream>
foldDown(unroll(<unroll cols>),
    mapColumn(
        name,
        each(<array>(type == '<arrayDataType>')),
        each(<array>, match(true())),
        location
    )) ~> <transformationName>
```

Example

```
source foldDown(unroll(goods.orders.shipped.orderItems, goods.orders),
    mapColumn(
        name,
        orderId = goods.orders.orderId,
        itemName = goods.orders.shipped.orderItems.itemName,
        itemQty = goods.orders.shipped.orderItems.itemQty,
        location = location
    ),
    skipDuplicateMapInputs: false,
    skipDuplicateMapOutputs: false)
```

Next steps

- Use the Pivot transformation to pivot rows to columns.
- Use the Unpivot transformation to pivot columns to rows.

Join transformation in mapping data flow

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

Use the join transformation to combine data from two sources or streams in a mapping data flow. The output stream will include all columns from both sources matched based on a join condition.

Join types

Mapping data flows currently supports five different join types.

Inner Join

Inner join only outputs rows that have matching values in both tables.

Left Outer

Left outer join returns all rows from the left stream and matched records from the right stream. If a row from the left stream has no match, the output columns from the right stream are set to NULL. The output will be the rows returned by an inner join plus the unmatched rows from the left stream.

> **NOTE**
> The Spark engine used by data flows will occasionally fail due to possible cartesian products in your join conditions. If this occurs, you can switch to a custom cross join and manually enter your join condition. This may result in slower performance in your data flows as the execution engine may need to calculate all rows from both sides of the relationship and then filter rows.

Right Outer

Right outer join returns all rows from the right stream and matched records from the left stream. If a row from the right stream has no match, the output columns from the left stream are set to NULL. The output will be the rows returned by an inner join plus the unmatched rows from the right stream.

Full Outer

Full outer join outputs all columns and rows from both sides with NULL values for columns that aren't matched.

Custom cross join

Cross join outputs the cross product of the two streams based upon a condition. If you're using a condition that isn't equality, specify a custom expression as your cross join condition. The output stream will be all rows that meet the join condition.

You can use this join type for non-equi joins and `OR` conditions.

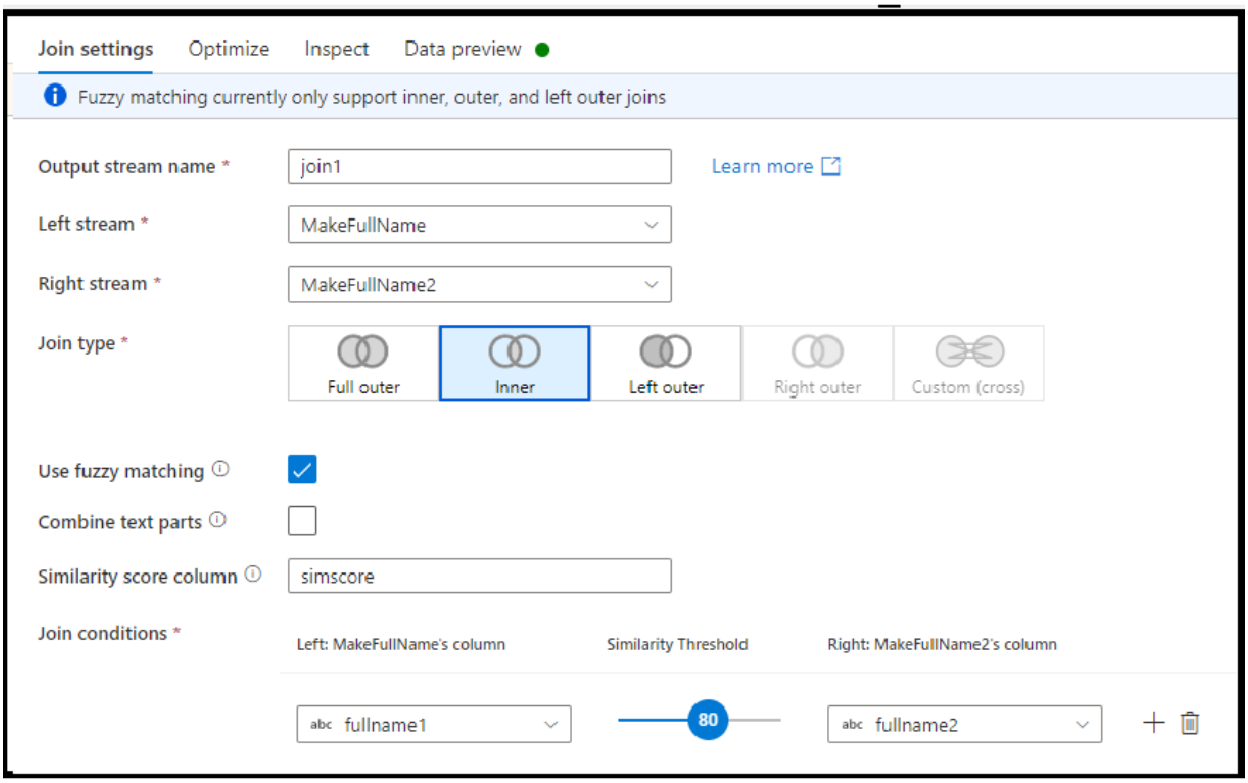
If you would like to explicitly produce a full cartesian product, use the Derived Column transformation in each of the two independent streams before the join to create a synthetic key to match on. For example, create a new column in Derived Column in each stream called `SyntheticKey` and set it equal to `1`. Then use `a.SyntheticKey == b.SyntheticKey` as your custom join expression.

> **NOTE**
> Make sure to include at least one column from each side of your left and right relationship in a custom cross join. Executing cross joins with static values instead of columns from each side results in full scans of the entire dataset, causing your data flow to perform poorly.

Fuzzy join

You can choose to join based on fuzzy join logic instead of exact column value matching by turning on the "Use fuzzy matching" checkbox option.

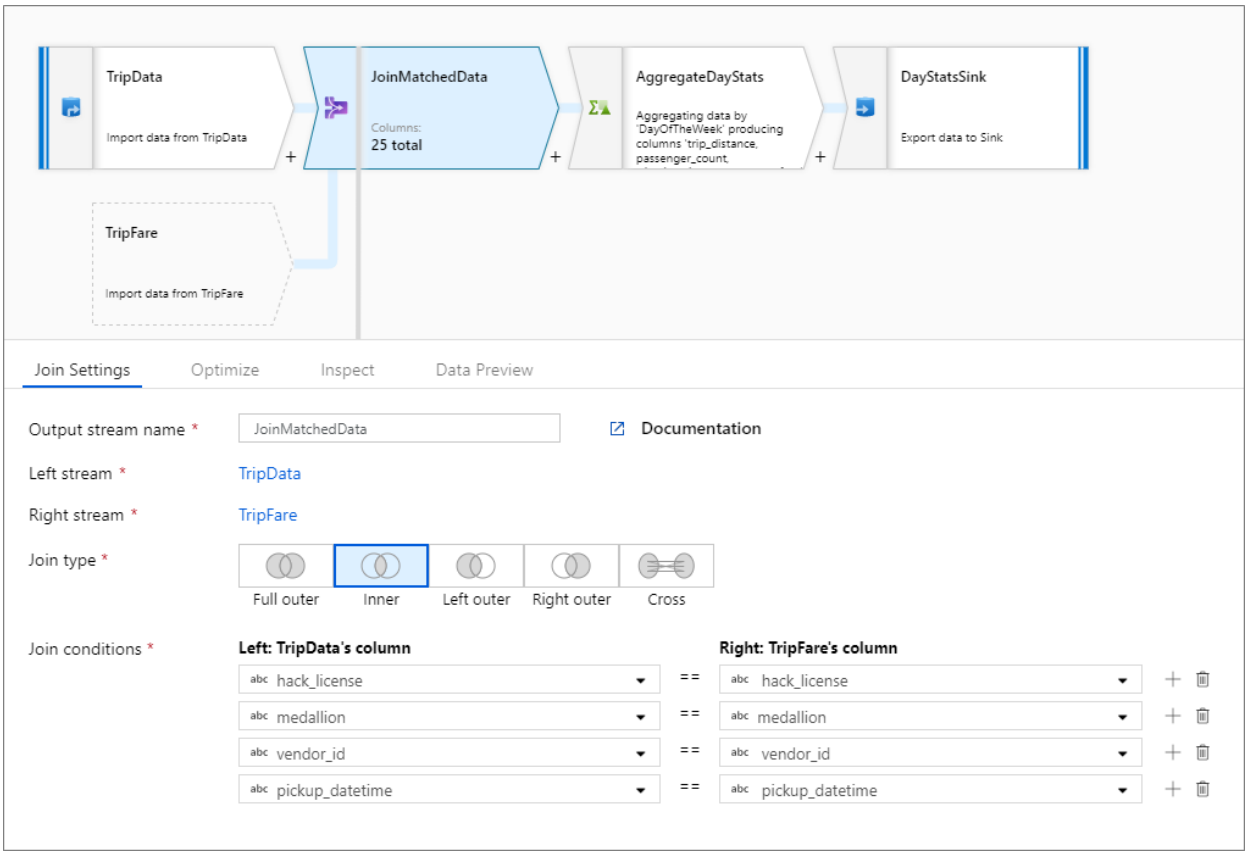
- Combine text parts: Use this option to find matches by remove space between words. For example, Data Factory is matched with DataFactory if this option is enabled.
- Similarity score column: You can optionally choose to store the matching score for each row in a column by entering a new column name here to store that value.
- Similarity threshold: Choose a value between 60 and 100 as a percentage match between values in the columns you've selected.



> **NOTE**
> Fuzzy matching currently works only with string column types and with inner, left outer, and full outer join types. You must turn off the broadcast optimization when using fuzzing matching joins.

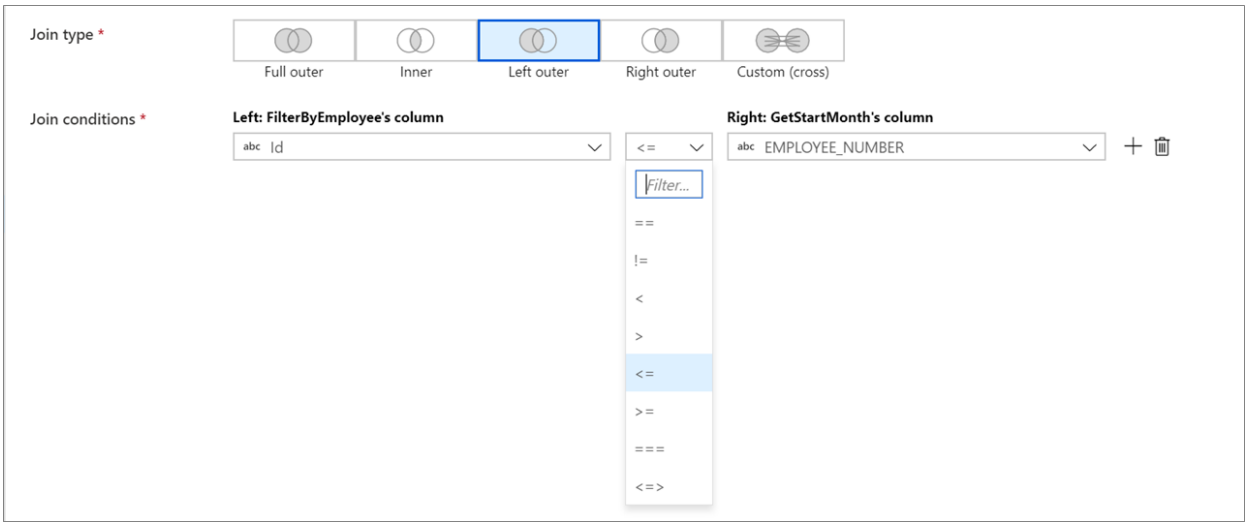
Configuration

1. Choose which data stream you're joining with in the **Right stream** dropdown.
2. Select your **Join type**
3. Choose which key columns you want to match on for your join condition. By default, data flow looks for equality between one column in each stream. To compare via a computed value, hover over the column dropdown and select **Computed column**.



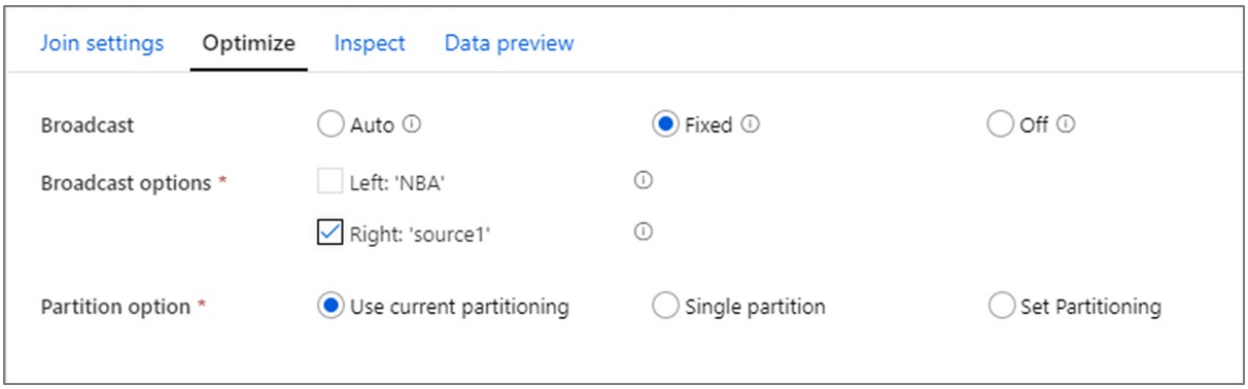
Non-equi joins

To use a conditional operator such as not equals (!=) or greater than (>) in your join conditions, change the operator dropdown between the two columns. Non-equi joins require at least one of the two streams to be broadcasted using **Fixed** broadcasting in the **Optimize** tab.



Optimizing join performance

Unlike merge join in tools like SSIS, the join transformation isn't a mandatory merge join operation. The join keys don't require sorting. The join operation occurs based on the optimal join operation in Spark, either broadcast or map-side join.

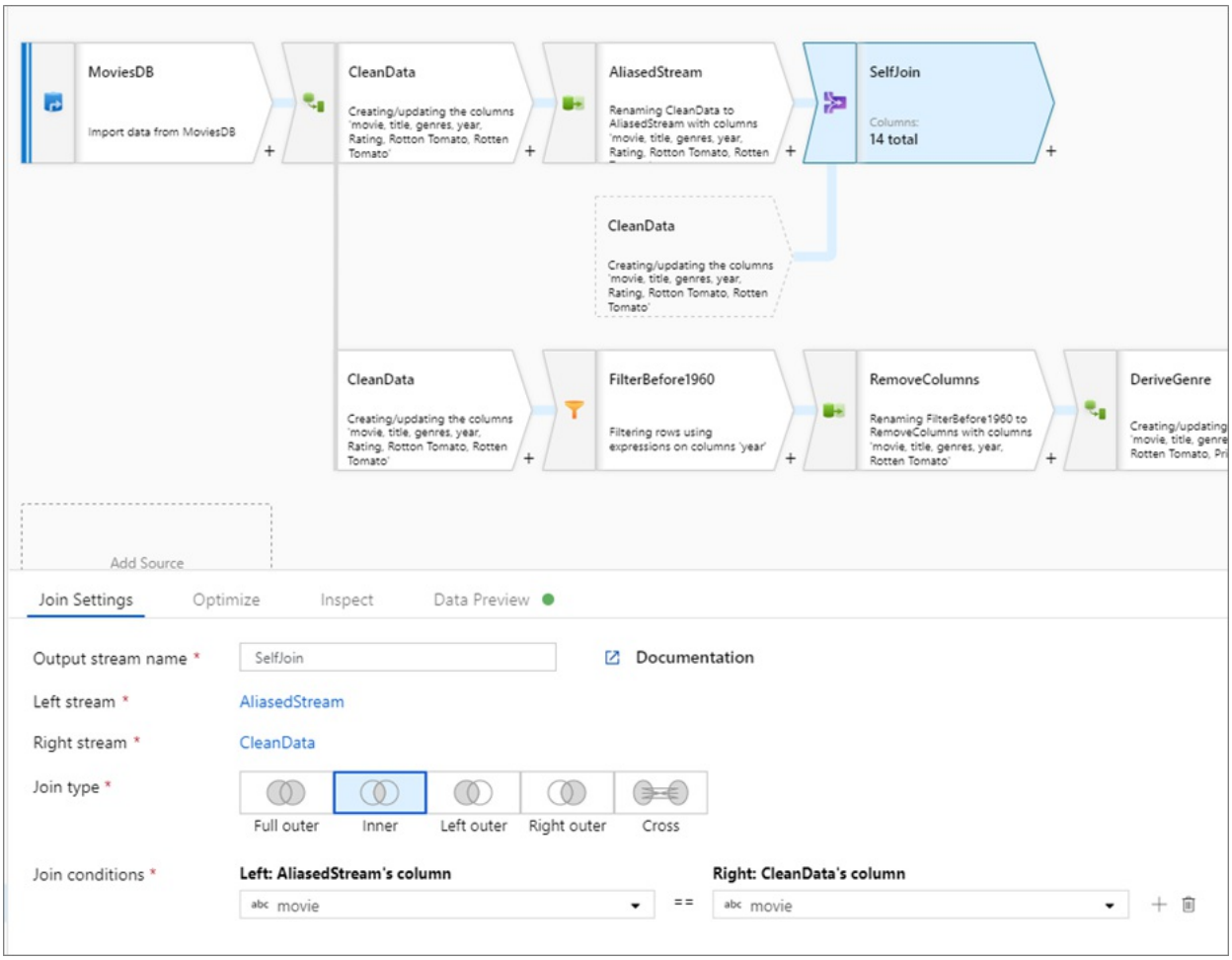


In joins, lookups and exists transformation, if one or both data streams fit into worker node memory, you can optimize performance by enabling **Broadcasting**. By default, the spark engine will automatically decide whether or not to broadcast one side. To manually choose which side to broadcast, select **Fixed**.

It's not recommended to disable broadcasting via the **Off** option unless your joins are running into timeout errors.

Self-Join

To self-join a data stream with itself, alias an existing stream with a select transformation. Create a new branch by clicking on the plus icon next to a transformation and selecting **New branch**. Add a select transformation to alias the original stream. Add a join transformation and choose the original stream as the **Left stream** and the select transformation as the **Right stream**.



Testing join conditions

When testing the join transformations with data preview in debug mode, use a small set of known data. When sampling rows from a large dataset, you can't predict which rows and keys will be read for testing. The result is

non-deterministic, meaning that your join conditions may not return any matches.

Data flow script

Syntax

```
<leftStream>, <rightStream>
    join(
        <conditionalExpression>,
        joinType: { 'inner'> | 'outer' | 'left_outer' | 'right_outer' | 'cross' }
        broadcast: { 'auto' | 'left' | 'right' | 'both' | 'off' }
    ) ~> <joinTransformationName>
```

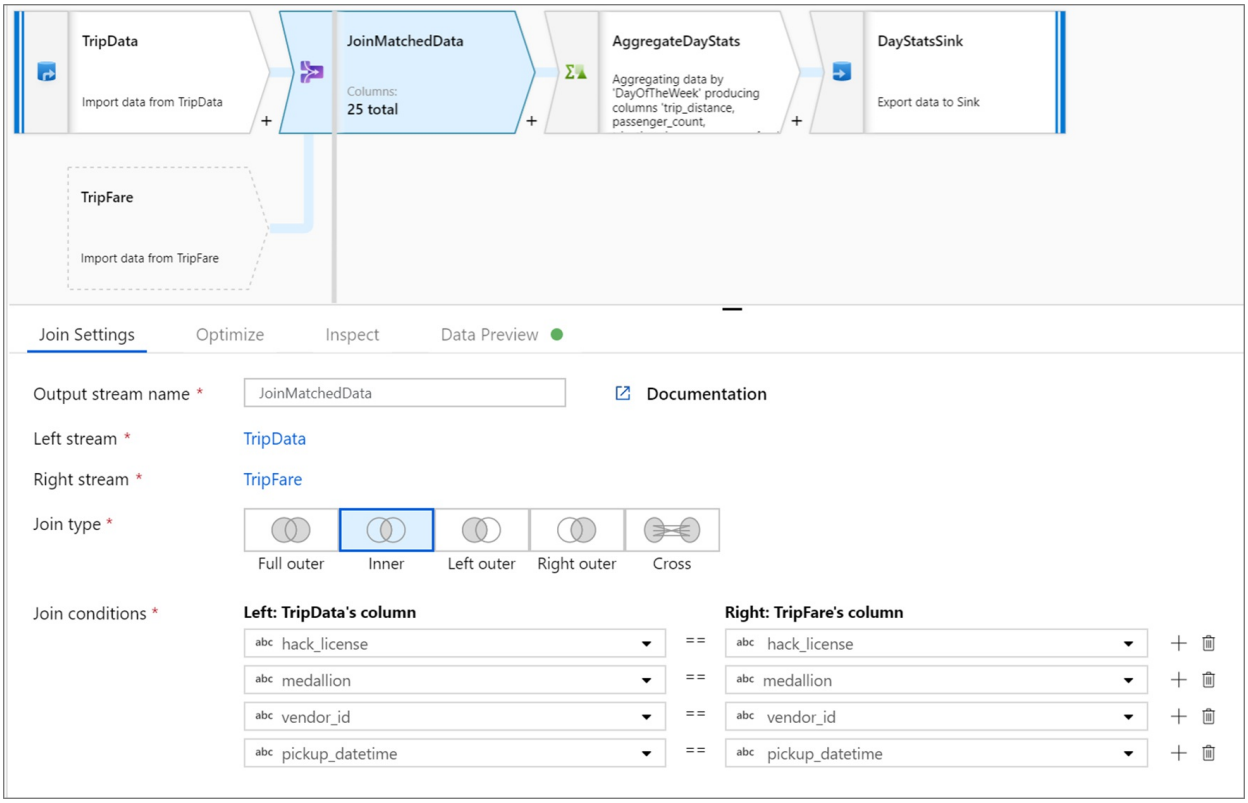
Inner join example

The below example is a join transformation named `JoinMatchedData` that takes left stream `TripData` and right stream `TripFare`. The join condition is the expression

```
hack_license == { hack_license} && TripData@medallion == TripFare@medallion && vendor_id == { vendor_id} &&
pickup_datetime == { pickup_datetime}
```

that returns true if the `hack_license`, `medallion`, `vendor_id`, and `pickup_datetime` columns in each stream match. The `joinType` is `'inner'`. We're enabling broadcasting in only the left stream so `broadcast` has value `'left'`.

In the UI, this transformation looks like the below image:



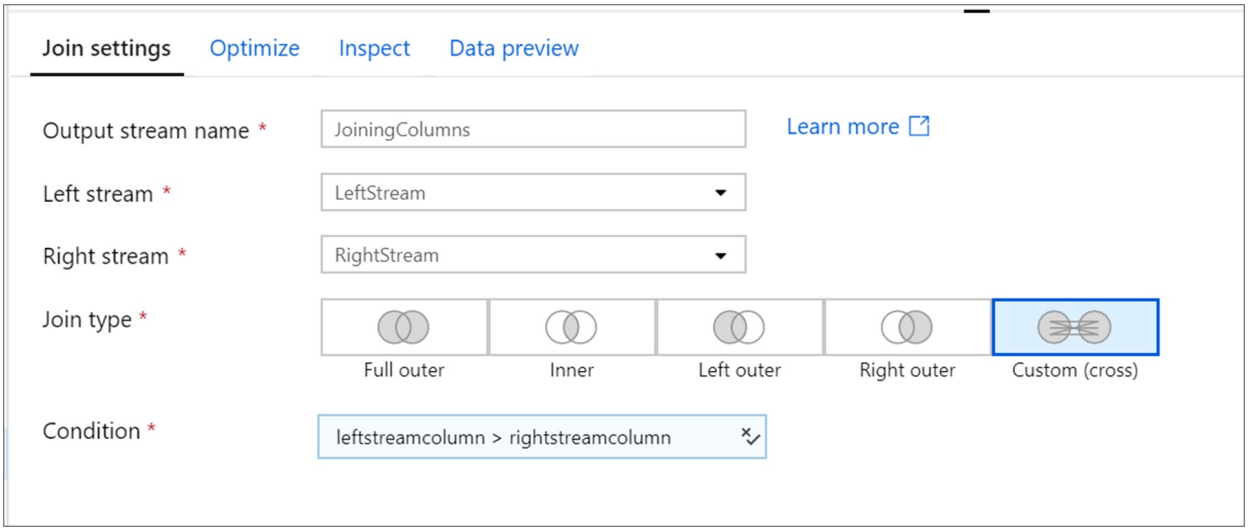
The data flow script for this transformation is in the snippet below:

```
TripData, TripFare
    join(
        hack_license == { hack_license}
        && TripData@medallion == TripFare@medallion
        && vendor_id == { vendor_id}
        && pickup_datetime == { pickup_datetime},
        joinType:'inner',
        broadcast: 'left'
    )~> JoinMatchedData
```

Custom cross join example

The below example is a join transformation named `JoiningColumns` that takes left stream `LeftStream` and right stream `RightStream`. This transformation takes in two streams and joins together all rows where column `leftstreamcolumn` is greater than column `rightstreamcolumn`. The `joinType` is `cross`. Broadcasting is not enabled `broadcast` has value `'none'`.

In the UI, this transformation looks like the below image:



The data flow script for this transformation is in the snippet below:

```
LeftStream, RightStream
    join(
        leftstreamcolumn > rightstreamcolumn,
        joinType:'cross',
        broadcast: 'none'
    )~> JoiningColumns
```

Next steps

After joining data, create a derived column and sink your data to a destination data store.

Lookup transformations in mapping data flow

8/19/2022 • 3 minutes to read • Edit Online

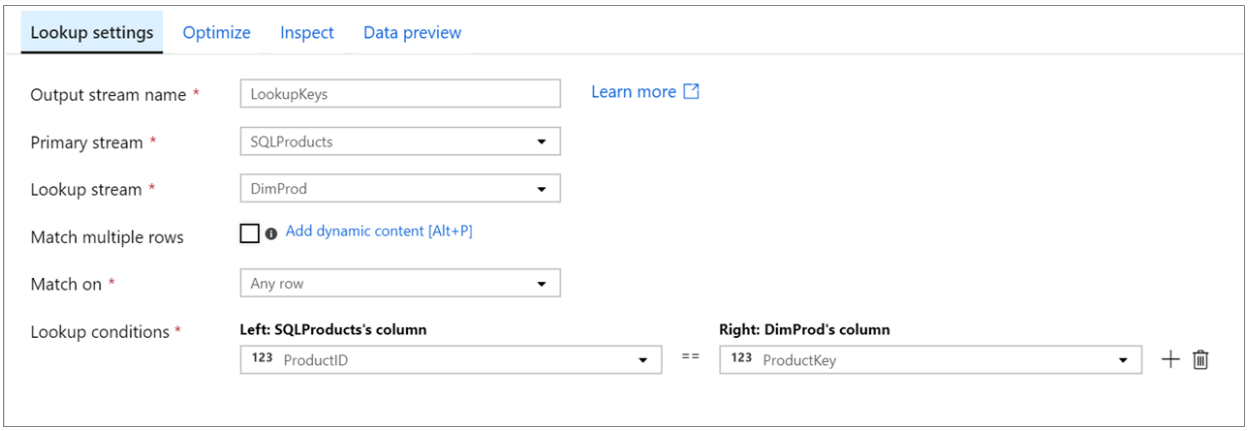
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

Use the lookup transformation to reference data from another source in a data flow stream. The lookup transformation appends columns from matched data to your source data.

A lookup transformation is similar to a left outer join. All rows from the primary stream will exist in the output stream with additional columns from the lookup stream.

Configuration



Primary stream: The incoming stream of data. This stream is equivalent to the left side of a join.

Lookup stream: The data that is appended to the primary stream. Which data is added is determined by the lookup conditions. This stream is equivalent to the right side of a join.

Match multiple rows: If enabled, a row with multiple matches in the primary stream will return multiple rows. Otherwise, only a single row will be returned based upon the 'Match on' condition.

Match on: Only visible if 'Match multiple rows' isn't selected. Choose whether to match on any row, the first match, or the last match. Any row is recommended as it executes the fastest. If first row or last row is selected, you'll be required to specify sort conditions.

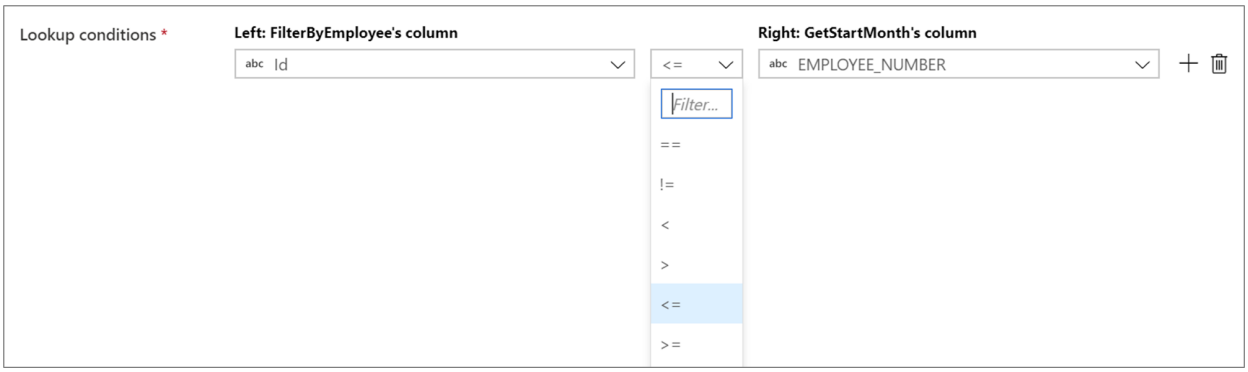
Lookup conditions: Choose which columns to match on. If the equality condition is met, then the rows will be considered a match. Hover and select 'Computed column' to extract a value using the data flow expression language.

All columns from both streams are included in the output data. To drop duplicate or unwanted columns, add a select transformation after your lookup transformation. Columns can also be dropped or renamed in a sink transformation.

Non-equi joins

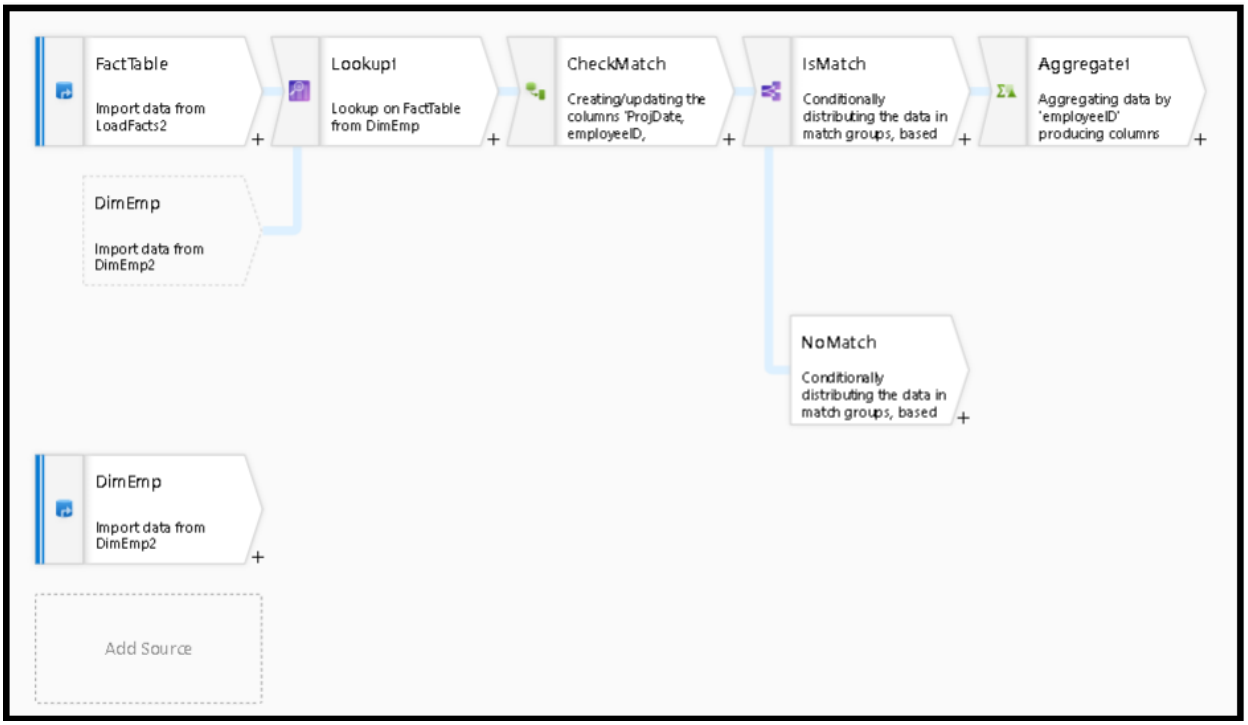
To use a conditional operator such as not equals (!=) or greater than (>) in your lookup conditions, change the

operator dropdown between the two columns. Non-equi joins require at least one of the two streams to be broadcasted using **Fixed** broadcasting in the **Optimize** tab.



Analyzing matched rows

After your lookup transformation, the function `isMatch()` can be used to see if the lookup matched for individual rows.

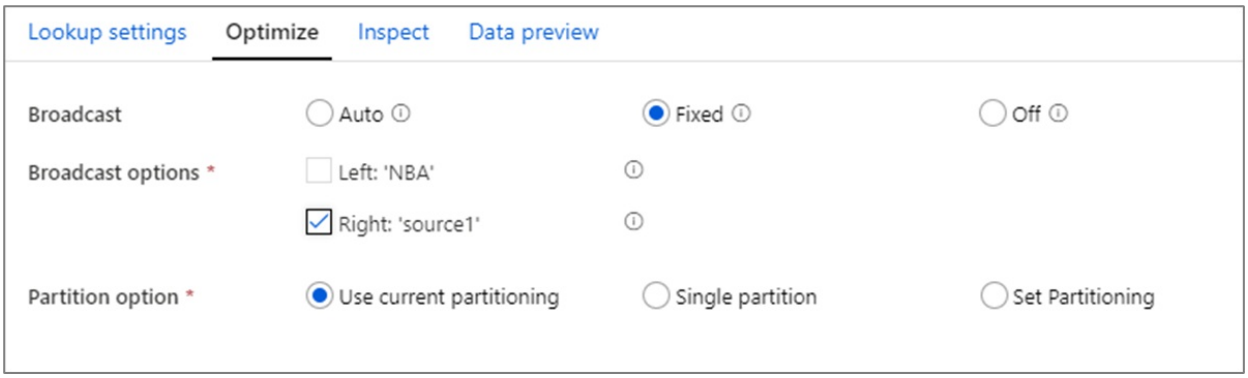


An example of this pattern is using the conditional split transformation to split on the `isMatch()` function. In the example above, matching rows go through the top stream and non-matching rows flow through the `NoMatch` stream.

Testing lookup conditions

When testing the lookup transformation with data preview in debug mode, use a small set of known data. When sampling rows from a large dataset, you can't predict which rows and keys will be read for testing. The result is non-deterministic, meaning that your join conditions may not return any matches.

Broadcast optimization



In joins, lookups and exists transformation, if one or both data streams fit into worker node memory, you can optimize performance by enabling **Broadcasting**. By default, the spark engine will automatically decide whether or not to broadcast one side. To manually choose which side to broadcast, select **Fixed**.

It's not recommended to disable broadcasting via the **Off** option unless your joins are running into timeout errors.

Cached lookup

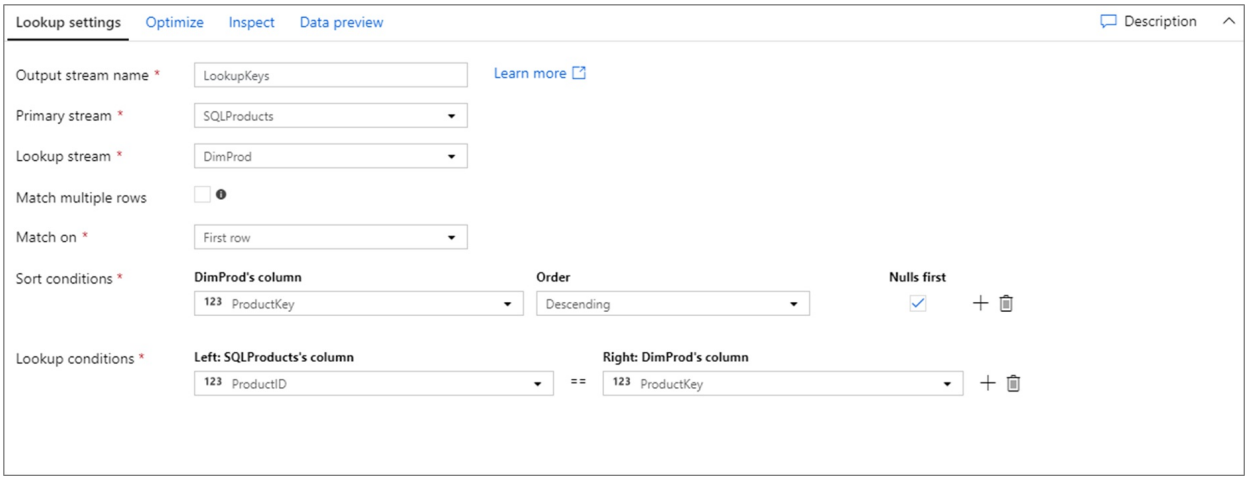
If you're doing multiple smaller lookups on the same source, a cached sink and lookup maybe a better use case than the lookup transformation. Common examples where a cache sink may be better are looking up a max value on a data store and matching error codes to an error message database. For more information, learn about cache sinks and cached lookups.

Data flow script

Syntax

```
<leftStream>, <rightStream>
    lookup(
        <lookupConditionExpression>,
        multiple: { true | false },
        pickup: { 'first' | 'last' | 'any' },  ## Only required if false is selected for multiple
        { desc | asc }( <sortColumn>, { true | false }), ## Only required if 'first' or 'last' is selected.
true/false determines whether to put nulls first
        broadcast: { 'auto' | 'left' | 'right' | 'both' | 'off' }
    ) ~> <lookupTransformationName>
```

Example



The data flow script for the above lookup configuration is in the code snippet below.

```
SQLProducts, DimProd lookup(ProductID == ProductKey,
    multiple: false,
    pickup: 'first',
    asc(ProductKey, true),
    broadcast: 'auto')~> LookupKeys
```

Next steps

- The join and exists transformations both take in multiple stream inputs
- Use a conditional split transformation with `isMatch()` to split rows on matching and non-matching values

Creating a new branch in mapping data flow

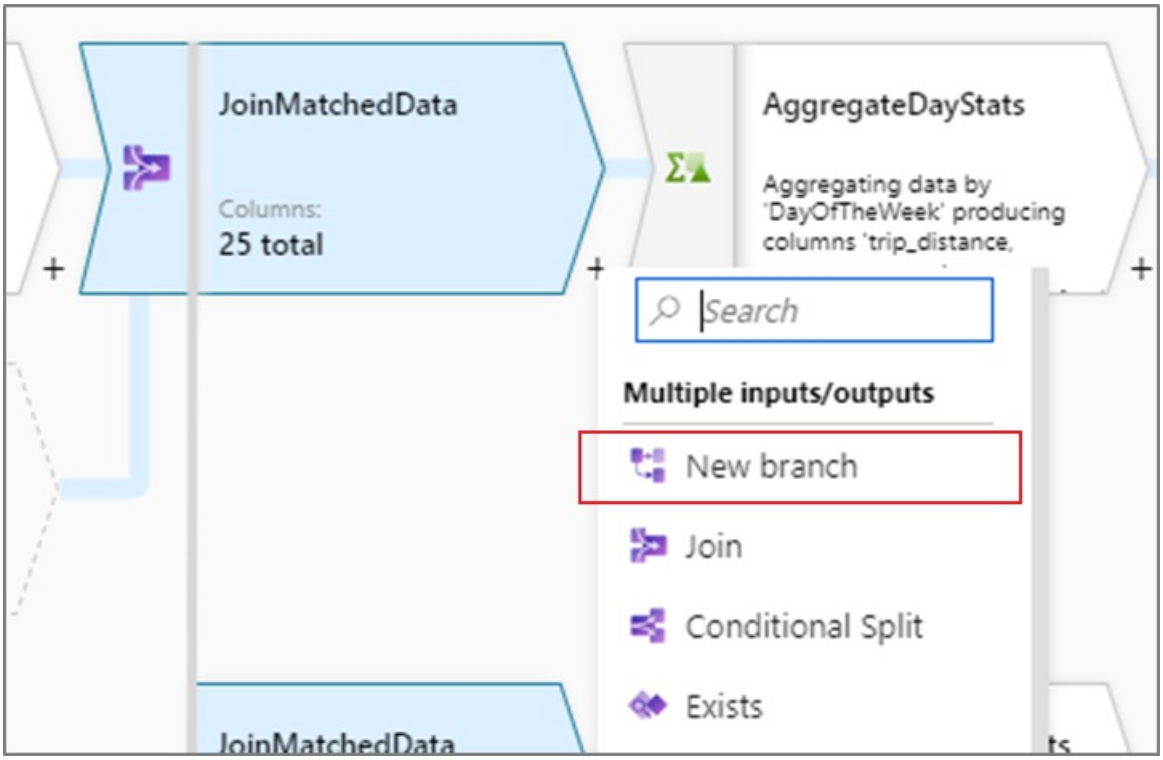
8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

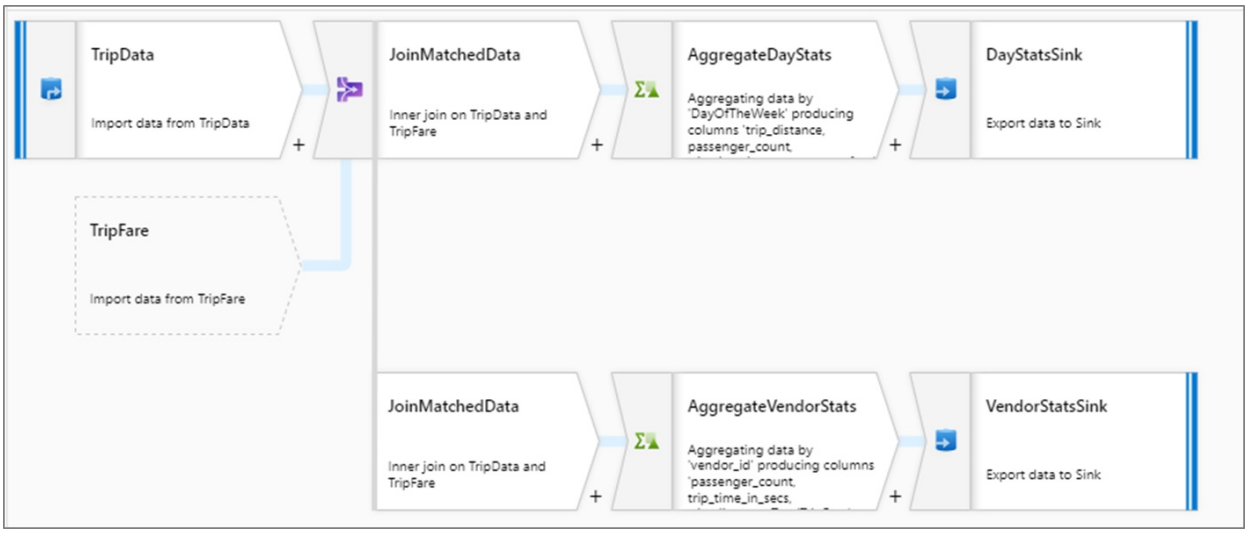
Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

Add a new branch to do multiple sets of operations and transformations against the same data stream. Adding a new branch is useful when you want to use the same source to for multiple sinks or for self-joining data together.

A new branch can be added from the transformation list similar to other transformations. **New Branch** will only be available as an action when there's an existing transformation following the transformation you're attempting to branch.



In the below example, the data flow is reading taxi trip data. Output aggregated by both day and vendor is required. Instead of creating two separate data flows that read from the same source, a new branch can be added. This way both aggregations can be executed as part of the same data flow.



NOTE

When clicking the plus (+) to add transformations to your graph, you will only see the New Branch option when there are subsequent transformation blocks. This is because New Branch creates a reference to the existing stream and requires further upstream processing to operate on. If you do not see the New Branch option, add a Derived Column or other transformation first, then return to the previous block and you will see New Branch as an option.

Next steps

After branching, you may want to use the data flow transformations

Pivot transformation in mapping data flow

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

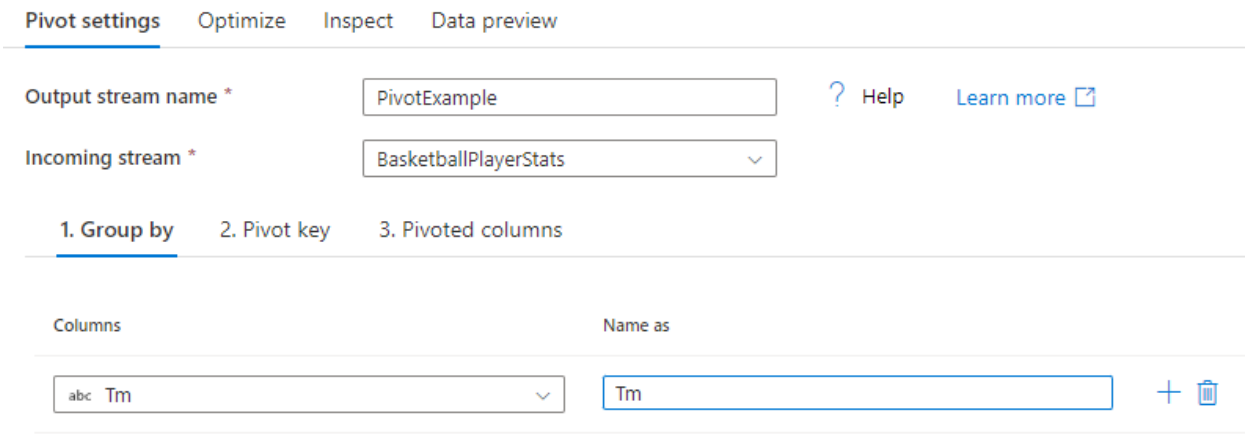
Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

Use the pivot transformation to create multiple columns from the unique row values of a single column. Pivot is an aggregation transformation where you select group by columns and generate pivot columns using aggregate functions.

Configuration

The pivot transformation requires three different inputs: group by columns, the pivot key, and how to generate the pivoted columns

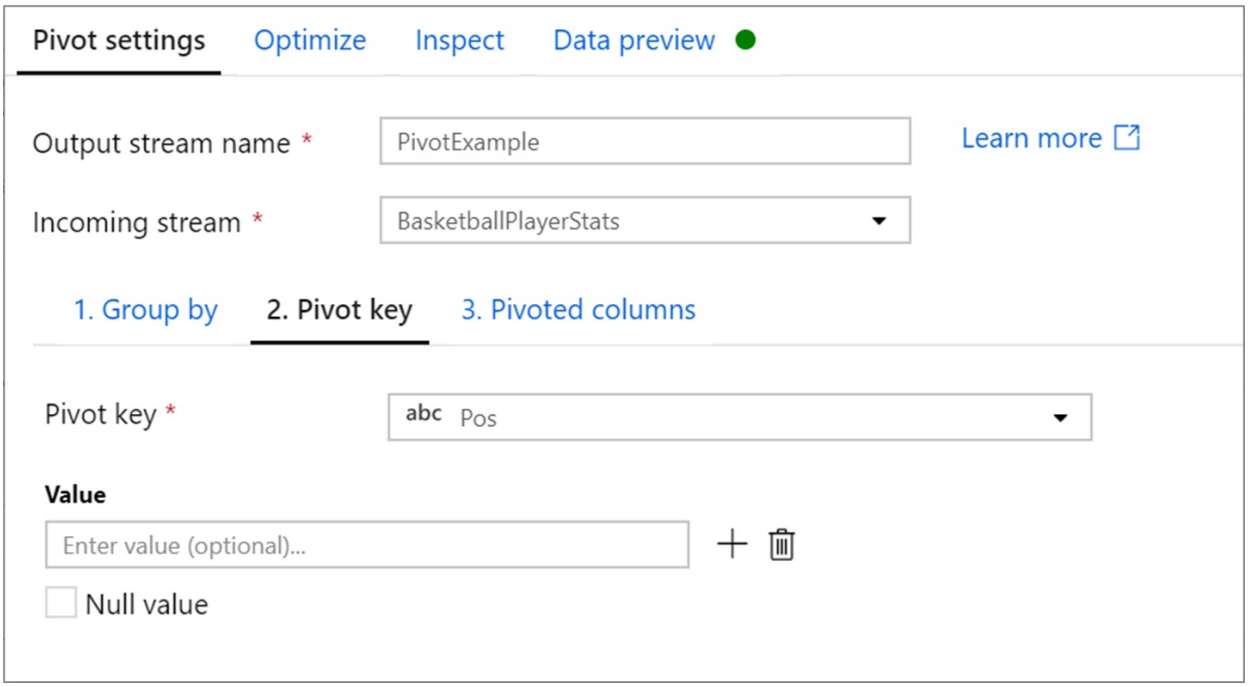
Group by



Select which columns to aggregate the pivoted columns over. The output data will group all rows with the same group by values into one row. The aggregation done in the pivoted column will occur over each group.

This section is optional. If no group by columns are selected, the entire data stream will be aggregated and only one row will be outputted.

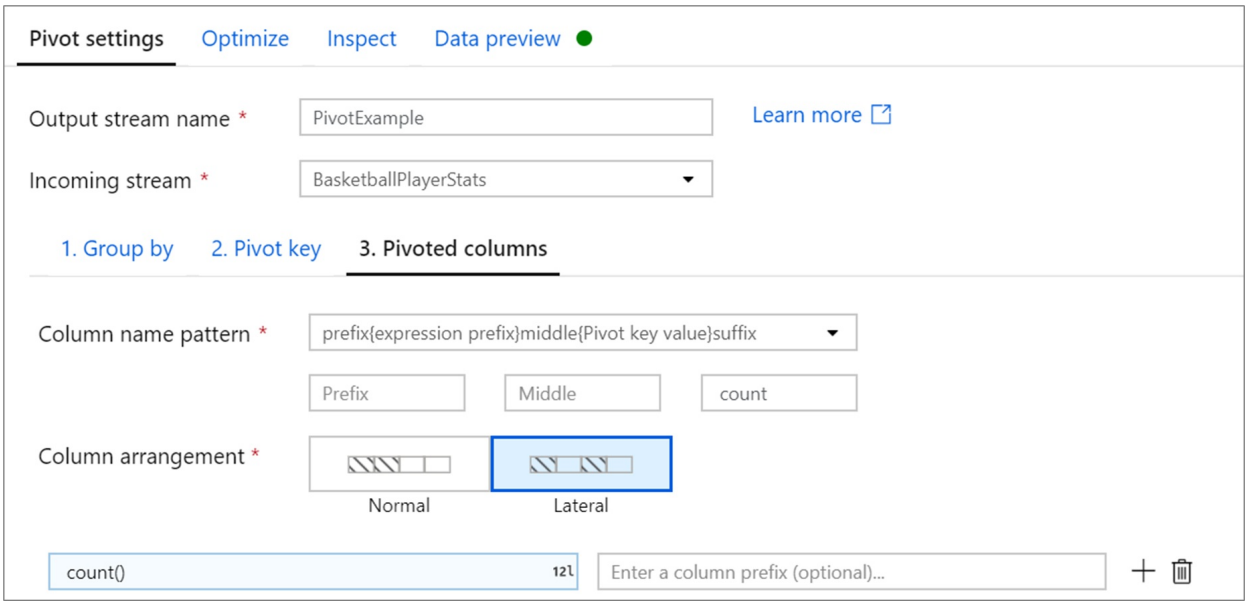
Pivot key



The pivot key is the column whose row values get pivoted into new columns. By default, the pivot transformation will create a new column for each unique row value.

In the section labeled **Value**, you can enter specific row values to be pivoted. Only the row values entered in this section will be pivoted. Enabling **Null value** will create a pivoted column for the null values in the column.

Pivoted columns



For each unique pivot key value that becomes a column, generate an aggregated row value for each group. You can create multiple columns per pivot key. Each pivot column must contain at least one aggregate function.

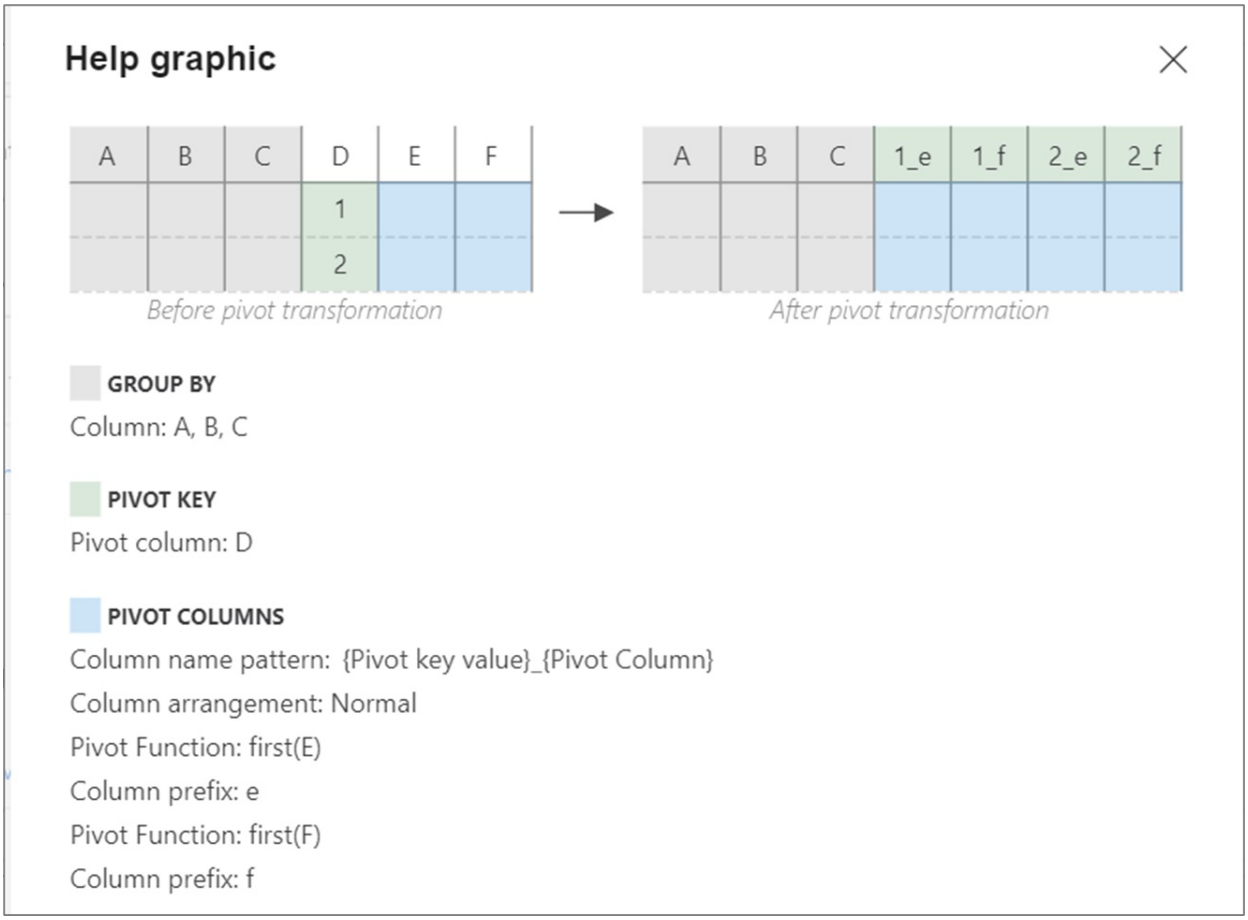
Column name pattern: Select how to format the column name of each pivot column. The outputted column name will be a combination of the pivot key value, column prefix and optional prefix, suffice, middle characters.

Column arrangement: If you generate more than one pivot column per pivot key, choose how you want the columns to be ordered.

Column prefix: If you generate more than one pivot column per pivot key, enter a column prefix for each column. This setting is optional if you only have one pivoted column.

Help graphic

The below help graphic shows how the different pivot components interact with one another



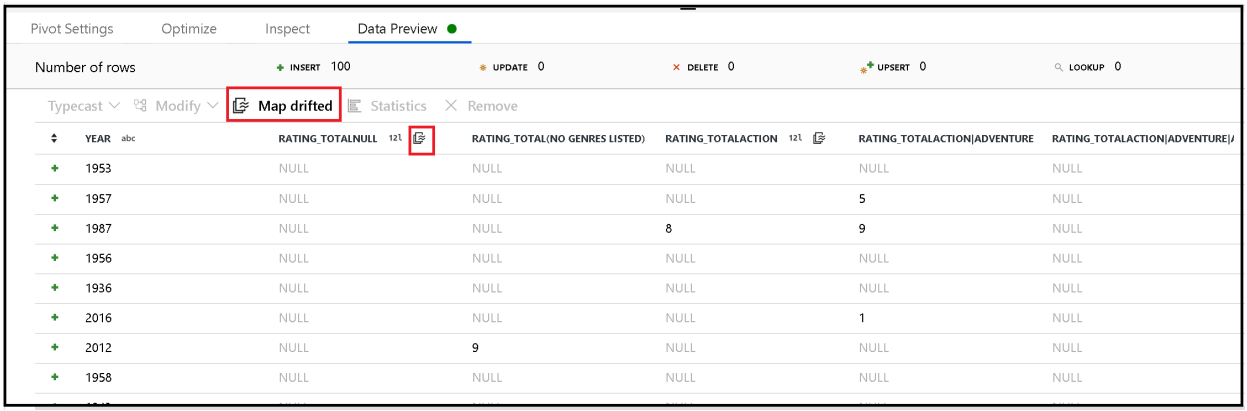
Pivot metadata

If no values are specified in the pivot key configuration, the pivoted columns will be dynamically generated at run time. The number of pivoted columns will equal the number of unique pivot key values multiplied by the number of pivot columns. As this can be a changing number, the UX will not display the column metadata in the **Inspect** tab and there will be no column propagation. To transformation these columns, use the column pattern capabilities of mapping data flow.

If specific pivot key values are set, the pivoted columns will appear in the metadata. The column names will be available to you in the Inspect and Sink mapping.

Generate metadata from drifted columns

Pivot generates new column names dynamically based on row values. You can add these new columns into the metadata that can be referenced later in your data flow. To do this, use the map drifted quick action in data preview.



Sinking pivoted columns

Although pivoted columns are dynamic, they can still be written into your destination data store. Enable **Allow schema drift** in your sink settings. This will allow you to write columns that are not included in metadata. You will not see the new dynamic names in your column metadata, but the schema drift option will allow you to land the data.

Rejoin original fields

The pivot transformation will only project the group by and pivoted columns. If you want your output data to include other input columns, use a self join pattern.

Data flow script

Syntax

```
<incomingStreamName>
    pivot(groupBy(Tm),
        pivotBy(<pivotKeyColumn, [<specifiedColumnName1>,...,<specifiedColumnNameN>]),
        <pivotColumnPrefix> = <pivotedColumnValue>,
        columnNaming: '< prefix >< $N | $V ><middle >< $N | $V >< suffix >',
        lateral: { 'true' | 'false'}
    ) ~> <pivotTransformationName
```

Example

The screens shown in the configuration section, have the following data flow script:

```
BasketballPlayerStats pivot(groupBy(Tm),
    pivotBy(Pos),
    {} = count(),
    columnNaming: '$V$N count',
    lateral: true) ~> PivotExample
```

Next steps

Try the unpivot transformation to turn column values into row values.

Rank transformation in mapping data flow

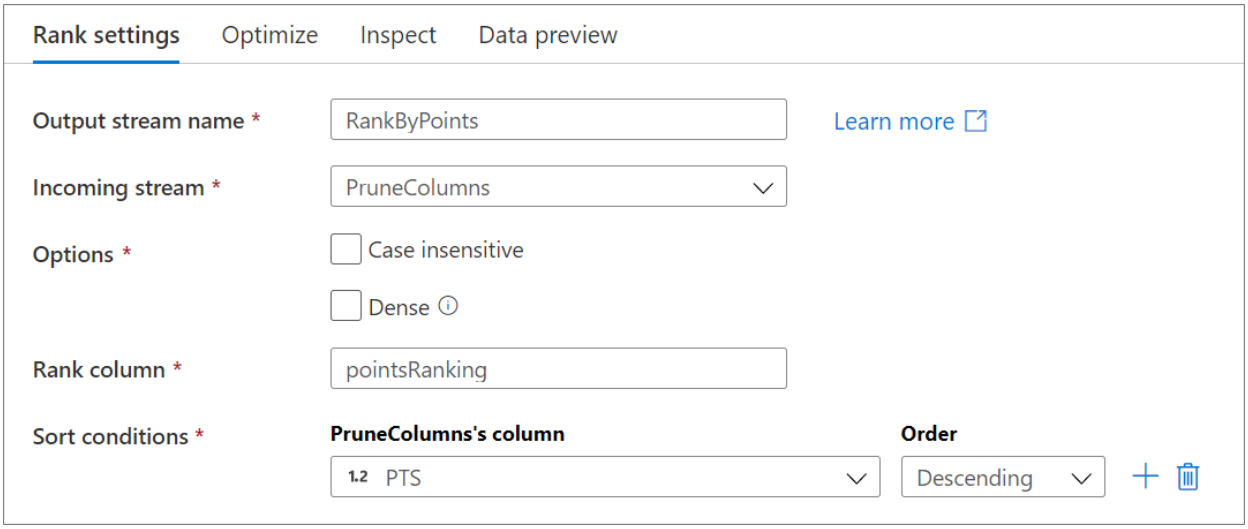
8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

Use the rank transformation to generate an ordered ranking based upon sort conditions specified by the user.

Configuration



Case insensitive: If a sort column is of type string, case will be factored into the ranking.

Dense: If enabled, the rank column will be dense ranked. Each rank count will be a consecutive number and rank values won't be skipped after a tie.

Rank column: The name of the rank column generated. This column will be of type long.

Sort conditions: Choose which columns you're sorting by and in which order the sort happens. The order determines sorting priority.

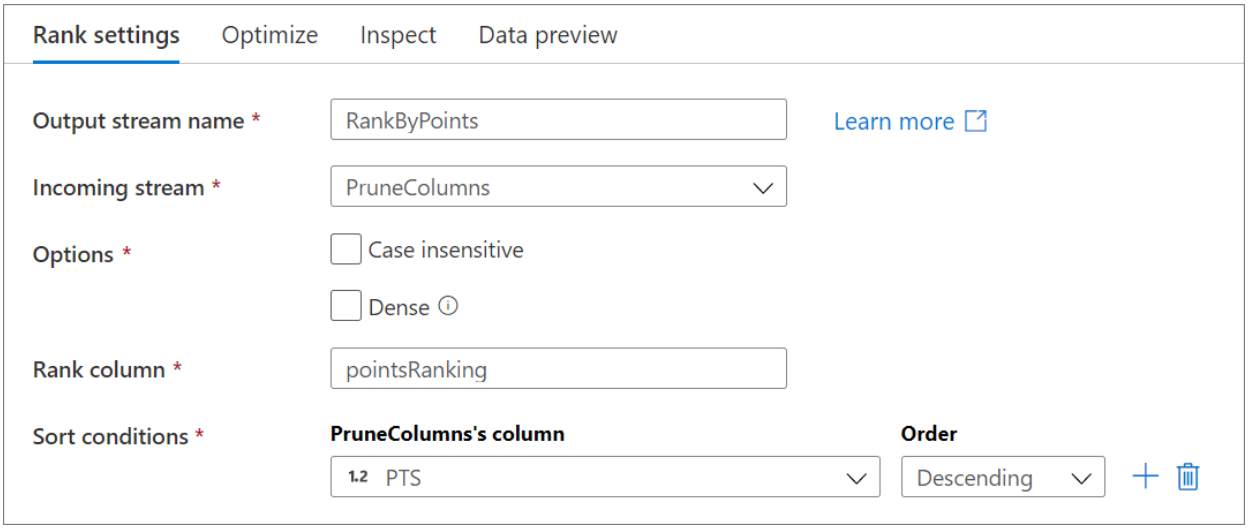
The above configuration takes incoming basketball data and creates a rank column called 'pointsRanking'. The row with the highest value of the column *PTS* will have a *pointsRanking* value of 1.

Data flow script

Syntax

```
<incomingStream>
    rank(
        desc(<sortColumn1>),
        asc(<sortColumn2>),
        ...,
        caseInsensitive: { true | false }
        dense: { true | false }
        output(<rankColumn> as long)
    ) ~> <sortTransformationName<>
```

Example



The data flow script for the above rank configuration is in the following code snippet.

```
PruneColumns
    rank(
        desc(PTS, true),
        caseInsensitive: false,
        output(pointsRanking as long),
        dense: false
    ) ~> RankByPoints
```

Next steps

Filter rows based upon the rank values using the filter transformation.

Select transformation in mapping data flow

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

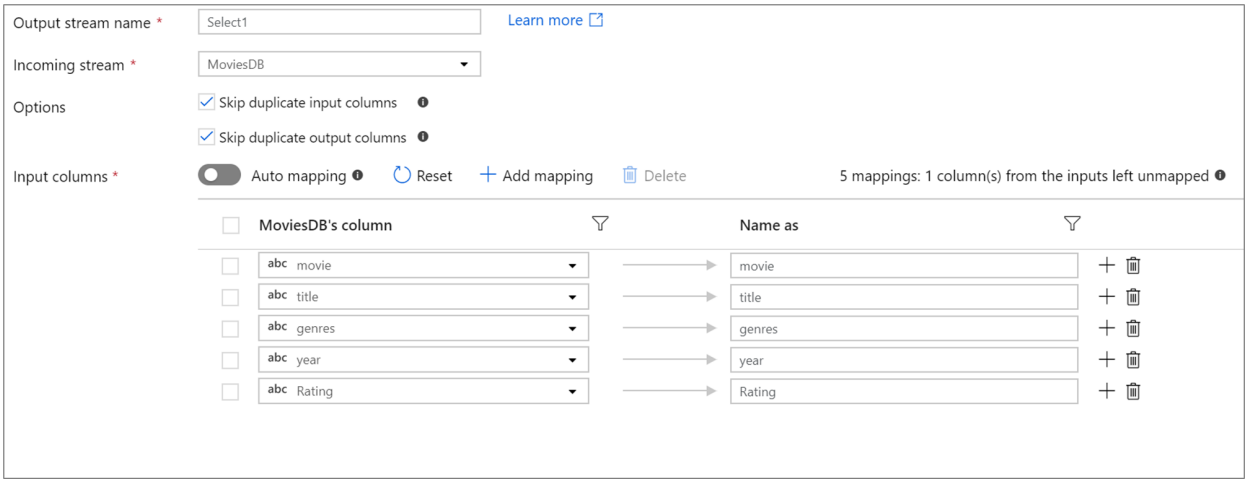
Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

Use the select transformation to rename, drop, or reorder columns. This transformation doesn't alter row data, but chooses which columns are propagated downstream.

In a select transformation, users can specify fixed mappings, use patterns to do rule-based mapping, or enable auto mapping. Fixed and rule-based mappings can both be used within the same select transformation. If a column doesn't match one of the defined mappings, it will be dropped.

Fixed mapping

If there are fewer than 50 columns defined in your projection, all defined columns will have a fixed mapping by default. A fixed mapping takes a defined, incoming column and maps it an exact name.

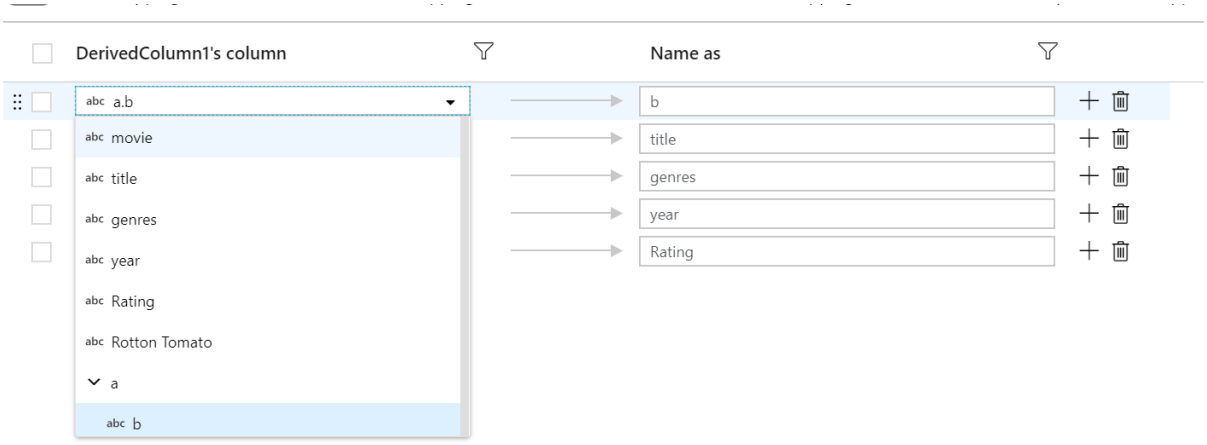


NOTE

You can't map or rename a drifted column using a fixed mapping

Mapping hierarchical columns

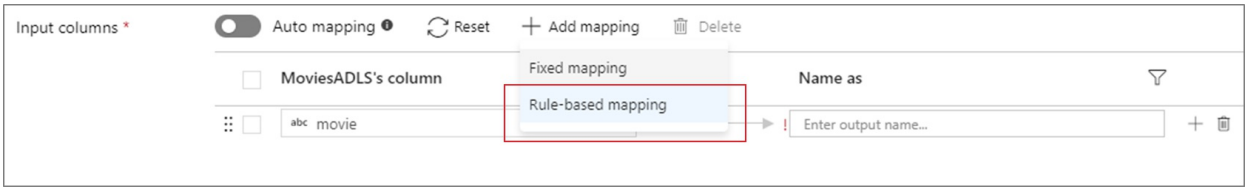
Fixed mappings can be used to map a subcolumn of a hierarchical column to a top-level column. If you have a defined hierarchy, use the column dropdown to select a subcolumn. The select transformation will create a new column with the value and data type of the subcolumn.



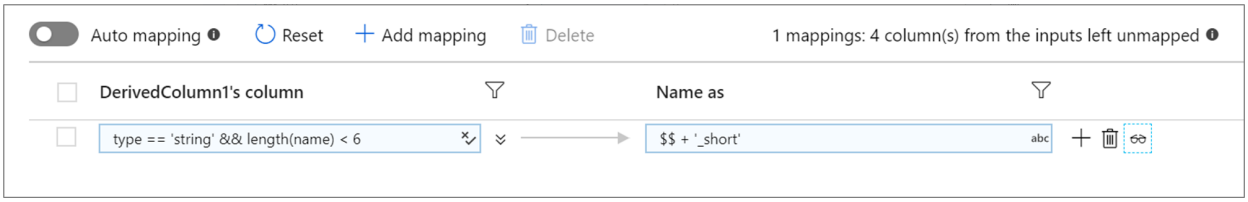
Rule-based mapping

If you wish to map many columns at once or pass drifted columns downstream, use rule-based mapping to define your mappings using column patterns. Match based on the `name`, `type`, `stream`, and `position` of columns. You can have any combination of fixed and rule-based mappings. By default, all projections with greater than 50 columns will default to a rule-based mapping that matches on every column and outputs the inputted name.

To add a rule-based mapping, click **Add mapping** and select **Rule-based mapping**.



Each rule-based mapping requires two inputs: the condition on which to match by and what to name each mapped column. Both values are inputted via the expression builder. In the left expression box, enter your boolean match condition. In the right expression box, specify what the matched column will be mapped to.

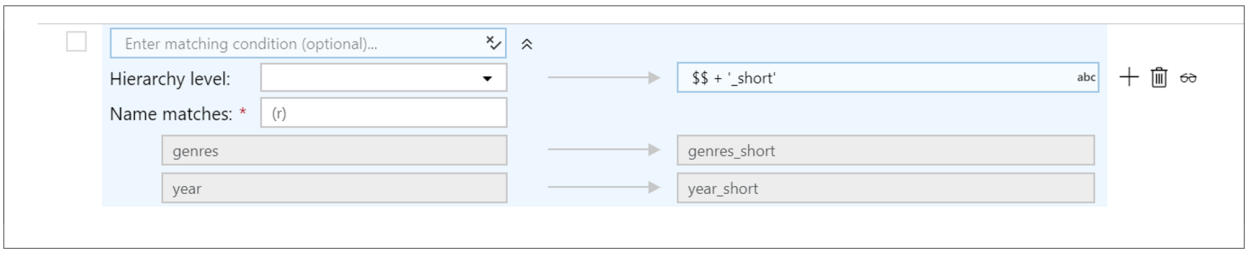


Use `$$` syntax to reference the input name of a matched column. Using the above image as an example, say a user wants to match on all string columns whose names are shorter than six characters. If one incoming column was named `test`, the expression `$$ + '_short'` will rename the column `test_short`. If that's the only mapping that exists, all columns that don't meet the condition will be dropped from the outputted data.

Patterns match both drifted and defined columns. To see which defined columns are mapped by a rule, click the eyeglasses icon next to the rule. Verify your output using data preview.

Regex mapping

If you click the downward chevron icon, you can specify a regex-mapping condition. A regex-mapping condition matches all column names that match the specified regex condition. This can be used in combination with standard rule-based mappings.

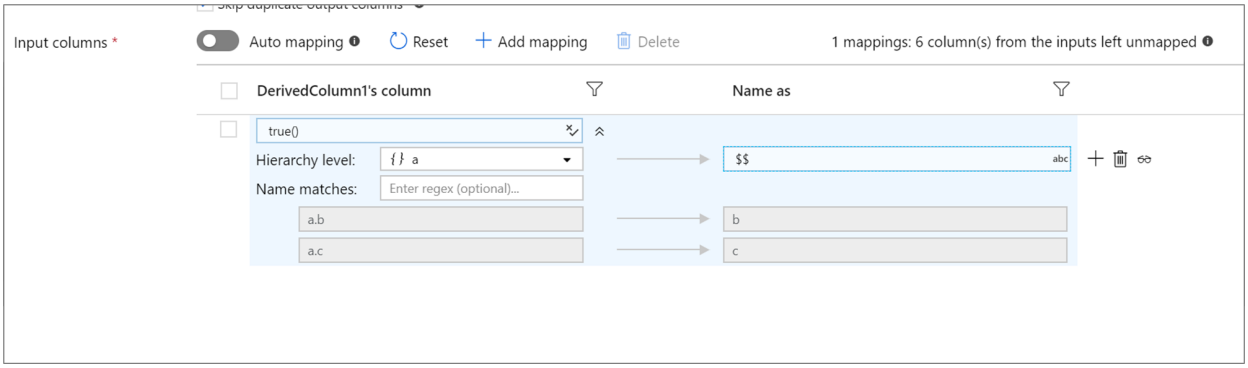


The above example matches on regex pattern `(r)` or any column name that contains a lower case r. Similar to standard rule-based mapping, all matched columns are altered by the condition on the right using `$$` syntax.

If you have multiple regex matches in your column name, you can refer to specific matches using `$n` where 'n' refers to which match. For example, '$2' refers to the second match within a column name.

Rule-based hierarchies

If your defined projection has a hierarchy, you can use rule-based mapping to map the hierarchies subcolumns. Specify a matching condition and the complex column whose subcolumns you wish to map. Every matched subcolumn will be outputted using the 'Name as' rule specified on the right.



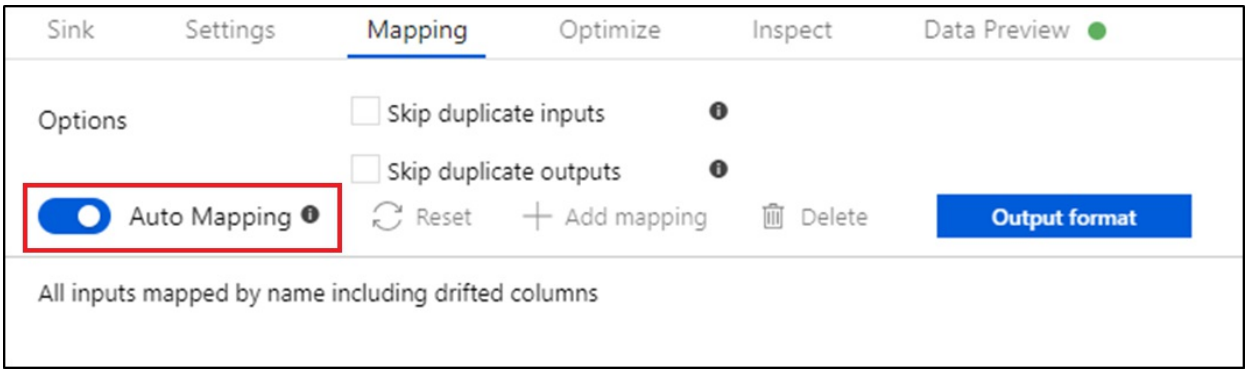
The above example matches on all subcolumns of complex column `a`. `a` contains two subcolumns `b` and `c`. The output schema will include two columns `b` and `c` as the 'Name as' condition is `$$`.

Parameterization

You can parameterize column names using rule-based mapping. Use the keyword `name` to match incoming column names against a parameter. For example, if you have a data flow parameter `mycolumn`, you can create a rule that matches any column name that is equal to `mycolumn`. You can rename the matched column to a hard-coded string such as 'business key' and reference it explicitly. In this example, the matching condition is `name == $mycolumn` and the name condition is 'business key'.

Auto mapping

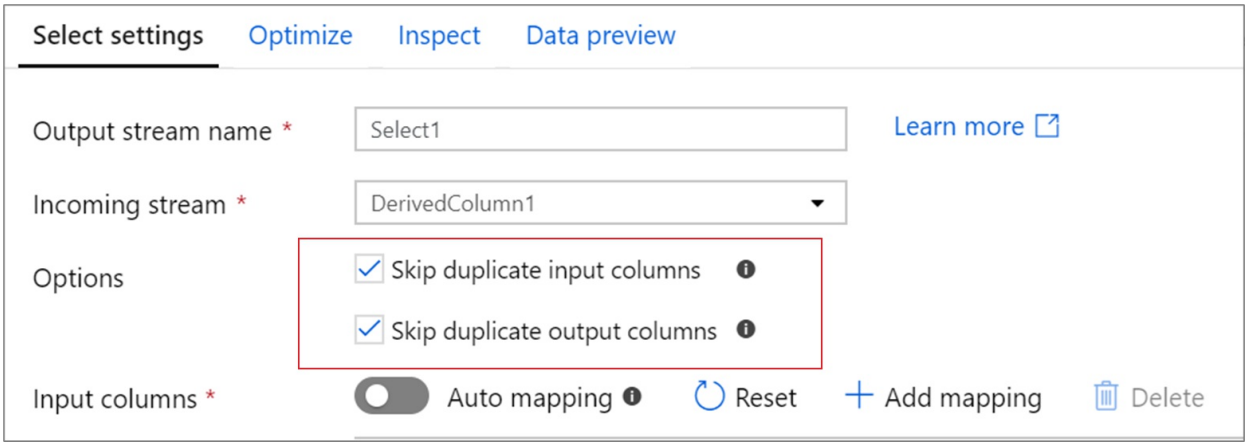
When adding a select transformation, **Auto mapping** can be enabled by switching the Auto mapping slider. With auto mapping, the select transformation maps all incoming columns, excluding duplicates, with the same name as their input. This will include drifted columns, which means the output data may contain columns not defined in your schema. For more information on drifted columns, see schema drift.



With auto mapping on, the select transformation will honor the skip duplicate settings and provide a new alias for the existing columns. Aliasing is useful when doing multiple joins or lookups on the same stream and in self-join scenarios.

Duplicate columns

By default, the select transformation drops duplicate columns in both the input and output projection. Duplicate input columns often come from join and lookup transformations where column names are duplicated on each side of the join. Duplicate output columns can occur if you map two different input columns to the same name. Choose whether to drop or pass on duplicate columns by toggling the checkbox.



Ordering of columns

The order of mappings determines the order of the output columns. If an input column is mapped multiple times, only the first mapping will be honored. For any duplicate column dropping, the first match will be kept.

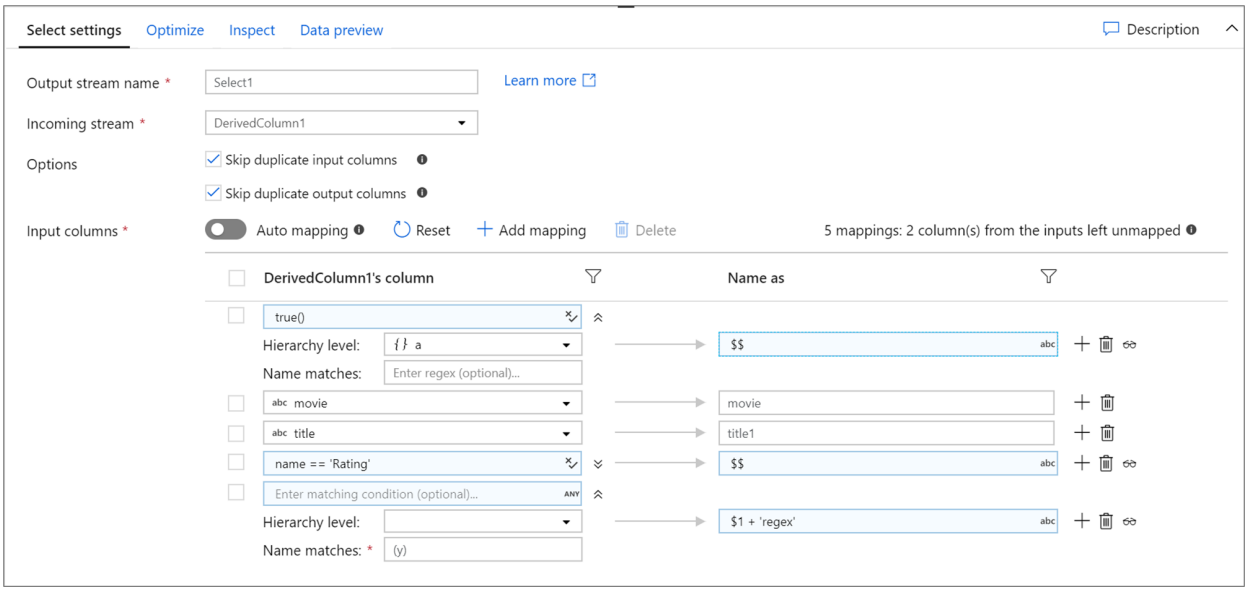
Data flow script

Syntax

```
<incomingStream>
    select(mapColumn(
        each(<hierarchicalColumn>, match(<matchCondition>), <nameCondition> = $$), ## hierarchical rule-
based matching
        <fixedColumn>, ## fixed mapping, no rename
        <renamedFixedColumn> = <fixedColumn>, ## fixed mapping, rename
        each(match(<matchCondition>), <nameCondition> = $$), ## rule-based mapping
        each(patternMatch(<regexMatching>), <nameCondition> = $$) ## regex mapping
    ),
    skipDuplicateMapInputs: { true | false },
    skipDuplicateMapOutputs: { true | false }) ~> <selectTransformationName>
```

Example

Below is an example of a select mapping and its data flow script:




```
DerivedColumn1 select(mapColumn(
        each(a, match(true())),
        movie,
        title1 = title,
        each(match(name == 'Rating')),
        each(patternMatch(`(y)`),
            $1 + 'regex' = $$)
    ),
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> Select1
```

Next steps

- After using Select to rename, reorder, and alias columns, use the Sink transformation to land your data into a data store.

Sink transformation in mapping data flow

8/19/2022 • 7 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

After you finish transforming your data, write it into a destination store by using the sink transformation. Every data flow requires at least one sink transformation, but you can write to as many sinks as necessary to complete your transformation flow. To write to additional sinks, create new streams via new branches and conditional splits.

Each sink transformation is associated with exactly one dataset object or linked service. The sink transformation determines the shape and location of the data you want to write to.

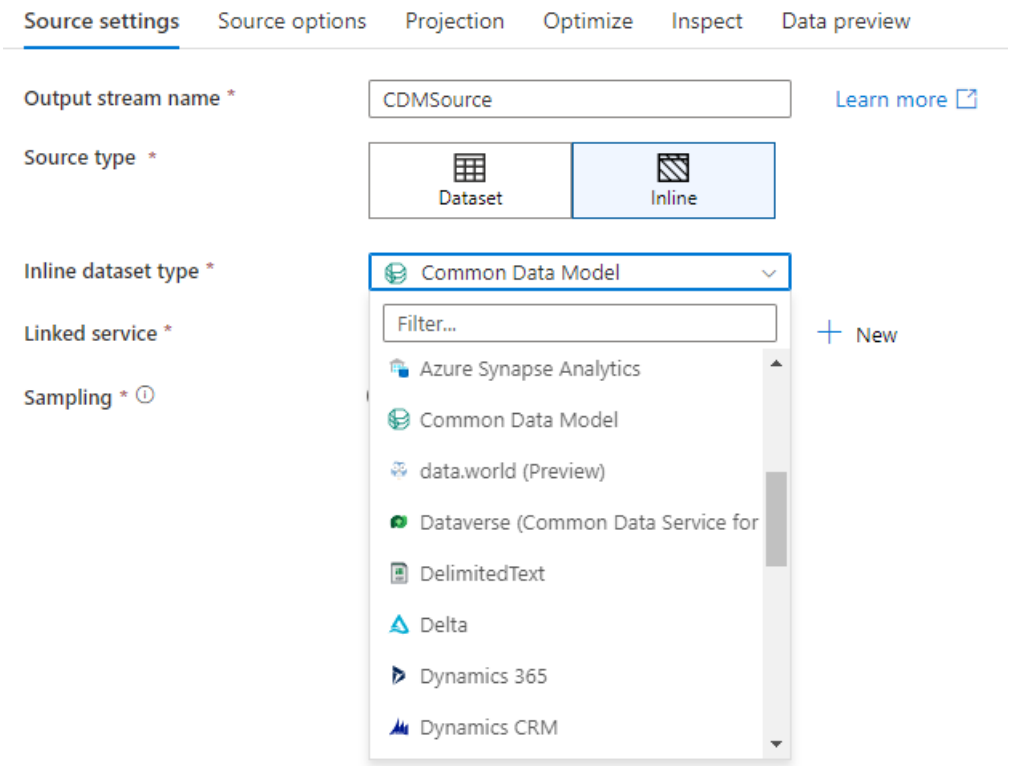
Inline datasets

When you create a sink transformation, choose whether your sink information is defined inside a dataset object or within the sink transformation. Most formats are available in only one or the other. To learn how to use a specific connector, see the appropriate connector document.

When a format is supported for both inline and in a dataset object, there are benefits to both. Dataset objects are reusable entities that can be used in other data flows and activities such as Copy. These reusable entities are especially useful when you use a hardened schema. Datasets aren't based in Spark. Occasionally, you might need to override certain settings or schema projection in the sink transformation.

Inline datasets are recommended when you use flexible schemas, one-off sink instances, or parameterized sinks. If your sink is heavily parameterized, inline datasets allow you to not create a "dummy" object. Inline datasets are based in Spark, and their properties are native to data flow.

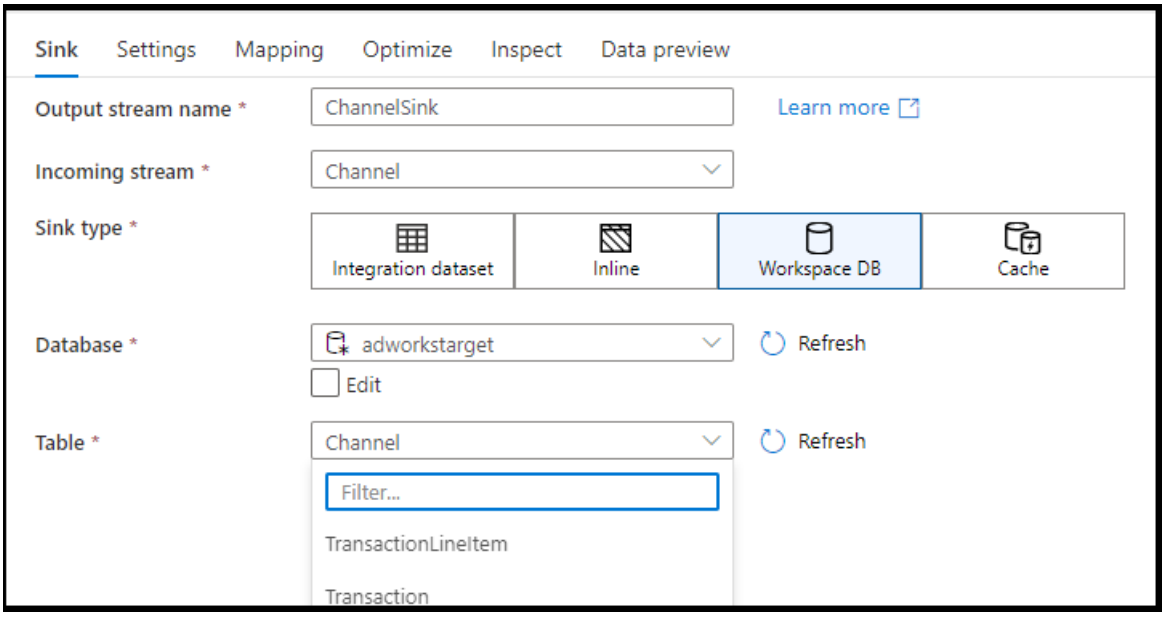
To use an inline dataset, select the format you want in the **Sink type** selector. Instead of selecting a sink dataset, you select the linked service you want to connect to.



Workspace DB (Synapse workspaces only)

When using data flows in Azure Synapse workspaces, you will have an additional option to sink your data directly into a database type that is inside your Synapse workspace. This will alleviate the need to add linked services or datasets for those databases.

> **NOTE**
> The Azure Synapse Workspace DB connector is currently in public preview and can only work with Spark Lake databases at this time



Supported sink types

Mapping data flow follows an extract, load, and transform (ELT) approach and works with *staging* datasets that are all in Azure. Currently, the following datasets can be used in a source transformation.

CONNECTOR	FORMAT	DATASET/INLINE
Azure Blob Storage	Avro Delimited text Delta JSON ORC Parquet	✓/- ✓/- -/✓ ✓/- ✓/✓ ✓/-
Azure Cosmos DB (SQL API)		✓/-
Azure Data Lake Storage Gen1	Avro Delimited text JSON ORC Parquet	✓/- ✓/- ✓/- ✓/✓ ✓/-
Azure Data Lake Storage Gen2	Avro Common Data Model Delimited text Delta JSON ORC Parquet	✓/- -/✓ ✓/- -/✓ ✓/- ✓/✓ ✓/-
Azure Database for MySQL		✓/✓
Azure Database for PostgreSQL		✓/✓
Azure Data Explorer		✓/✓
Azure SQL Database		✓/✓
Azure SQL Managed Instance		✓/-
Azure Synapse Analytics		✓/-
Dataverse		✓/✓
Dynamics 365		✓/✓
Dynamics CRM		✓/✓
SFTP	Avro Delimited text JSON ORC Parquet	✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓
Snowflake		✓/✓
SQL Server		✓/✓

Settings specific to these connectors are located on the **Settings** tab. Information and data flow script examples on these settings are located in the connector documentation.

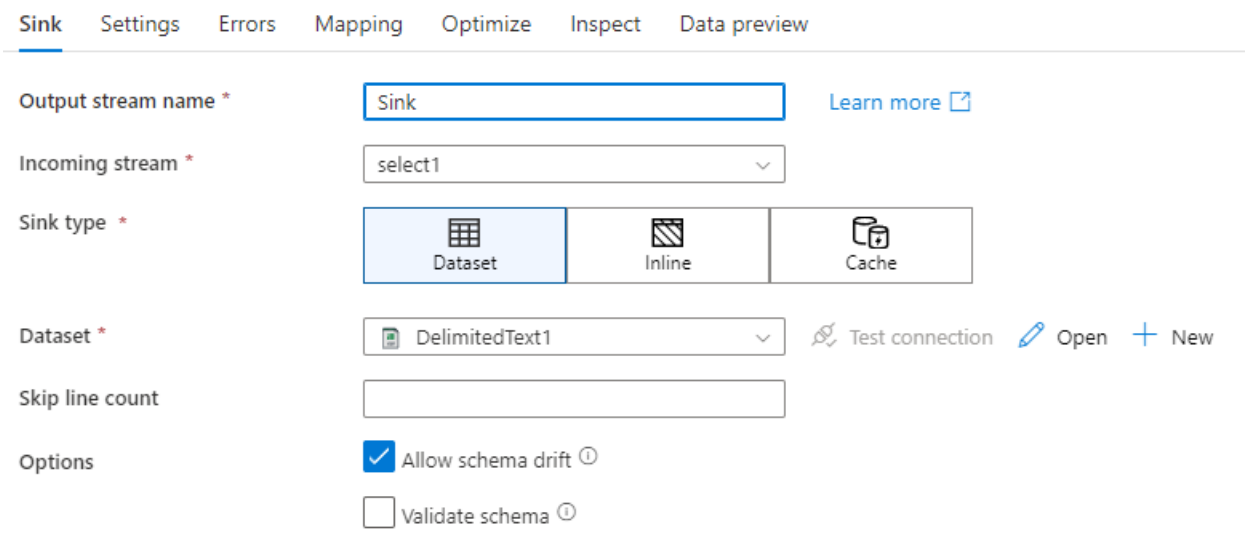
The service has access to more than 90 native connectors. To write data to those other sources from your data

flow, use the Copy Activity to load that data from a supported sink.

Sink settings

After you've added a sink, configure via the **Sink** tab. Here you can pick or create the dataset your sink writes to. Development values for dataset parameters can be configured in Debug settings. (Debug mode must be turned on.)

The following video explains a number of different sink options for text-delimited file types.



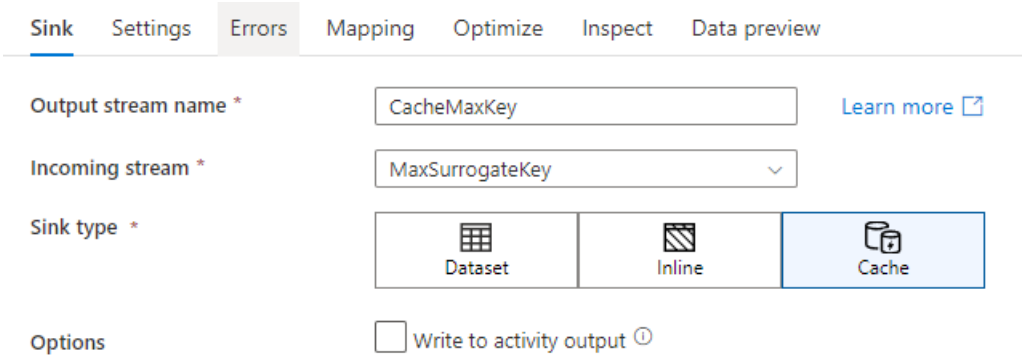
Schema drift: Schema drift is the ability of the service to natively handle flexible schemas in your data flows without needing to explicitly define column changes. Enable **Allow schema drift** to write additional columns on top of what's defined in the sink data schema.

Validate schema: If validate schema is selected, the data flow will fail if any column in sink projection isn't found in the sink store, or if the data types don't match. Use this setting to enforce that the sink schema meets the contract of your defined projection. It's useful in database sink scenarios to signal that column names or types have changed.

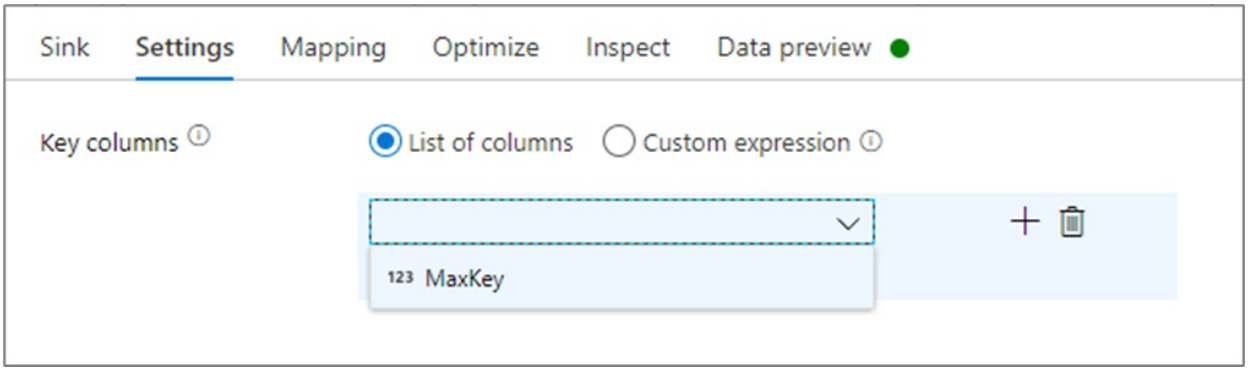
Cache sink

A *cache sink* is when a data flow writes data into the Spark cache instead of a data store. In mapping data flows, you can reference this data within the same flow many times using a *cache lookup*. This is useful when you want to reference data as part of an expression but don't want to explicitly join the columns to it. Common examples where a cache sink can help are looking up a max value on a data store and matching error codes to an error message database.

To write to a cache sink, add a sink transformation and select **Cache** as the sink type. Unlike other sink types, you don't need to select a dataset or linked service because you aren't writing to an external store.



In the sink settings, you can optionally specify the key columns of the cache sink. These are used as matching conditions when using the `lookup()` function in a cache lookup. If you specify key columns, you can't use the `outputs()` function in a cache lookup. To learn more about the cache lookup syntax, see cached lookups.



For example, if I specify a single key column of `column1` in a cache sink called `cacheExample`, calling `cacheExample#lookup()` would have one parameter specifies which row in the cache sink to match on. The function outputs a single complex column with subcolumns for each column mapped.

> **NOTE**
> A cache sink must be in a completely independent data stream from any transformation referencing it via a cache lookup. A cache sink also must be the first sink written.

Write to activity output The cached sink can optionally write your output data to the input of the next pipeline activity. This will allow you to quickly and easily pass data out of your data flow activity without needing to persist the data in a data store.

Field mapping

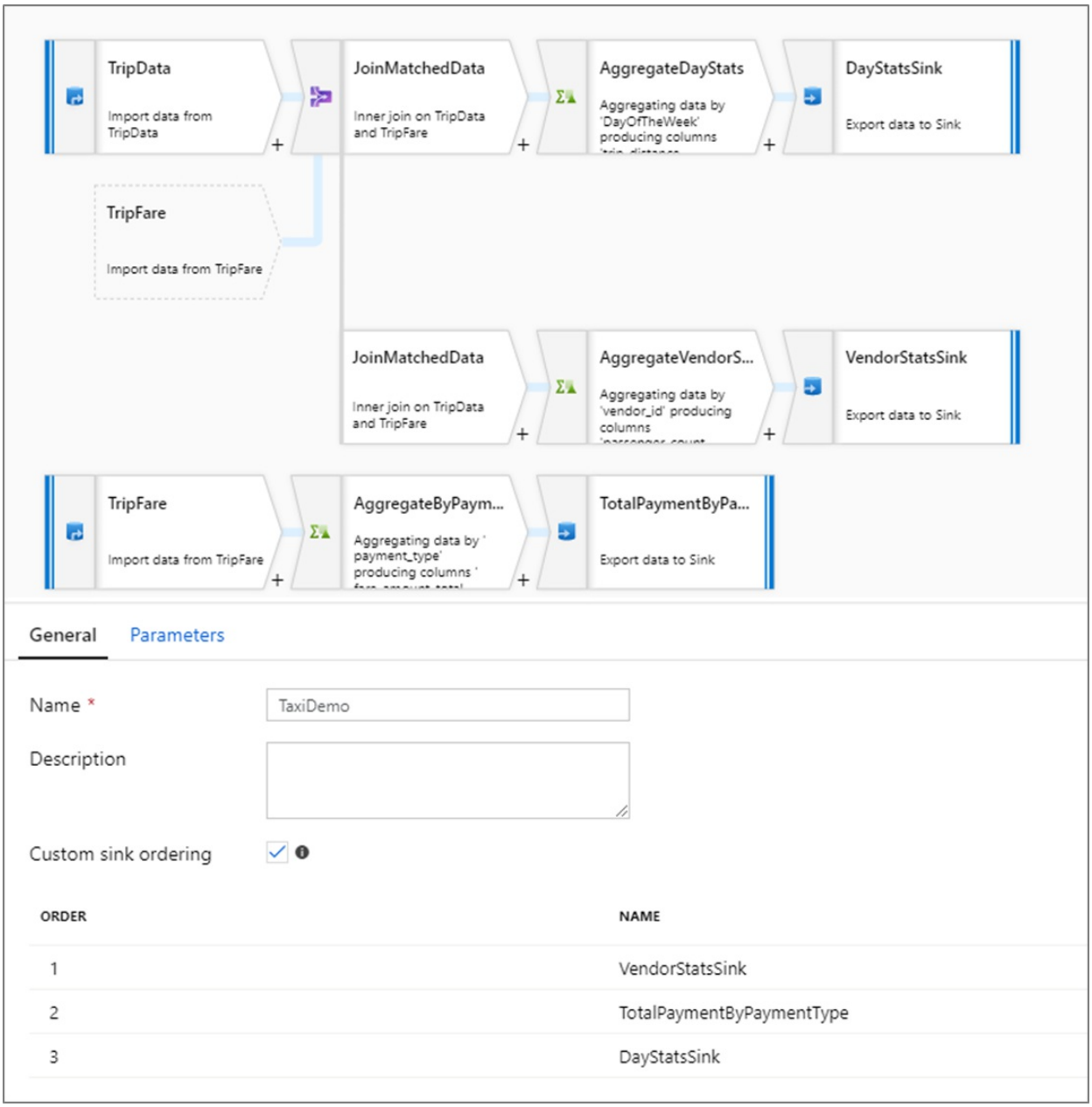
Similar to a select transformation, on the **Mapping** tab of the sink, you can decide which incoming columns will get written. By default, all input columns, including drifted columns, are mapped. This behavior is known as *automapping*.

When you turn off automapping, you can add either fixed column-based mappings or rule-based mappings. With rule-based mappings, you can write expressions with pattern matching. Fixed mapping maps logical and physical column names. For more information on rule-based mapping, see Column patterns in mapping data flow.

Custom sink ordering

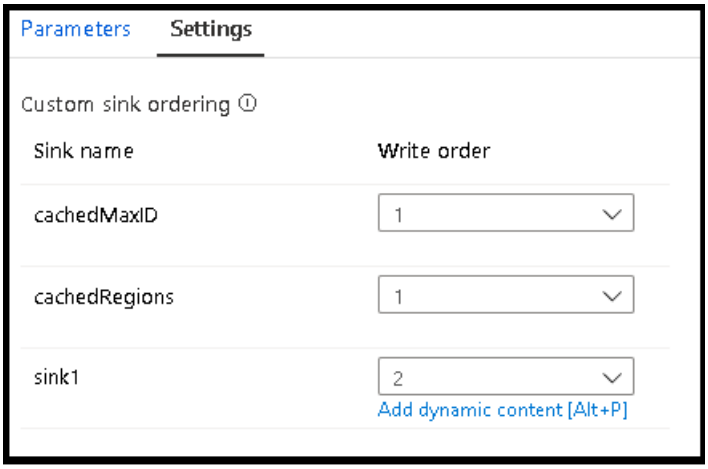
By default, data is written to multiple sinks in a nondeterministic order. The execution engine writes data in parallel as the transformation logic is completed, and the sink ordering might vary each run. To specify an exact sink ordering, enable **Custom sink ordering** on the **General** tab of the data flow. When enabled, sinks are

written sequentially in increasing order.



NOTE

When utilizing cached lookups, make sure that your sink ordering has the cached sinks set to 1, the lowest (or first) in ordering.



Sink groups

You can group sinks together by applying the same order number for a series of sinks. The service will treat those sinks as groups that can execute in parallel. Options for parallel execution will surface in the pipeline data flow activity.

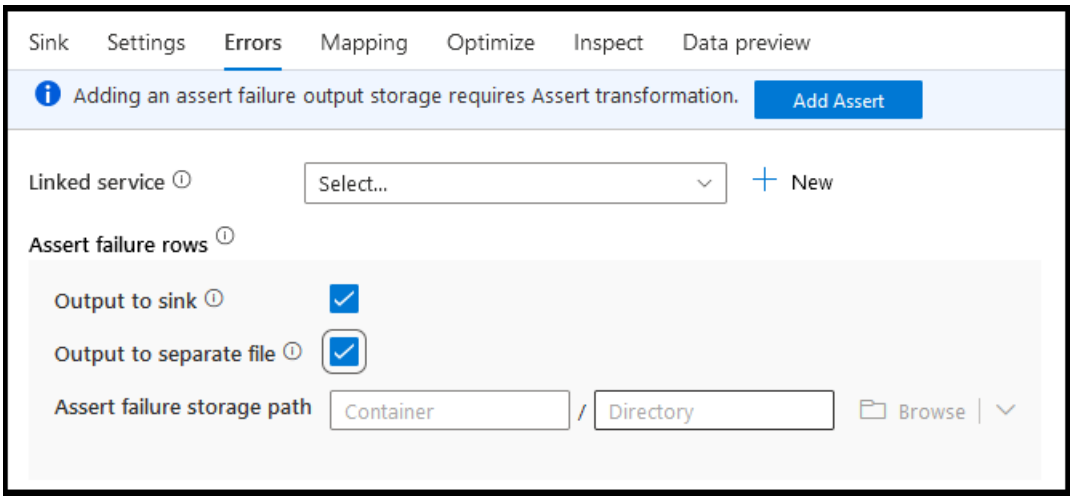
Errors

On the sink errors tab you can configure error row handling to capture and redirect output for database driver errors and failed assertions.

When writing to databases, certain rows of data may fail due to constraints set by the destination. By default, a data flow run will fail on the first error it gets. In certain connectors, you can choose to **Continue on error** that allows your data flow to complete even if individual rows have errors. Currently, this capability is only available in Azure SQL Database and Azure Synapse. For more information, see error row handling in Azure SQL DB.

Below is a video tutorial on how to use database error row handling automatically in your sink transformation.

For assert failure rows, you can use the Assert transformation upstream in your data flow and then redirect failed assertions to an output file here in the sink errors tab. You also have an option here to ignore rows with assertion failures and not output those rows at all to the sink destination data store.



Data preview in sink

When fetching a data preview in debug mode, no data will be written to your sink. A snapshot of what the data looks like will be returned, but nothing will be written to your destination. To test writing data into your sink, run a pipeline debug from the pipeline canvas.

Data flow script

Example

Below is an example of a sink transformation and its data flow script:

```
sink(input(
  movie as integer,
  title as string,
  genres as string,
  year as integer,
  Rating as integer
 ),
 allowSchemaDrift: true,
 validateSchema: false,
 deletable:false,
 insertable:false,
 updateable:true,
 upsertable:false,
 keys:['movie'],
 format: 'table',
 skipDuplicateMapInputs: true,
 skipDuplicateMapOutputs: true,
 saveOrder: 1,
 errorHandlingOption: 'stopOnFirstError') ~> sink1
```

Next steps

Now that you've created your data flow, add a data flow activity to your pipeline.

Sort transformation in mapping data flow

8/19/2022 • 2 minutes to read • Edit Online

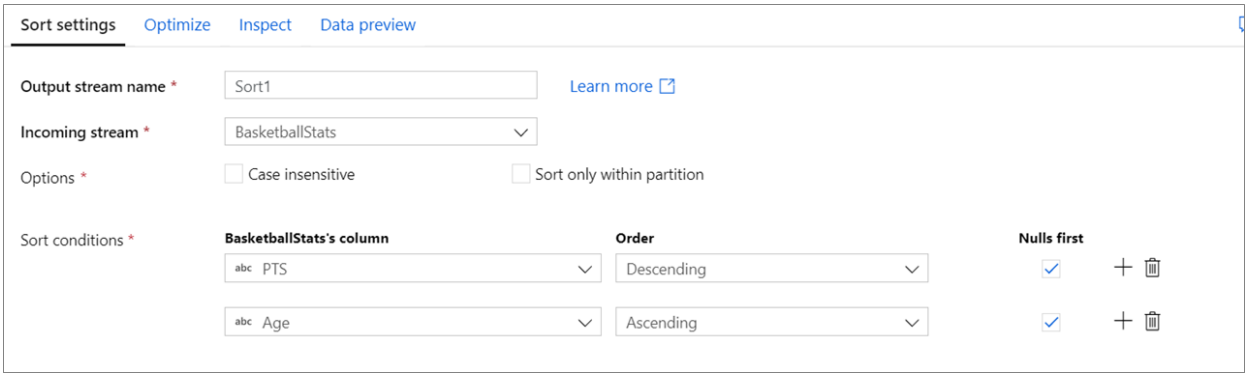
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

The sort transformation allows you to sort the incoming rows on the current data stream. You can choose individual columns and sort them in ascending or descending order.

> **NOTE**
> Mapping data flows are executed on spark clusters which distribute data across multiple nodes and partitions. If you choose to repartition your data in a subsequent transformation, you may lose your sorting due to reshuffling of data. The best way to maintain sort order in your data flow is to set single partition in the Optimize tab on the transformation and keep the Sort transformation as close to the Sink as possible.

Configuration



Case insensitive: Whether or not you wish to ignore case when sorting string or text fields

Sort Only Within Partitions: As data flows are run on spark, each data stream is divided into partitions. This setting sorts data only within the incoming partitions rather than sorting the entire data stream.

Sort conditions: Choose which columns you are sorting by and in which order the sort happens. The order determines sorting priority. Choose whether or not nulls will appear at the beginning or end of the data stream.

Computed columns

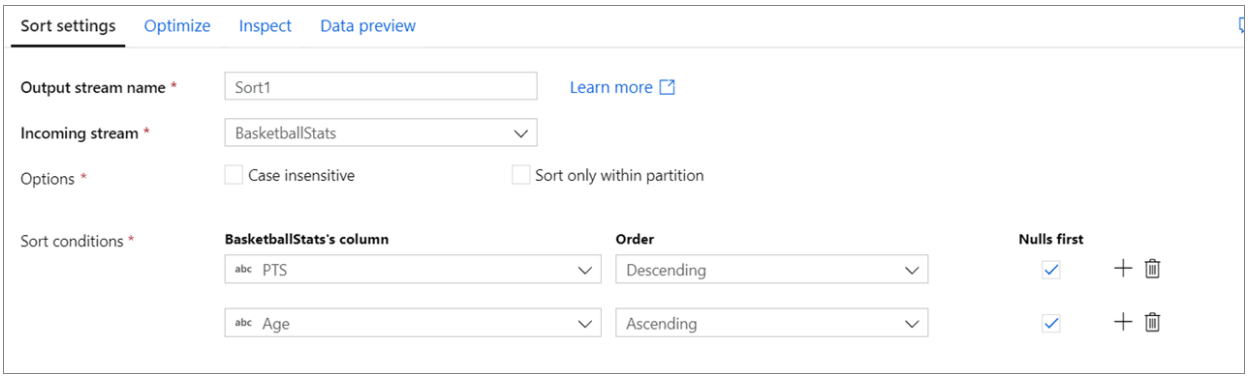
To modify or extract a column value before applying the sort, hover over the column and select "computed column". This will open the expression builder to create an expression for the sort operation instead of using a column value.

Data flow script

Syntax

```
<incomingStream>
    sort(
        desc(<sortColumn1>, { true | false }),
        asc(<sortColumn2>, { true | false }),
        ...
    ) ~> <sortTransformationName<>
```

Example



The data flow script for the above sort configuration is in the code snippet below.

```
BasketballStats sort(desc(PTS, true),
    asc(Age, true)) ~> Sort1
```

Next steps

After sorting, you may want to use the Aggregate Transformation

Source transformation in mapping data flow

8/19/2022 • 7 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

A source transformation configures your data source for the data flow. When you design data flows, your first step is always configuring a source transformation. To add a source, select the **Add Source** box in the data flow canvas.

Every data flow requires at least one source transformation, but you can add as many sources as necessary to complete your data transformations. You can join those sources together with a join, lookup, or a union transformation.

Each source transformation is associated with exactly one dataset or linked service. The dataset defines the shape and location of the data you want to write to or read from. If you use a file-based dataset, you can use wildcards and file lists in your source to work with more than one file at a time.

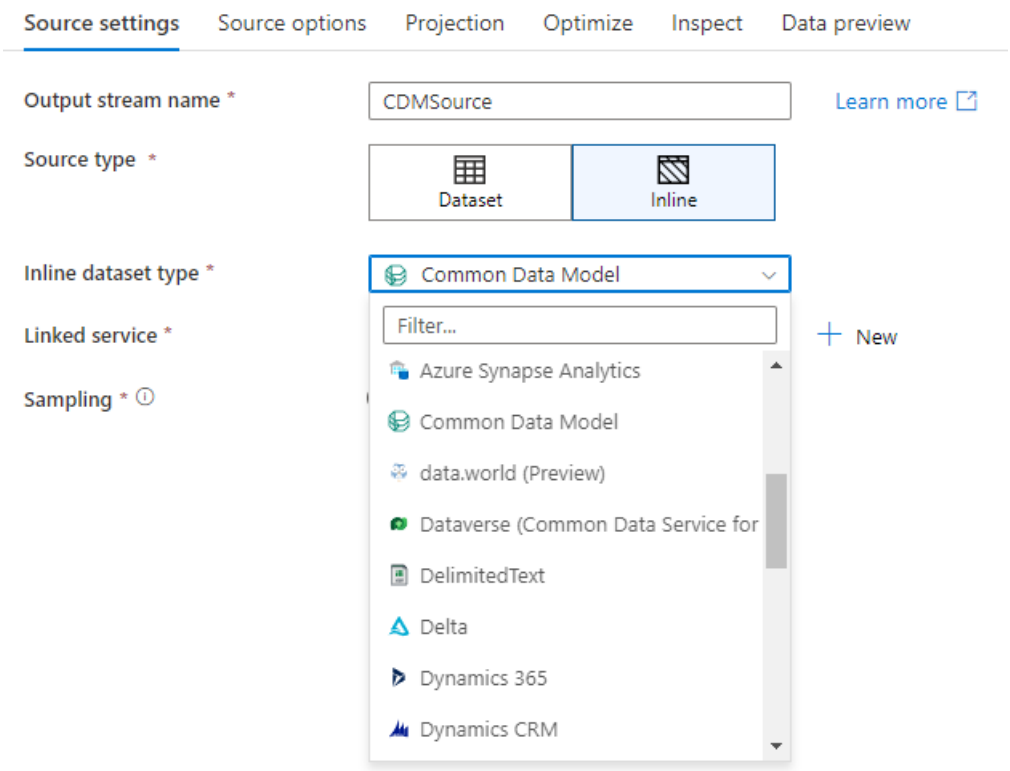
Inline datasets

The first decision you make when you create a source transformation is whether your source information is defined inside a dataset object or within the source transformation. Most formats are available in only one or the other. To learn how to use a specific connector, see the appropriate connector document.

When a format is supported for both inline and in a dataset object, there are benefits to both. Dataset objects are reusable entities that can be used in other data flows and activities such as Copy. These reusable entities are especially useful when you use a hardened schema. Datasets aren't based in Spark. Occasionally, you might need to override certain settings or schema projection in the source transformation.

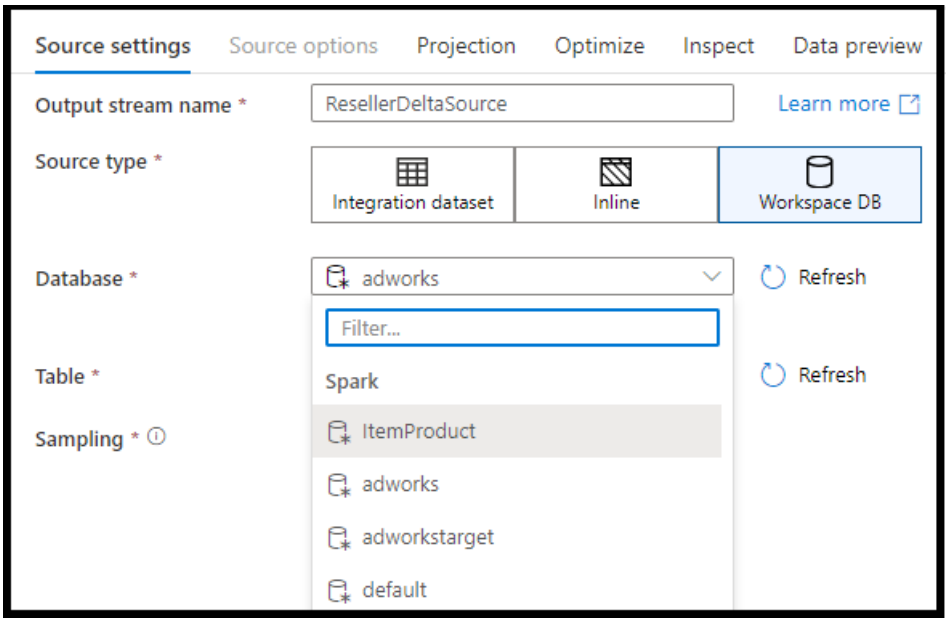
Inline datasets are recommended when you use flexible schemas, one-off source instances, or parameterized sources. If your source is heavily parameterized, inline datasets allow you to not create a "dummy" object. Inline datasets are based in Spark, and their properties are native to data flow.

To use an inline dataset, select the format you want in the **Source type** selector. Instead of selecting a source dataset, you select the linked service you want to connect to.



Workspace DB (Synapse workspaces only)

In Azure Synapse workspaces, an additional option is present in data flow source transformations called `Workspace DB`. This will allow you to directly pick a workspace database of any available type as your source data without requiring additional linked services or datasets.



Supported source types

Mapping data flow follows an extract, load, and transform (ELT) approach and works with *staging* datasets that are all in Azure. Currently, the following datasets can be used in a source transformation.

CONNECTOR	FORMAT	DATASET/INLINE

CONNECTOR	FORMAT	DATASET/INLINE
Amazon S3	Avro Delimited text Delta Excel JSON ORC Parquet XML	✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓
Asana (Preview)		-/✓
Azure Blob Storage	Avro Delimited text Delta Excel JSON ORC Parquet XML	✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓
Azure Cosmos DB (SQL API)		✓/-
Azure Data Lake Storage Gen1	Avro Delimited text Excel JSON ORC Parquet XML	✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓
Azure Data Lake Storage Gen2	Avro Common Data Model Delimited text Delta Excel JSON ORC Parquet XML	✓/✓ -/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓
Azure Database for MySQL		✓/✓
Azure Database for PostgreSQL		✓/✓
Azure Data Explorer		✓/✓
Azure SQL Database		✓/✓
Azure SQL Managed Instance		✓/✓
Azure Synapse Analytics		✓/✓
data.world (Preview)		-/✓
Dataverse		✓/✓

CONNECTOR	FORMAT	DATASET/INLINE
Dynamics 365		✓/✓
Dynamics CRM		✓/✓
Hive		-/✓
Quickbase (Preview)		-/✓
SFTP	Avro Delimited text Excel JSON ORC Parquet XML	✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓ ✓/✓
Smartsheet (Preview)		-/✓
Snowflake		✓/✓
SQL Server		✓/✓
REST		✓/✓
TeamDesk (Preview)		-/✓
Twilio (Preview)		-/✓
Zendesk (Preview)		-/✓

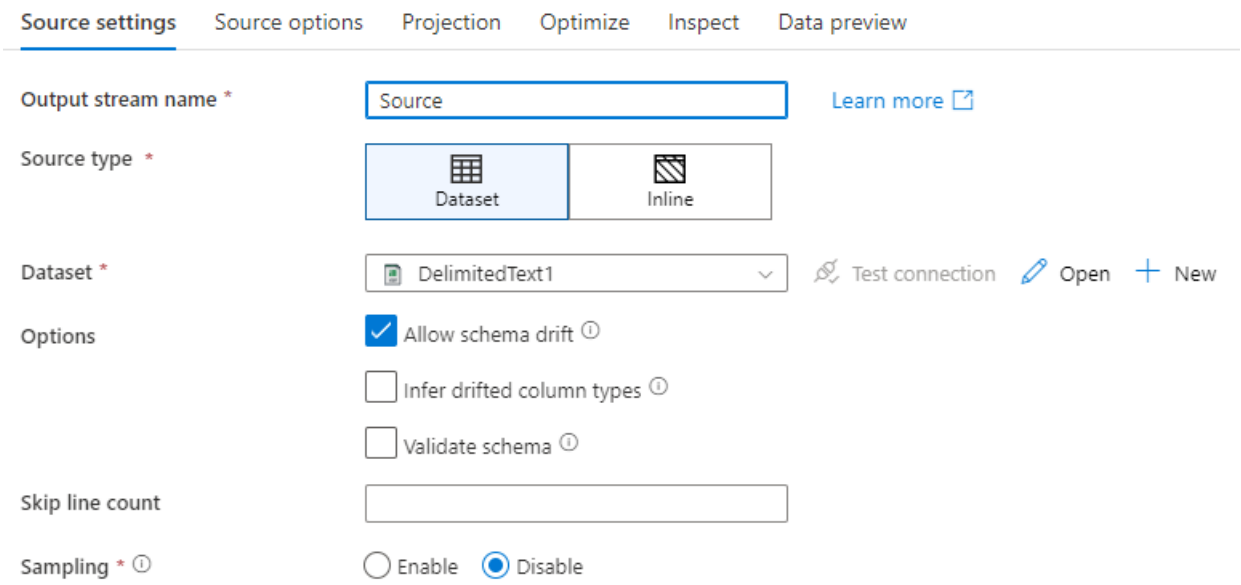
Settings specific to these connectors are located on the **Source options** tab. Information and data flow script examples on these settings are located in the connector documentation.

Azure Data Factory and Synapse pipelines have access to more than 90 native connectors. To include data from those other sources in your data flow, use the Copy Activity to load that data into one of the supported staging areas.

Source settings

After you've added a source, configure via the **Source settings** tab. Here you can pick or create the dataset your source points at. You can also select schema and sampling options for your data.

Development values for dataset parameters can be configured in debug settings. (Debug mode must be turned on.)



Output stream name: The name of the source transformation.

Source type: Choose whether you want to use an inline dataset or an existing dataset object.

Test connection: Test whether or not the data flow's Spark service can successfully connect to the linked service used in your source dataset. Debug mode must be on for this feature to be enabled.

Schema drift: Schema drift is the ability of the service to natively handle flexible schemas in your data flows without needing to explicitly define column changes.

- Select the **Allow schema drift** check box if the source columns will change often. This setting allows all incoming source fields to flow through the transformations to the sink.
- Selecting **Infer drifted column types** instructs the service to detect and define data types for each new column discovered. With this feature turned off, all drifted columns will be of type string.

Validate schema: If **Validate schema** is selected, the data flow will fail to run if the incoming source data doesn't match the defined schema of the dataset.

Skip line count: The **Skip line count** field specifies how many lines to ignore at the beginning of the dataset.

Sampling: Enable **Sampling** to limit the number of rows from your source. Use this setting when you test or sample data from your source for debugging purposes. This is very useful when executing data flows in debug mode from a pipeline.

To validate your source is configured correctly, turn on debug mode and fetch a data preview. For more information, see Debug mode.

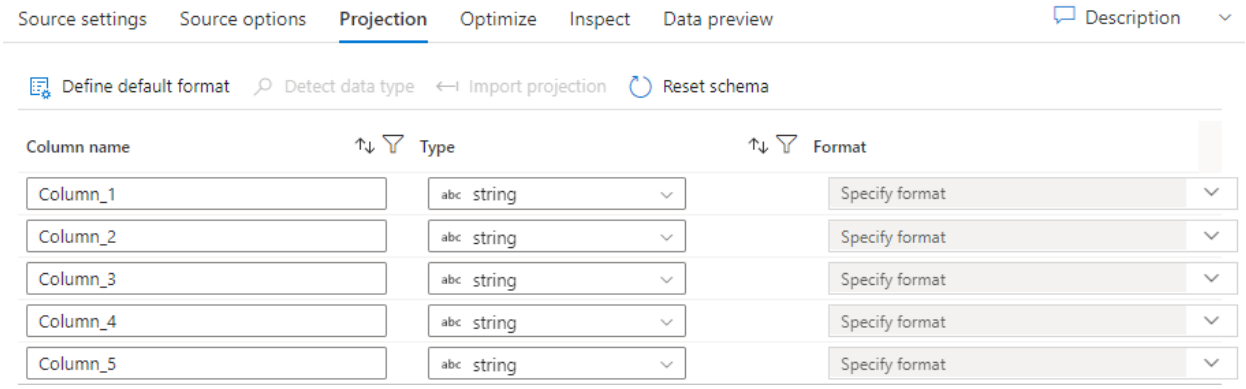
> **NOTE**
> When debug mode is turned on, the row limit configuration in debug settings will overwrite the sampling setting in the source during data preview.

Source options

The **Source options** tab contains settings specific to the connector and format chosen. For more information and examples, see the relevant connector documentation.

Projection

Like schemas in datasets, the projection in a source defines the data columns, types, and formats from the source data. For most dataset types, such as SQL and Parquet, the projection in a source is fixed to reflect the schema defined in a dataset. When your source files aren't strongly typed (for example, flat .csv files rather than Parquet files), you can define the data types for each field in the source transformation.



If your text file has no defined schema, select **Detect data type** so that the service will sample and infer the data types. Select **Define default format** to autodetect the default data formats.

Reset schema resets the projection to what is defined in the referenced dataset.

Overwrite schema allows you to modify the projected data types here the source, overwriting the schema-defined data types. You can alternatively modify the column data types in a downstream derived-column transformation. Use a select transformation to modify the column names.

Import schema

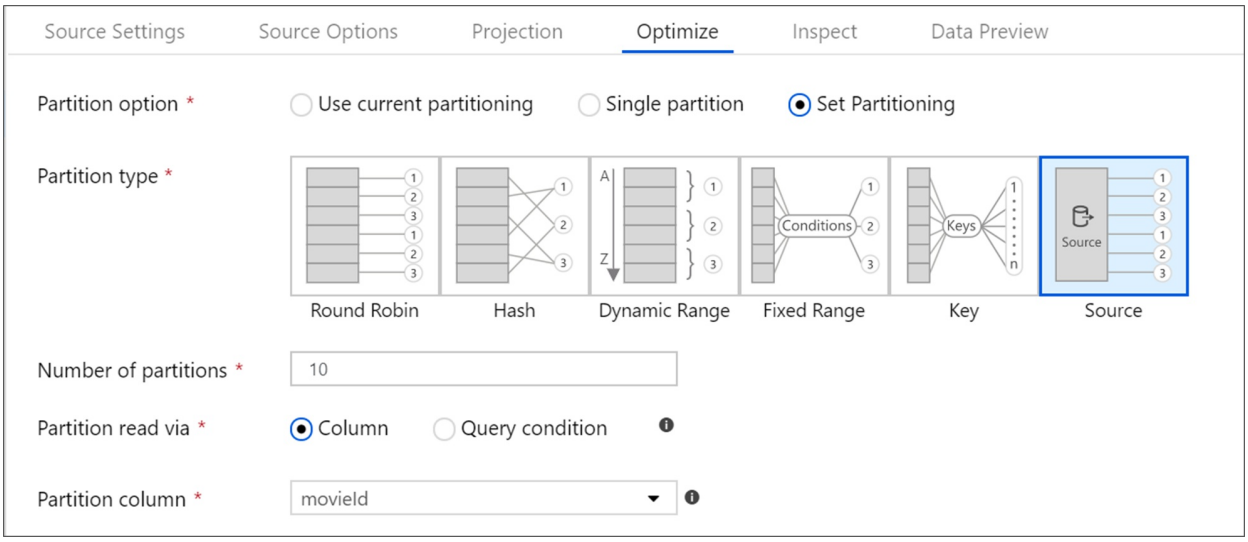
Select the **Import schema** button on the **Projection** tab to use an active debug cluster to create a schema projection. It's available in every source type. Importing the schema here will override the projection defined in the dataset. The dataset object won't be changed.

Importing schema is useful in datasets like Avro and Azure Cosmos DB that support complex data structures that don't require schema definitions to exist in the dataset. For inline datasets, importing schema is the only way to reference column metadata without schema drift.

Optimize the source transformation

The **Optimize** tab allows for editing of partition information at each transformation step. In most cases, **Use current partitioning** will optimize for the ideal partitioning structure for a source.

If you're reading from an Azure SQL Database source, custom **Source** partitioning will likely read data the fastest. The service will read large queries by making connections to your database in parallel. This source partitioning can be done on a column or by using a query.



For more information on optimization within mapping data flow, see the Optimize tab.

Next steps

Begin building your data flow with a derived-column transformation and a select transformation.

Surrogate key transformation in mapping data flow

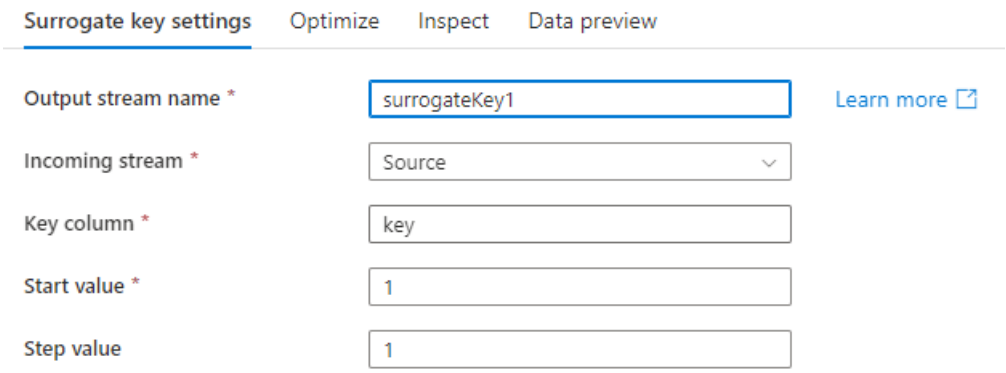
8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

Use the surrogate key transformation to add an incrementing key value to each row of data. This is useful when designing dimension tables in a star schema analytical data model. In a star schema, each member in your dimension tables requires a unique key that is a non-business key.

Configuration

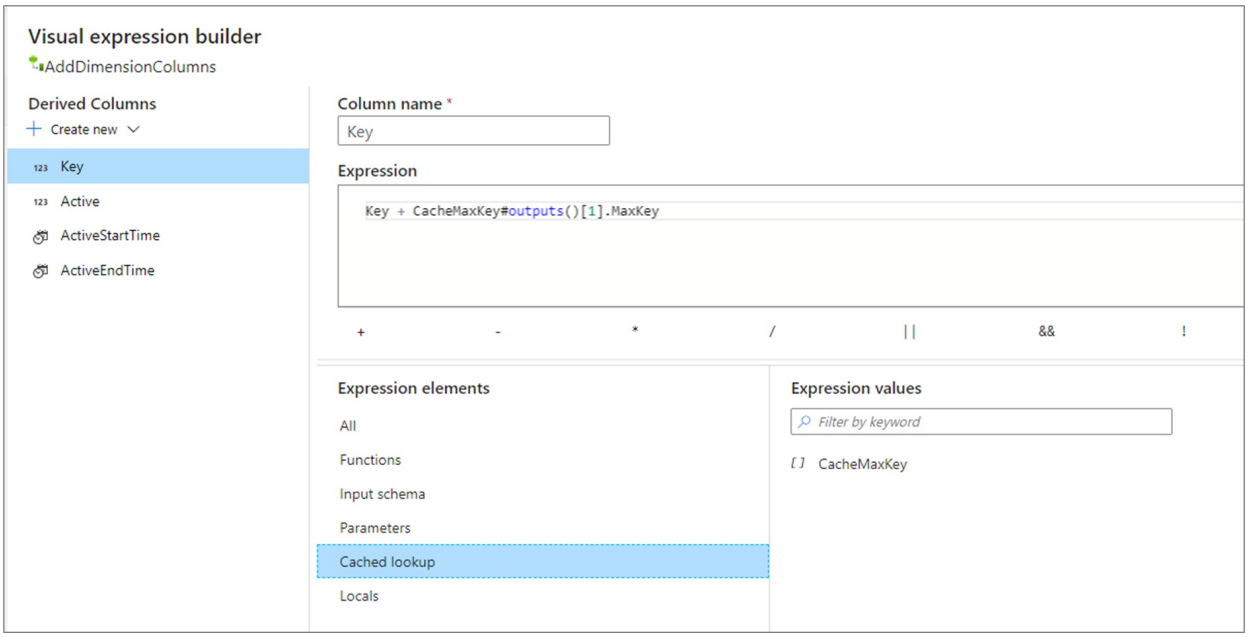


Key column: The name of the generated surrogate key column.

Start value: The lowest key value that will be generated.

Increment keys from existing sources

To start your sequence from a value that exists in a source, we recommend to use a cache sink to save that value and use a derived column transformation to add the two values together. Use a cached lookup to get the output and append it to the generated key. For more information, learn about cache sinks and cached lookups.

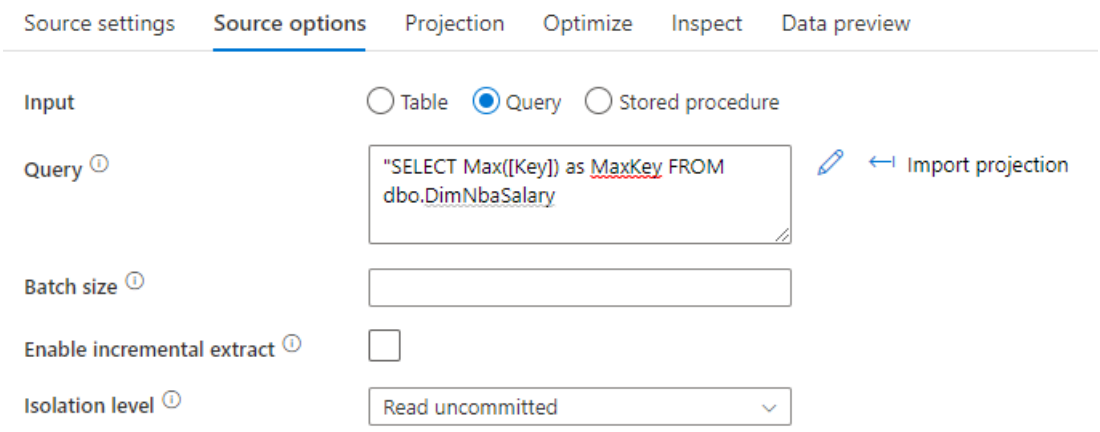


Increment from existing maximum value

To seed the key value with the previous max, there are two techniques that you can use based on where your source data is.

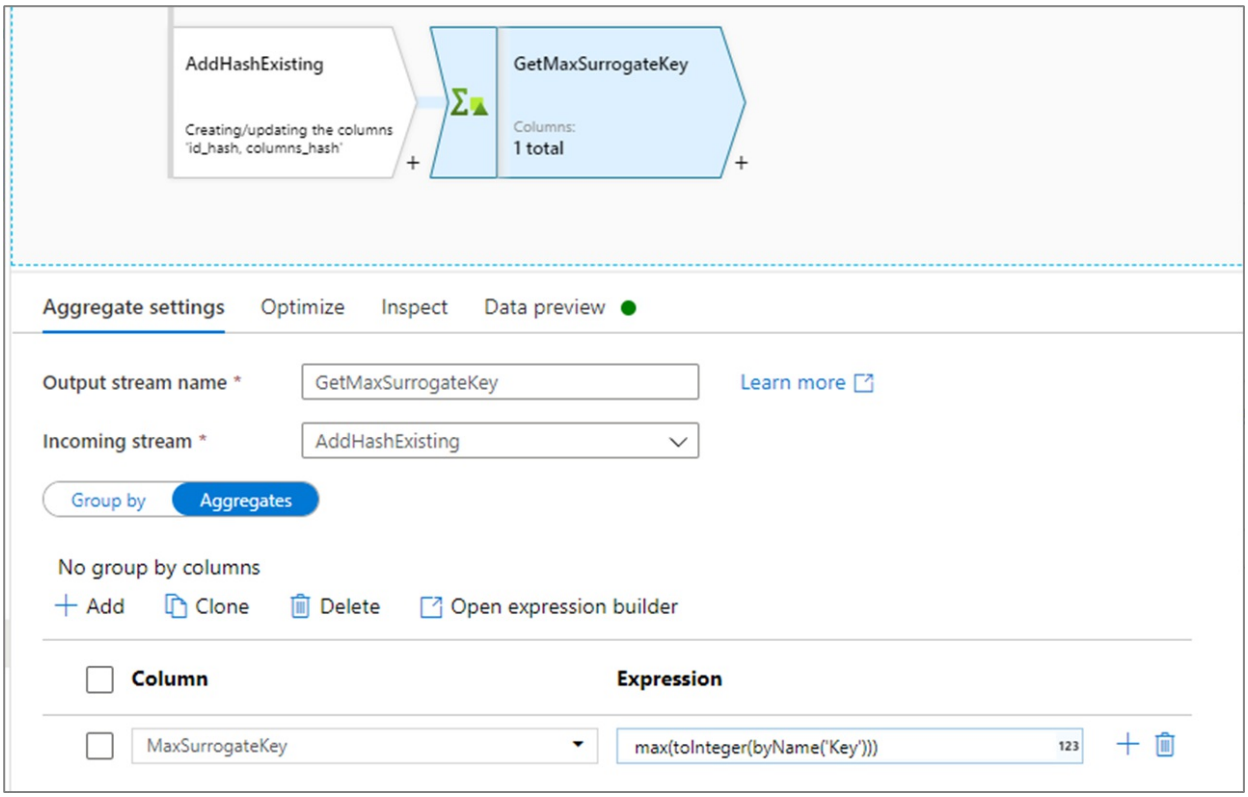
Database sources

Use a SQL query option to select MAX() from your source. For example, `Select MAX(<surrogateKeyName>) as maxval from <sourceTable>`.



File sources

If your previous max value is in a file, use the `max()` function in the aggregate transformation to get the previous max value:



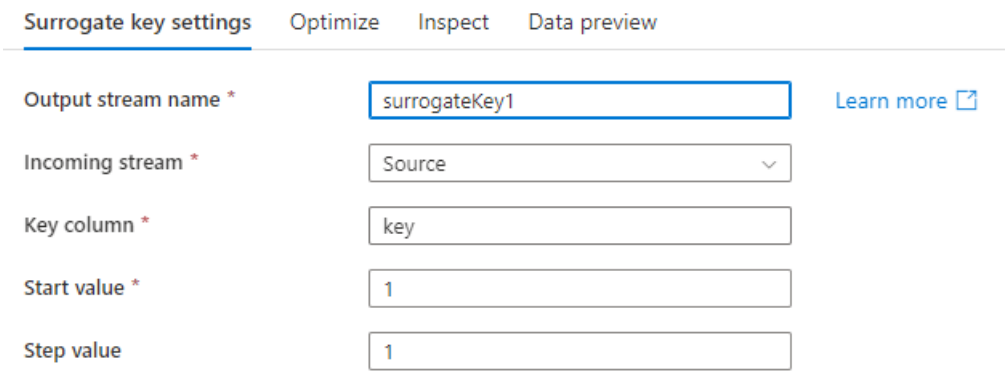
In both cases, you will need to write to a cache sink and lookup the value.

Data flow script

Syntax

```
<incomingStream>
    keyGenerate(
        output(<surrogateColumnName> as long),
        startAt: <number>L
    ) ~> <surrogateKeyTransformationName>
```

Example



The data flow script for the above surrogate key configuration is in the code snippet below.

```
AggregateDayStats
    keyGenerate(
        output(key as long),
        startAt: 1L
    ) ~> SurrogateKey1
```

Next steps

These examples use the Join and Derived Column transformations.

Union transformation in mapping data flow

8/19/2022 • 2 minutes to read • Edit Online

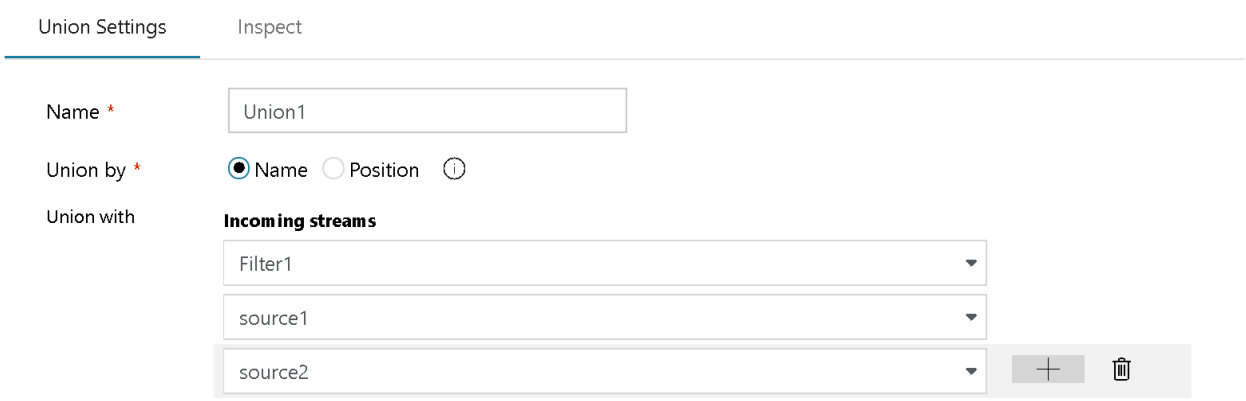
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

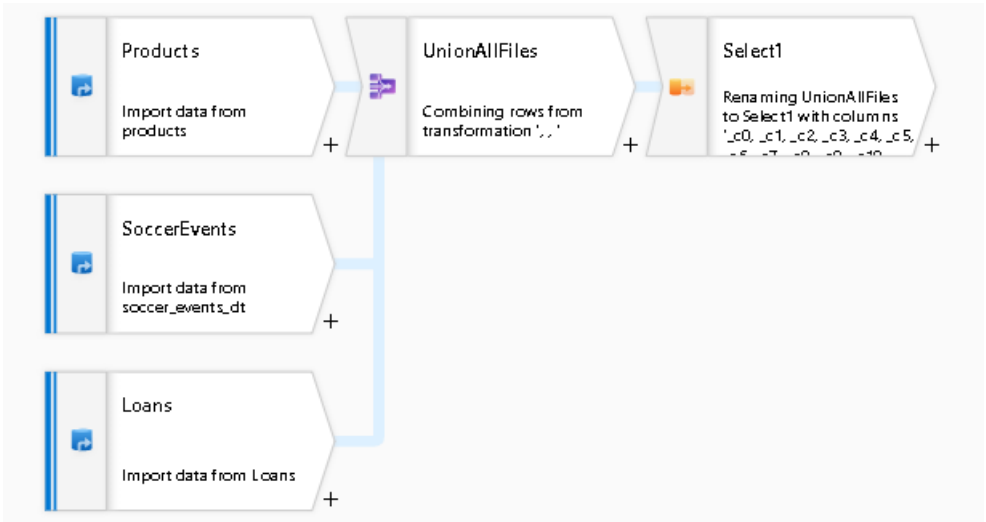
Union will combine multiple data streams into one, with the SQL Union of those streams as the new output from the Union transformation. All of the schema from each input stream will be combined inside of your data flow, without needing to have a join key.

You can combine n-number of streams in the settings table by selecting the "+" icon next to each configured row, including both source data as well as streams from existing transformations in your data flow.

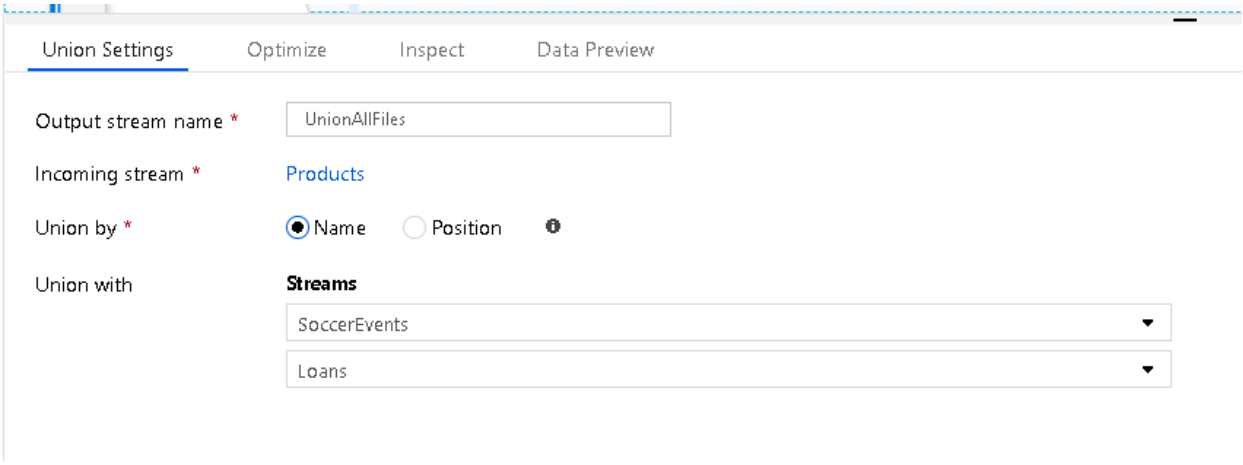
Here is a short video walk-through of the union transformation in the mapping data flow:



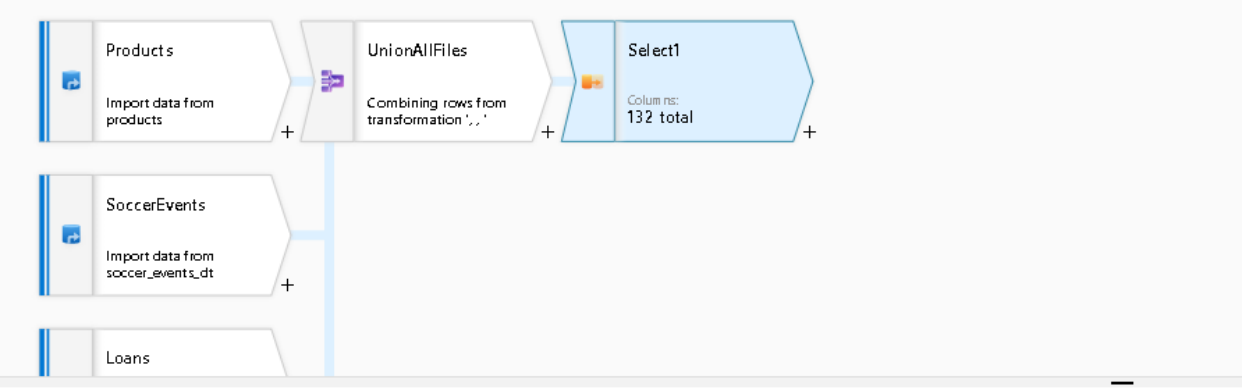
In this case, you can combine disparate metadata from multiple sources (in this example, three different source files) and combine them into a single stream:

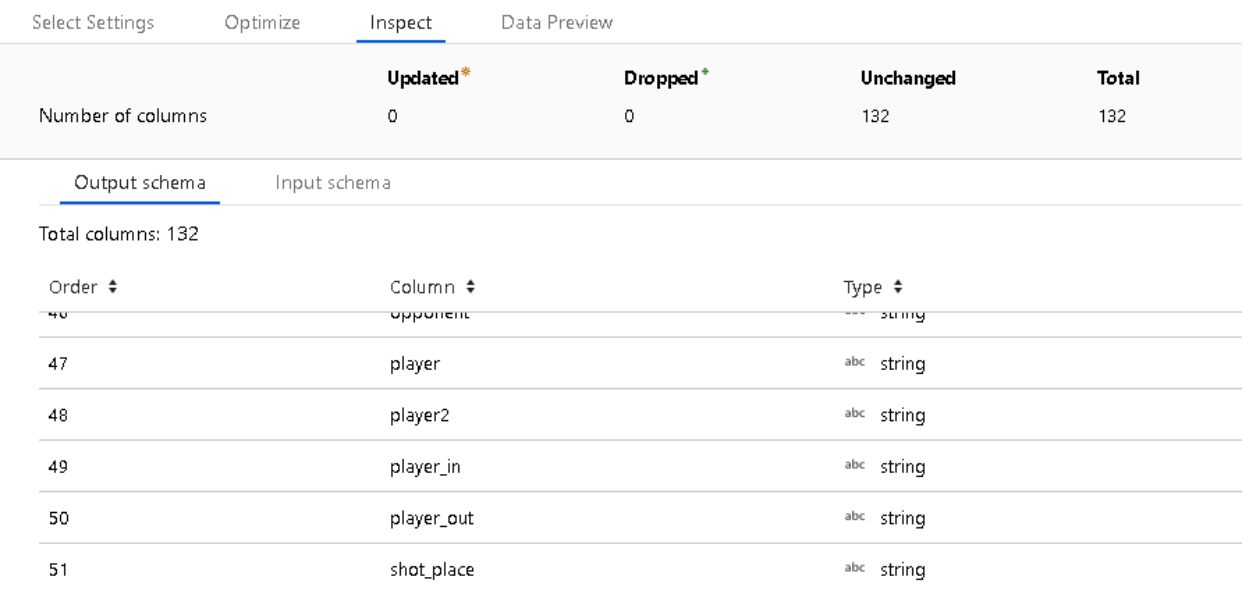


To achieve this, add additional rows in the Union Settings by including all source you wish to add. There is no need for a common lookup or join key:



If you set a Select transformation after your Union, you will be able to rename overlapping fields or fields that were not named from headerless sources. Click on "Inspect" to see the combine metadata with 132 total columns in this example from three different sources:

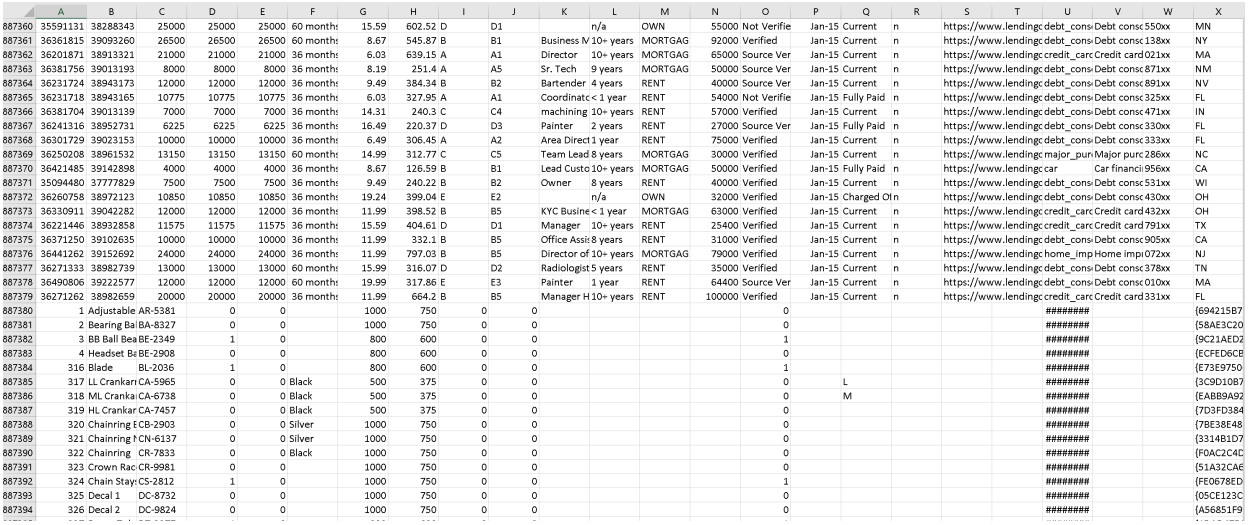




Name and position

When you choose "union by name", each column value will drop into the corresponding column from each source, with a new concatenated metadata schema.

If you choose "union by position", each column value will drop into the original position from each corresponding source, resulting in a new combined stream of data where the data from each source is added to the same stream:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
887360	35591131	38288343	25000	25000	25000	60 months	15.59	602.52	D	D1		n/a	OWN	55000	Not Verifie	Jan-15	Current	n	https://www.lendingc	debt_consi	Debt consc	550xx	MN	
887361	36361815	39093260	26500	26500	26500	60 months	8.67	545.87	B	B1	Business M	10+ years	MORTGAG	92000	Verified	Jan-15	Current	n	https://www.lendingc	debt_consi	Debt consc	138xx	NY	
887362	36201871	38913321	21000	21000	21000	36 months	6.03	639.15	A	A1	Director	10+ years	MORTGAG	65000	Source Ver	Jan-15	Current	n	https://www.lendingc	credit_carc	Credit card	021xx	MA	
887363	36381756	39013193	8000	8000	8000	36 months	8.19	251.4	A	A5	Sr. Tech	9 years	MORTGAG	50000	Source Ver	Jan-15	Current	n	https://www.lendingc	debt_consi	Debt consc	871xx	NM	
887364	36231724	38943173	12000	12000	12000	36 months	9.49	384.34	B	B2	Bartender	4 years	RENT	40000	Source Ver	Jan-15	Current	n	https://www.lendingc	debt_consi	Debt consc	891xx	NV	
887365	36231718	38943165	10775	10775	10775	36 months	6.03	327.95	A	A1	Coordinato	< 1 year	RENT	54000	Not Verifie	Jan-15	Fully Paid	n	https://www.lendingc	debt_consi	Debt consc	325xx	FL	
887366	36381704	39013139	7000	7000	7000	36 months	14.31	240.3	C	C4	machining	10+ years	RENT	57000	Verified	Jan-15	Current	n	https://www.lendingc	debt_consi	Debt consc	471xx	IN	
887367	36241316	38952731	6225	6225	6225	36 months	16.49	220.37	D	D3	Painter	2 years	RENT	27000	Source Ver	Jan-15	Fully Paid	n	https://www.lendingc	debt_consi	Debt consc	330xx	FL	
887368	36301729	39023153	10000	10000	10000	36 months	6.49	306.45	A	A2	Area Direct	1 year	RENT	75000	Verified	Jan-15	Current	n	https://www.lendingc	debt_consi	Debt consc	333xx	FL	
887369	36250208	38961532	13150	13150	13150	60 months	14.99	312.77	C	C5	Team Lead	8 years	MORTGAG	30000	Verified	Jan-15	Current	n	https://www.lendingc	major_pur	Major purc	286xx	NC	
887370	36421485	39142898	4000	4000	4000	36 months	8.67	126.59	B	B1	Lead Custo	10+ years	MORTGAG	50000	Verified	Jan-15	Fully Paid	n	https://www.lendingc	car	Car financi	956xx	CA	
887371	35094480	37777829	7500	7500	7500	36 months	9.49	240.22	B	B2	Owner	8 years	RENT	40000	Verified	Jan-15	Current	n	https://www.lendingc	debt_consi	Debt consc	531xx	WI	
887372	36260758	38972123	10850	10850	10850	36 months	19.24	399.04	E	E2		n/a	OWN	32000	Verified	Jan-15	Charged Of	n	https://www.lendingc	debt_consi	Debt consc	430xx	OH	
887373	36330911	39042282	12000	12000	12000	36 months	11.99	398.52	B	B5	KYC Busine	< 1 year	MORTGAG	63000	Verified	Jan-15	Current	n	https://www.lendingc	credit_carc	Credit card	432xx	OH	
887374	36221446	38932858	11575	11575	11575	36 months	15.59	404.61	D	D1	Manager	10+ years	RENT	25400	Verified	Jan-15	Current	n	https://www.lendingc	credit_carc	Credit card	791xx	TX	
887375	36371250	39102635	10000	10000	10000	36 months	11.99	332.1	B	B5	Office Assi	8 years	RENT	31000	Verified	Jan-15	Current	n	https://www.lendingc	debt_consi	Debt consc	905xx	CA	
887376	36441262	39152692	24000	24000	24000	36 months	11.99	797.03	B	B5	Director of	10+ years	MORTGAG	79000	Verified	Jan-15	Current	n	https://www.lendingc	home_imp	Home impi	072xx	NJ	
887377	36271333	38982739	13000	13000	13000	60 months	15.99	316.07	D	D2	Radiologist	5 years	RENT	35000	Verified	Jan-15	Current	n	https://www.lendingc	debt_consi	Debt consc	378xx	TN	
887378	36490806	39222577	12000	12000	12000	60 months	19.99	317.86	E	E3	Painter	1 year	RENT	64400	Source Ver	Jan-15	Current	n	https://www.lendingc	debt_consi	Debt consc	010xx	MA	
887379	36271262	38982659	20000	20000	20000	36 months	11.99	664.2	B	B5	Manager H	10+ years	RENT	100000	Verified	Jan-15	Current	n	https://www.lendingc	credit_carc	Credit card	331xx	FL	
887380	1	Adjustable	AR-5381	0	0		1000	750	0	0					0						########			{694215B7
887381	2	Bearing Ba	BA-8327	0	0		1000	750	0	0					0						########			{58AE3C20
887382	3	BB Ball Bea	BE-2349	1	0		800	600	0	0					1						########			{9C21AED2
887383	4	Headset Ba	BE-2908	0	0		800	600	0	0					0						########			{ECFED6CE
887384	316	Blade	BL-2036	1	0		800	600	0	0					1						########			{E73E9750
887385	317	LL Crankar	CA-5965	0	0	Black	500	375	0	0					0		L				########			{3C9D10B7
887386	318	ML Cranka	CA-6738	0	0	Black	500	375	0	0					0		M				########			{EABB9A92
887387	319	HL Crankar	CA-7457	0	0	Black	500	375	0	0					0						########			{7D3FD384
887388	320	Chainring E	CB-2903	0	0	Silver	1000	750	0	0					0						########			{7BE38E48
887389	321	Chainring I	CN-6137	0	0	Silver	1000	750	0	0					0						########			{3314B1D7
887390	322	Chainring	CR-7833	0	0	Black	1000	750	0	0					0						########			{F0AC2C4D
887391	323	Crown Rac	CR-9981	0	0		1000	750	0	0					0						########			{51A32CA6
887392	324	Chain Stay	CS-2812	1	0		1000	750	0	0					1						########			{FE0678ED
887393	325	Decal 1	DC-8732	0	0		1000	750	0	0					0						########			{05CE123C
887394	326	Decal 2	DC-9824	0	0		1000	750	0	0					0						########			{A56851F9

Next steps

Explore similar transformations including Join and Exists.

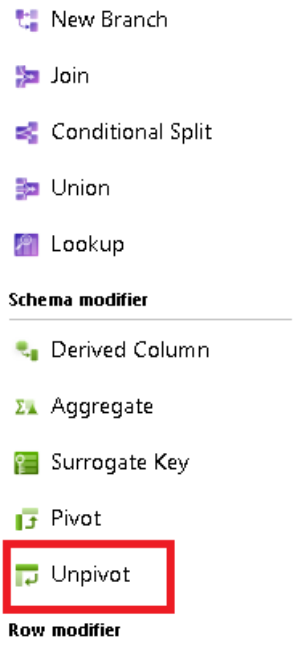
Unpivot transformation in mapping data flow

8/19/2022 • 2 minutes to read • Edit Online

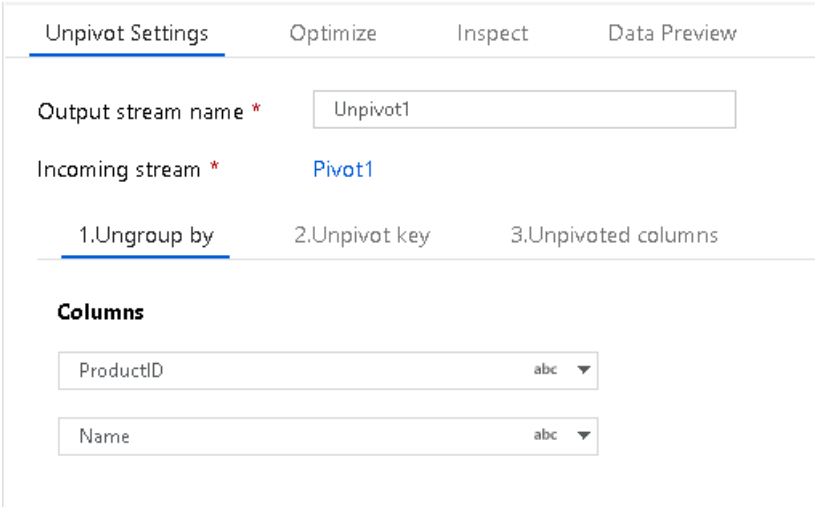
APPLIES TO: Azure Data Factory Azure Synapse Analytics

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

Use Unpivot in a mapping data flow as a way to turn an unnormalized dataset into a more normalized version by expanding values from multiple columns in a single record into multiple records with the same values in a single column.

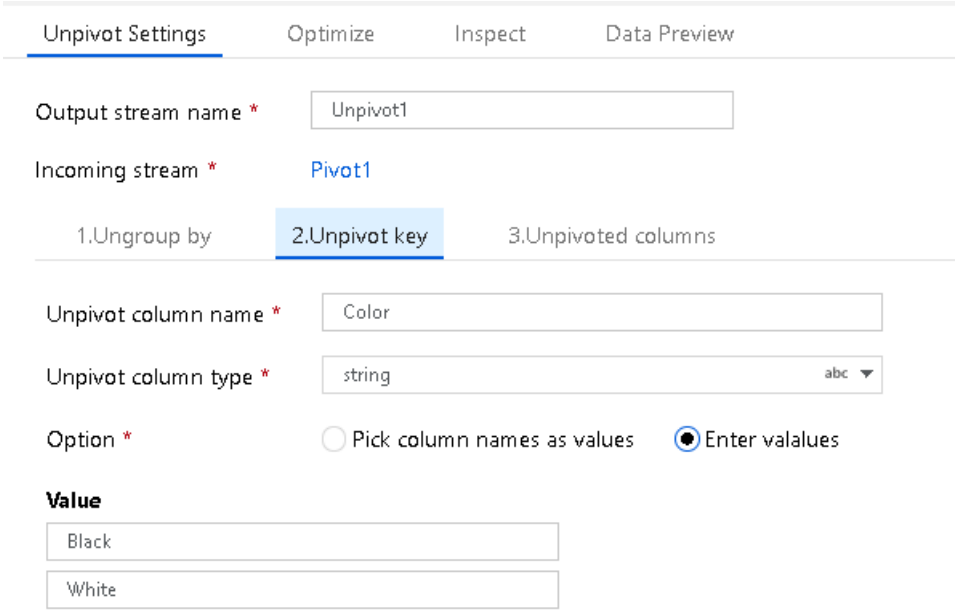


Ungroup By



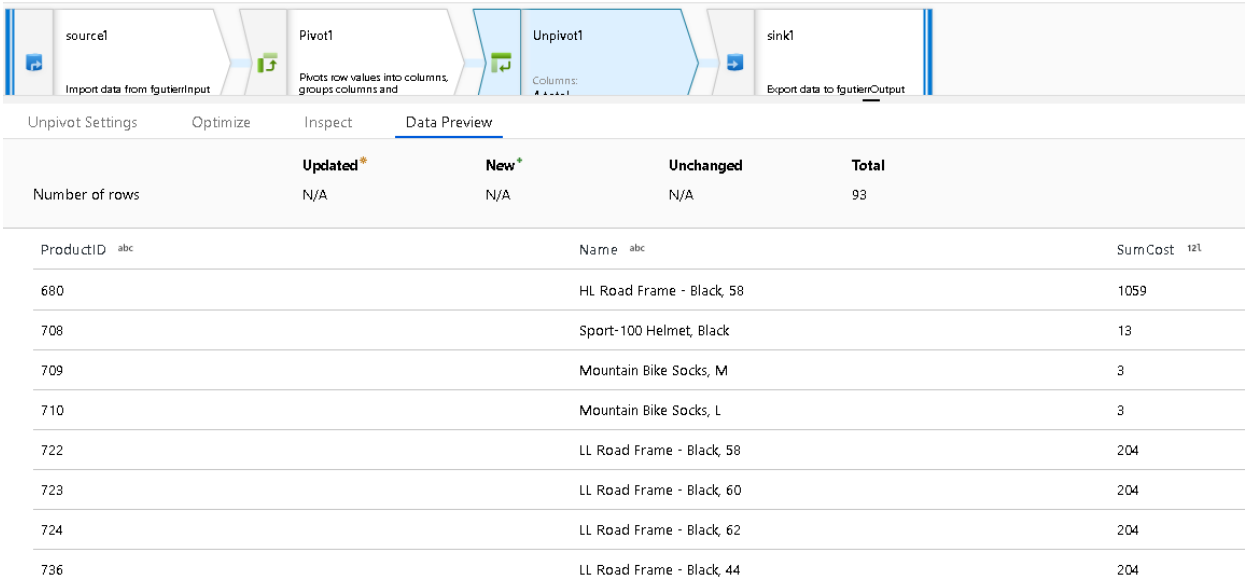
First, set the columns that you wish to ungroup by for your unpivot aggregation. Set one or more columns for ungrouping with the + sign next to the column list.

Unpivot Key



The Unpivot Key is the column that the service will pivot from column to row. By default, each unique value in the dataset for this field will pivot to a row. However, you can optionally enter the values from the dataset that you wish to pivot to row values.

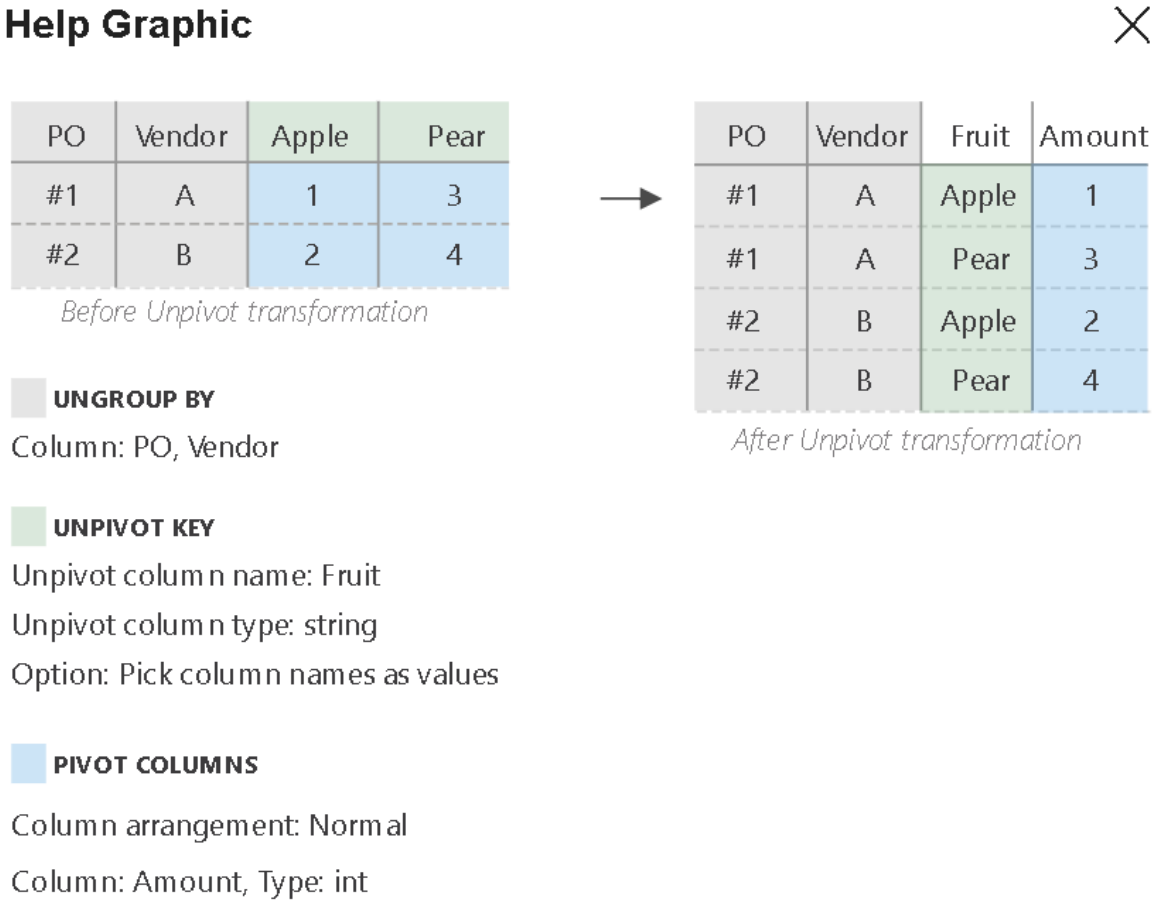
Unpivoted Columns



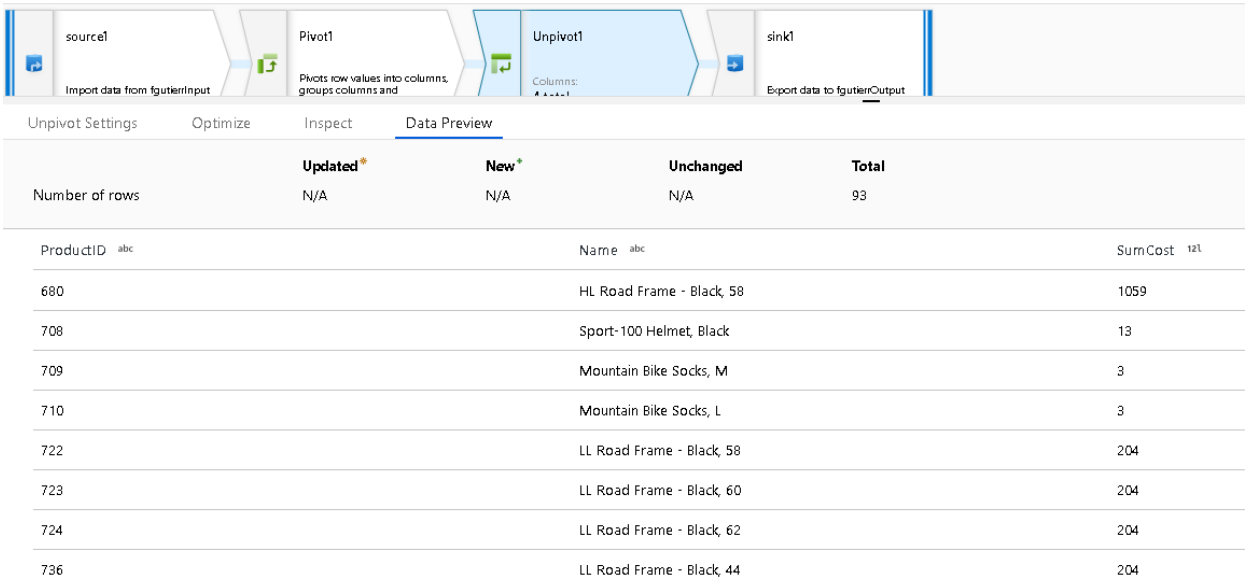
Lastly, choose the column name for storing the values for unpivoted columns that are transformed into rows.

(Optional) You can drop rows with Null values.

For instance, SumCost is the column name that is chosen in the example shared above.



Setting the Column Arrangement to "Normal" will group together all of the new unpivoted columns from a single value. Setting the columns arrangement to "Lateral" will group together new unpivoted columns generated from an existing column.



The final unpivoted data result set shows the column totals now unpivoted into separate row values.

Next steps

Use the Pivot transformation to pivot rows to columns.

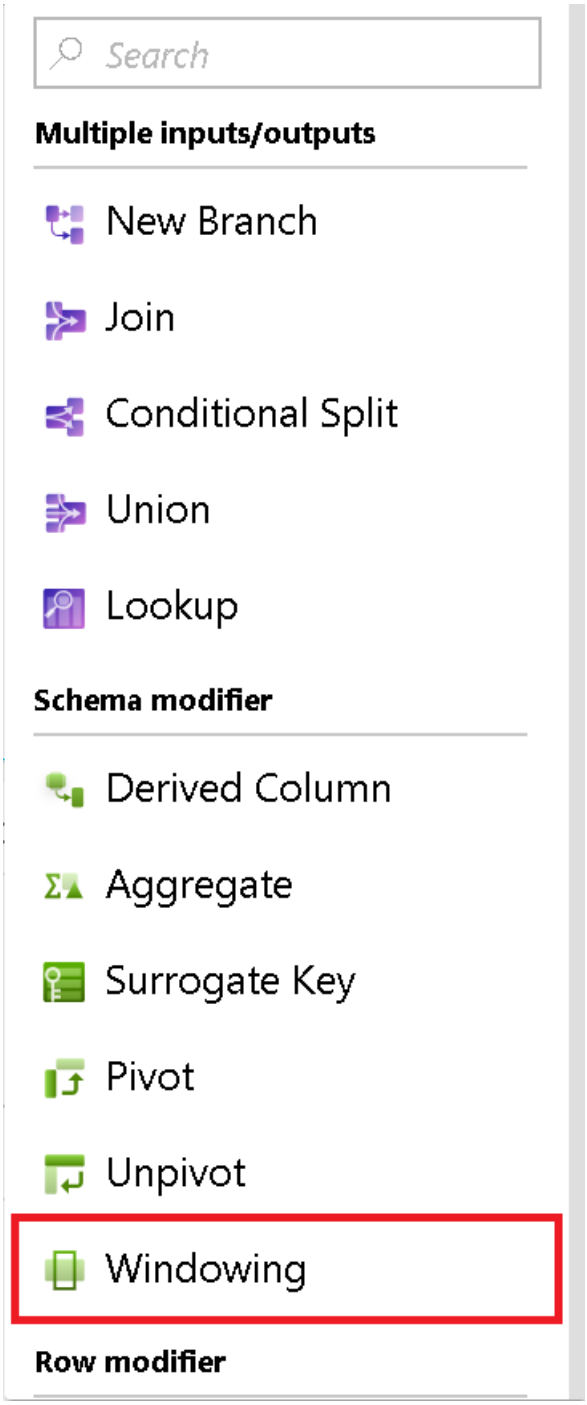
Window transformation in mapping data flow

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

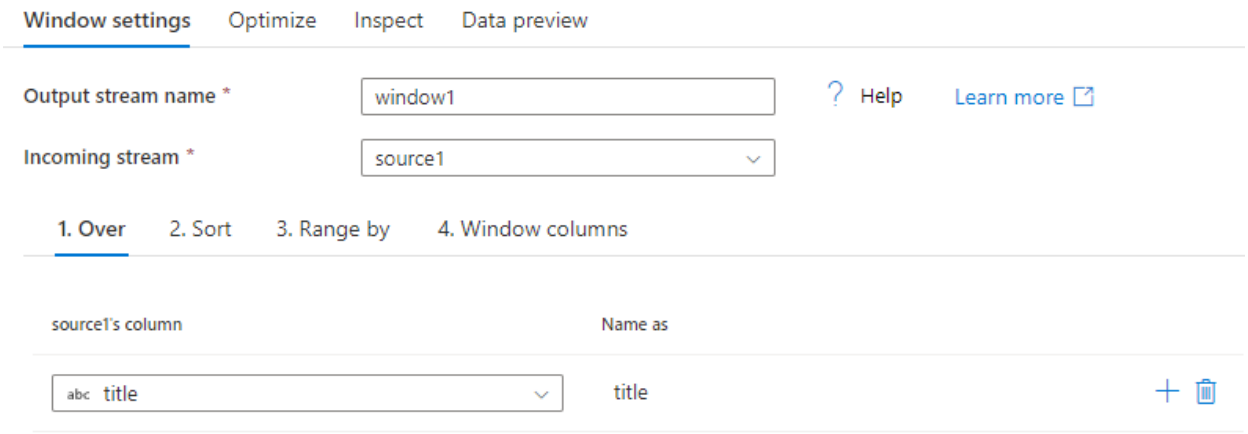
Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article Transform data using a mapping data flow.

The Window transformation is where you will define window-based aggregations of columns in your data streams. In the Expression Builder, you can define different types of aggregations that are based on data or time windows (SQL OVER clause) such as LEAD, LAG, NTILE, CUMEDIST, RANK, etc.). A new field will be generated in your output that includes these aggregations. You can also include optional group-by fields.



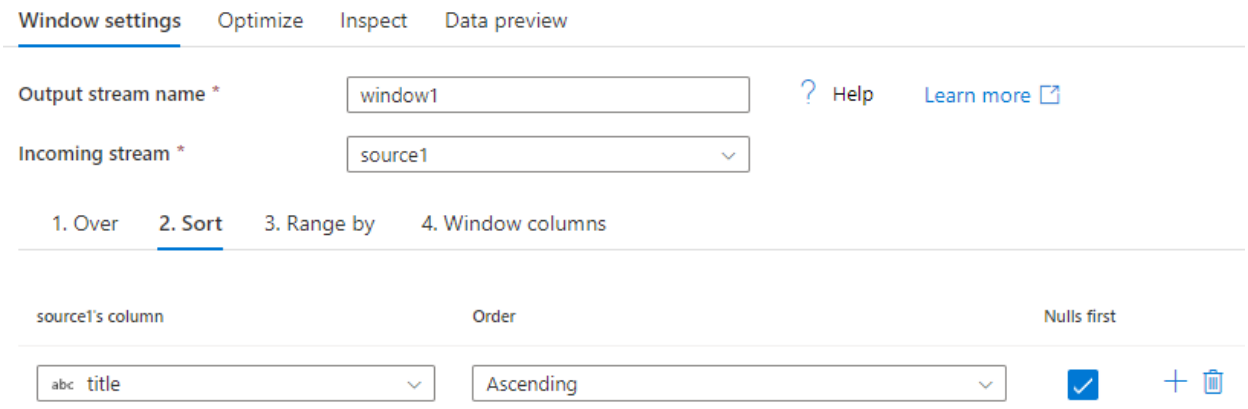
Over

Set the partitioning of column data for your window transformation. The SQL equivalent is the `Partition By` in the Over clause in SQL. If you wish to create a calculation or create an expression to use for the partitioning, you can do that by hovering over the column name and select "computed column".



Sort

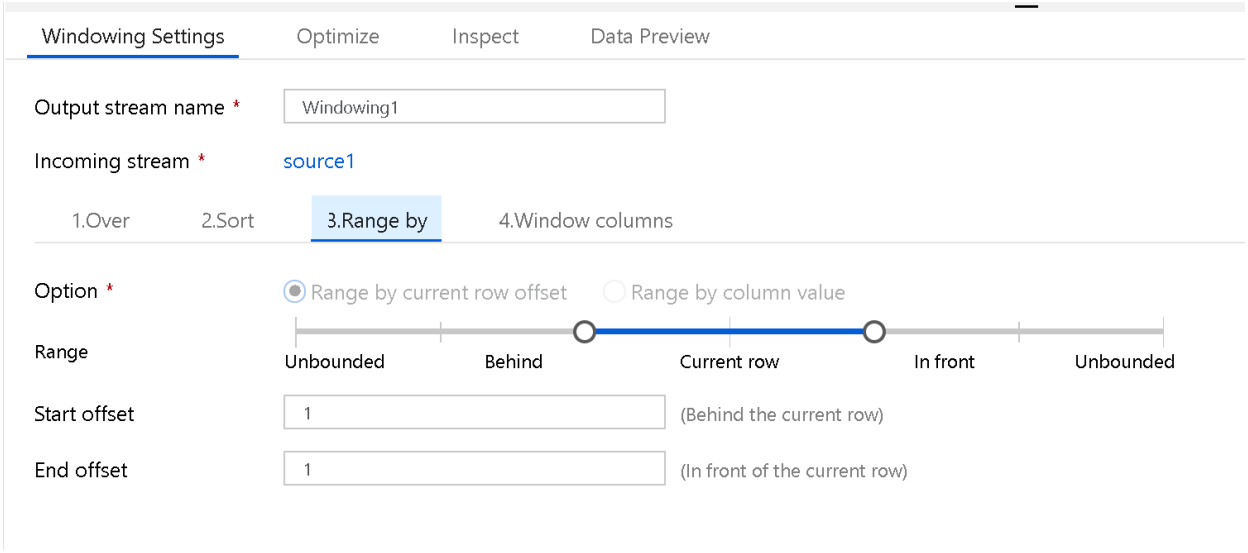
Another part of the Over clause is setting the `Order By`. This will set the data sort ordering. You can also create an expression for a calculate value in this column field for sorting.



Range By

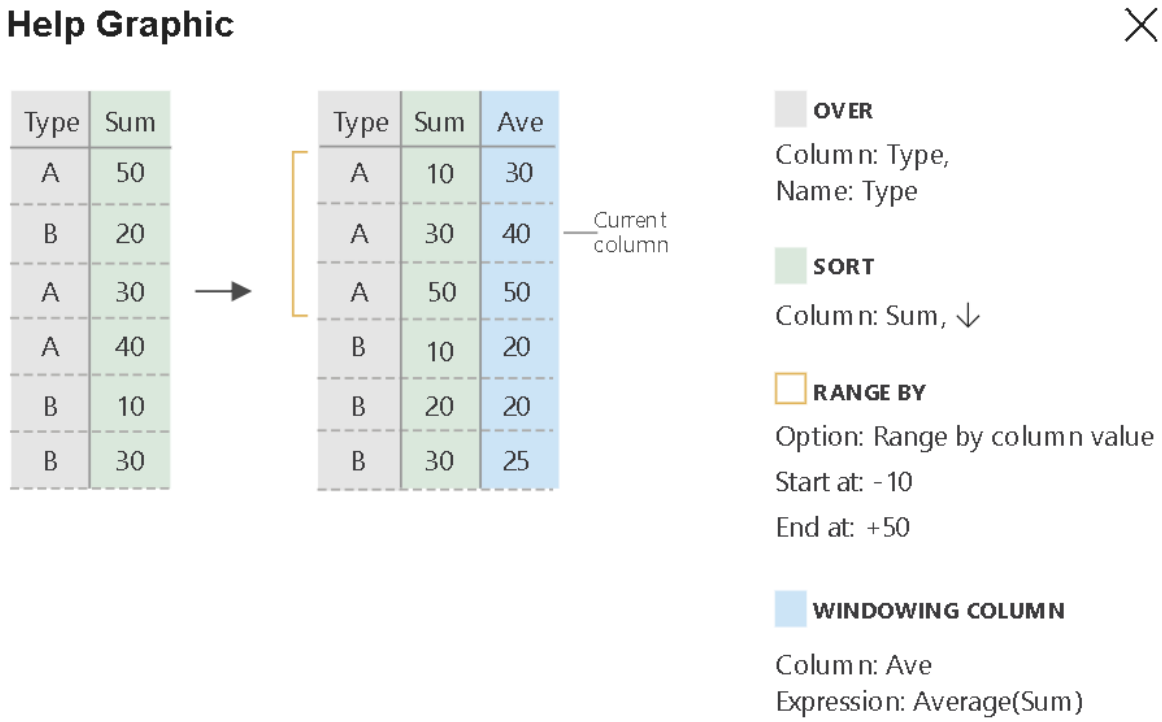
Next, set the window frame as Unbounded or Bounded. To set an unbounded window frame, set the slider to Unbounded on both ends. If you choose a setting between Unbounded and Current Row, then you must set the Offset start and end values. Both values will be positive integers. You can use either relative numbers or values from your data.

The window slider has two values to set: the values before the current row and the values after the current row. The Start and End offset matches the two selectors on the slider.



Window columns

Lastly, use the Expression Builder to define the aggregations you wish to use with the data windows such as RANK, COUNT, MIN, MAX, DENSE RANK, LEAD, LAG, etc.



The full list of aggregation and analytical functions available for you to use in the Data Flow Expression Language via the Expression Builder are listed in Data transformation expressions in mapping data flow.

Next steps

If you are looking for a simple group-by aggregation, use the Aggregate transformation

Parameterize linked services in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

You can now parameterize a linked service and pass dynamic values at run time. For example, if you want to connect to different databases on the same logical SQL server, you can now parameterize the database name in the linked service definition. This prevents you from having to create a linked service for each database on the logical SQL server. You can parameterize other properties in the linked service definition as well - for example, *User name*.

You can use the UI in the Azure portal or a programming interface to parameterize linked services.

> **TIP**
> We recommend not to parameterize passwords or secrets. Store all secrets in Azure Key Vault instead, and parameterize the *Secret Name*.

> **NOTE**
> There is open bug to use "-" in parameter names, we recommend to use names without "-" until the bug is resolved.

For a seven-minute introduction and demonstration of this feature, watch the following video:

Supported linked service types

All the linked service types are supported for parameterization.

Natively supported in UI: When authoring linked service on UI, the service provides built-in parameterization experience for the following types of linked services. In linked service creation/edit blade, you can find options to new parameters and add dynamic content. Refer to UI experience.

- Amazon Redshift
- Amazon S3
- Amazon S3 Compatible Storage
- Azure Blob Storage
- Azure Cosmos DB (SQL API)
- Azure Data Explorer
- Azure Data Lake Storage Gen1
- Azure Data Lake Storage Gen2
- Azure Database for MySQL
- Azure Database for PostgreSQL
- Azure Databricks
- Azure File Storage

- Azure Function
- Azure Key Vault
- Azure SQL Database
- Azure SQL Managed Instance
- Azure Synapse Analytics
- Azure Table Storage
- File System
- FTP
- Generic HTTP
- Generic REST
- Google AdWords
- Informix
- Microsoft Access
- MySQL
- OData
- ODBC
- Oracle
- Oracle Cloud Storage
- PostgreSQL
- Salesforce
- Salesforce Service Cloud
- SAP ODP
- SFTP
- SharePoint Online List
- Snowflake
- SQL Server

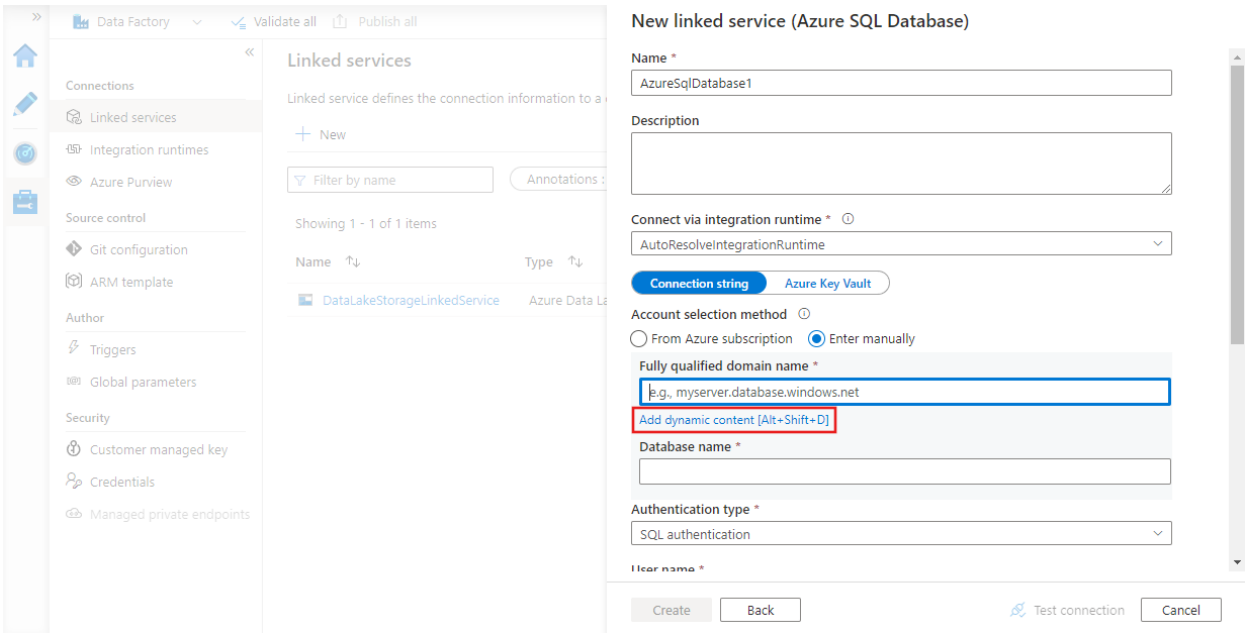
Advanced authoring: For other linked service types that are not in above list, you can parameterize the linked service by editing the JSON on UI:

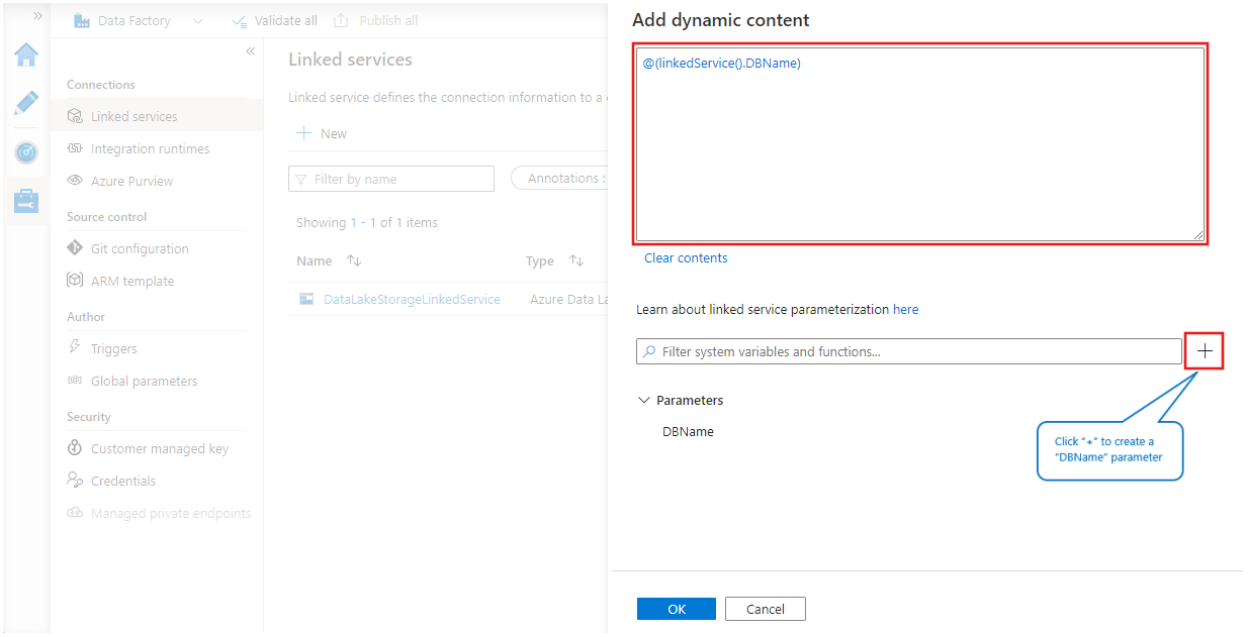
- In linked service creation/edit blade -> expand "Advanced" at the bottom -> check "Specify dynamic contents in JSON format" checkbox -> specify the linked service JSON payload.
- Or, after you create a linked service without parameterization, in Management hub -> Linked services -> find the specific linked service -> click "Code" (button "{}") to edit the JSON.

Refer to the JSON sample to add `parameters` section to define parameters and reference the parameter using `@{linkedService().paramName}`.

UI Experience

- Azure Data Factory
- Azure Synapse





JSON

```
{
 "name": "AzureSqlDatabase",
 "properties": {
  "type": "AzureSqlDatabase",
  "typeProperties": {
   "connectionString":
"Server=tcp:myserver.database.windows.net,1433;Database=@{linkedService().DBName};User
ID=user;Password=fake;Trusted_Connection=False;Encrypt=True;Connection Timeout=30"
  },
  "connectVia": null,
  "parameters": {
   "DBName": {
    "type": "String"
   }
  }
 }
}
```

Expressions and functions in Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 54 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides details about expressions and functions supported by Azure Data Factory and Azure Synapse Analytics.

Expressions

JSON values in the definition can be literal or expressions that are evaluated at runtime. For example:

```
"name": "value"
```

or

```
"name": "@pipeline().parameters.password"
```

Expressions can appear anywhere in a JSON string value and always result in another JSON value. If a JSON value is an expression, the body of the expression is extracted by removing the at-sign (@). If a literal string is needed that starts with @, it must be escaped by using @@. The following examples show how expressions are evaluated.

JSON VALUE	RESULT
"parameters"	The characters 'parameters' are returned.
"parameters[1]"	The characters 'parameters[1]' are returned.
"@@"	A 1 character string that contains '@' is returned.
" @"	A 2 character string that contains ' @' is returned.

Expressions can also appear inside strings, using a feature called *string interpolation* where expressions are wrapped in `@{ ... }`. For example:

```
"name" : "First Name: @{pipeline().parameters.firstName} Last Name: @{pipeline().parameters.lastName}"
```

Using string interpolation, the result is always a string. Say I have defined `myNumber` as `42` and `myString` as `foo`:

JSON VALUE	RESULT
"@pipeline().parameters.myString"	Returns `foo` as a string.
"@{pipeline().parameters.myString}"	Returns `foo` as a string.
"@pipeline().parameters.myNumber"	Returns `42` as a *number*.

JSON VALUE	RESULT
"@{pipeline().parameters.myNumber}"	Returns `42` as a *string*.
"Answer is: @{pipeline().parameters.myNumber}"	Returns the string `Answer is: 42`.
"@concat('Answer is: ', string(pipeline().parameters.myNumber))"	Returns the string `Answer is: 42`
"Answer is: @@{pipeline().parameters.myNumber}"	Returns the string `Answer is: @{pipeline().parameters.myNumber}`.

In the control flow activities like ForEach activity, you can provide an array to be iterated over for the property items and use @item() to iterate over a single enumeration in ForEach activity. For example, if items is an array: [1, 2, 3], @item() returns 1 in the first iteration, 2 in the second iteration, and 3 in the third iteration. You can also use @range(0,10) like expression to iterate ten times starting with 0 ending with 9.

You can use @activity('activity name') to capture output of activity and make decisions. Consider a web activity called Web1. For placing the output of the first activity in the body of the second, the expression generally looks like: @activity('Web1').output or @activity('Web1').output.data or something similar depending upon what the output of the first activity looks like.

Examples

Complex expression example

The below example shows a complex example that references a deep sub-field of activity output. To reference a pipeline parameter that evaluates to a sub-field, use [] syntax instead of dot(.) operator (as in case of subfield1 and subfield2), as part of an activity output.

```
@activity('*activityName*').output.*subfield1*.*subfield2*[pipeline().parameters.*subfield3*].*subfield4*
```

Creating files dynamically and naming them is common pattern. Let us explore few dynamic file naming examples.

1. Append Date to a filename: `@concat('Test_', formatDateTime(utcnow(), 'yyyy-dd-MM'))`
2. Append DateTime in customer timezone : `@concat('Test_', convertFromUtc(utcnow(), 'Pacific Standard Time'))`
3. Append Trigger Time : `@concat('Test_', pipeline().TriggerTime)`
4. Output a custom filename in a Mapping Data Flow when outputting to a single file with date : `'Test_' + toString(currentDate()) + '.csv'`

In above cases, 4 dynamic filenames are created starting with Test_.

Dynamic content editor

Dynamic content editor automatically escapes characters in your content when you finish editing. For example, the following content in content editor is a string interpolation with two expression functions.

```
{
  "type": "@{if(equals(1, 2), 'Blob', 'Table' )}",
  "name": "@{toUpper('myData')}"
}
```

Dynamic content editor converts above content to expression

`"{ \n \"type\": \"@{if(equals(1, 2), 'Blob', 'Table' )}\",\n \"name\": \"@{toUpper('myData')}\"\n}"` . The result of this expression is a JSON format string showed below.

```
{
  "type": "Table",
  "name": "MYDATA"
}
```

A dataset with a parameter

In the following example, the BlobDataset takes a parameter named **path**. Its value is used to set a value for the **folderPath** property by using the expression: `dataset().path` .

```
{
    "name": "BlobDataset",
    "properties": {
        "type": "AzureBlob",
        "typeProperties": {
            "folderPath": "@dataset().path"
        },
        "linkedServiceName": {
            "referenceName": "AzureStorageLinkedService",
            "type": "LinkedServiceReference"
        },
        "parameters": {
            "path": {
                "type": "String"
            }
        }
    }
}
```

A pipeline with a parameter

In the following example, the pipeline takes **inputPath** and **outputPath** parameters. The **path** for the parameterized blob dataset is set by using values of these parameters. The syntax used here is: `pipeline().parameters.parametername` .

```
{
    "name": "Adfv2QuickStartPipeline",
    "properties": {
        "activities": [
            {
                "name": "CopyFromBlobToBlob",
                "type": "Copy",
                "inputs": [
                    {
                        "referenceName": "BlobDataset",
                        "parameters": {
                            "path": "@pipeline().parameters.inputPath"
                        },
                        "type": "DatasetReference"
                    }
                ],
                "outputs": [
                    {
                        "referenceName": "BlobDataset",
                        "parameters": {
                            "path": "@pipeline().parameters.outputPath"
                        },
                        "type": "DatasetReference"
                    }
                ],
                "typeProperties": {
                    "source": {
                        "type": "BlobSource"
                    },
                    "sink": {
                        "type": "BlobSink"
                    }
                }
            }
        ],
        "parameters": {
            "inputPath": {
                "type": "String"
            },
            "outputPath": {
                "type": "String"
            }
        }
    }
}
```

Replacing special characters

Dynamic content editor automatically escapes characters like double quote, backslash in your content when you finish editing. This causes trouble if you want to replace line feed or tab by using **\n**, **\t** in replace() function. You can of edit your dynamic content in code view to remove the extra \ in the expression, or you can follow below steps to replace special characters using expression language:

1. URL encoding against the original string value
2. Replace URL encoded string, for example, line feed (%0A), carriage return(%0D), horizontal tab(%09).
3. URL decoding

For example, variable *companyName* with a newline character in its value, expression `@uriComponentToString(replace(uriComponent(variables('companyName')), '%0A', ''))` can remove the newline character.

```
Contoso-
Corporation
```

Escaping single quote character

Expression functions use single quote for string value parameters. Use two single quotes to escape a ' character in string functions. For example, expression `@concat('Baba', '''s ', 'book store')` will return below result.

```
Baba's book store
```

Tutorial

This tutorial walks you through how to pass parameters between a pipeline and activity as well as between the activities. The tutorial specifically demonstrates steps for an Azure Data Factory although steps for a Synapse workspace are nearly equivalent but with a slightly different user interface.

Functions

You can call functions within expressions. The following sections provide information about the functions that can be used in an expression.

Date functions

DATE OR TIME FUNCTION	TASK
addDays	Add a number of days to a timestamp.
addHours	Add a number of hours to a timestamp.
addMinutes	Add a number of minutes to a timestamp.
addSeconds	Add a number of seconds to a timestamp.
addToTime	Add a number of time units to a timestamp. See also getFutureTime.
convertFromUtc	Convert a timestamp from Universal Time Coordinated (UTC) to the target time zone.
convertTimeZone	Convert a timestamp from the source time zone to the target time zone.
convertToUtc	Convert a timestamp from the source time zone to Universal Time Coordinated (UTC).
dayOfMonth	Return the day of the month component from a timestamp.
dayOfWeek	Return the day of the week component from a timestamp.
dayOfYear	Return the day of the year component from a timestamp.
formatDateTime	Return the timestamp as a string in optional format.

DATE OR TIME FUNCTION	TASK
getFutureTime	Return the current timestamp plus the specified time units. See also addToTime.
getPastTime	Return the current timestamp minus the specified time units. See also subtractFromTime.
startOfDay	Return the start of the day for a timestamp.
startOfHour	Return the start of the hour for a timestamp.
startOfMonth	Return the start of the month for a timestamp.
subtractFromTime	Subtract a number of time units from a timestamp. See also getPastTime.
ticks	Return the `ticks` property value for a specified timestamp.
utcNow	Return the current timestamp as a string.

String functions

To work with strings, you can use these string functions and also some collection functions. String functions work only on strings.

STRING FUNCTION	TASK
concat	Combine two or more strings, and return the combined string.
endsWith	Check whether a string ends with the specified substring.
guid	Generate a globally unique identifier (GUID) as a string.
indexOf	Return the starting position for a substring.
lastIndexOf	Return the starting position for the last occurrence of a substring.
replace	Replace a substring with the specified string, and return the updated string.
split	Return an array that contains substrings, separated by commas, from a larger string based on a specified delimiter character in the original string.
startsWith	Check whether a string starts with a specific substring.
substring	Return characters from a string, starting from the specified position.
toLower	Return a string in lowercase format.

STRING FUNCTION	TASK
toUpper	Return a string in uppercase format.
trim	Remove leading and trailing whitespace from a string, and return the updated string.

Collection functions

To work with collections, generally arrays, strings, and sometimes, dictionaries, you can use these collection functions.

COLLECTION FUNCTION	TASK
contains	Check whether a collection has a specific item.
empty	Check whether a collection is empty.
first	Return the first item from a collection.
intersection	Return a collection that has *only* the common items across the specified collections.
join	Return a string that has *all* the items from an array, separated by the specified character.
last	Return the last item from a collection.
length	Return the number of items in a string or array.
skip	Remove items from the front of a collection, and return *all the other* items.
take	Return items from the front of a collection.
union	Return a collection that has *all* the items from the specified collections.

Logical functions

These functions are useful inside conditions, they can be used to evaluate any type of logic.

LOGICAL COMPARISON FUNCTION	TASK
and	Check whether all expressions are true.
equals	Check whether both values are equivalent.
greater	Check whether the first value is greater than the second value.
greaterOrEquals	Check whether the first value is greater than or equal to the second value.

LOGICAL COMPARISON FUNCTION	TASK
if	Check whether an expression is true or false. Based on the result, return a specified value.
less	Check whether the first value is less than the second value.
lessOrEquals	Check whether the first value is less than or equal to the second value.
not	Check whether an expression is false.
or	Check whether at least one expression is true.

Conversion functions

These functions are used to convert between each of the native types in the language:

- string
- integer
- float
- boolean
- arrays
- dictionaries

CONVERSION FUNCTION	TASK
array	Return an array from a single specified input. For multiple inputs, see createArray.
base64	Return the base64-encoded version for a string.
base64ToBinary	Return the binary version for a base64-encoded string.
base64ToString	Return the string version for a base64-encoded string.
binary	Return the binary version for an input value.
bool	Return the Boolean version for an input value.
coalesce	Return the first non-null value from one or more parameters.
createArray	Return an array from multiple inputs.
dataUri	Return the data URI for an input value.
dataUriToBinary	Return the binary version for a data URI.
dataUriToString	Return the string version for a data URI.
decodeBase64	Return the string version for a base64-encoded string.

CONVERSION FUNCTION	TASK
decodeDataUri	Return the binary version for a data URI.
decodeUriComponent	Return a string that replaces escape characters with decoded versions.
encodeUriComponent	Return a string that replaces URL-unsafe characters with escape characters.
float	Return a floating point number for an input value.
int	Return the integer version for a string.
json	Return the JavaScript Object Notation (JSON) type value or object for a string or XML.
string	Return the string version for an input value.
uriComponent	Return the URI-encoded version for an input value by replacing URL-unsafe characters with escape characters.
uriComponentToBinary	Return the binary version for a URI-encoded string.
uriComponentToString	Return the string version for a URI-encoded string.
xml	Return the XML version for a string.
xpath	Check XML for nodes or values that match an XPath (XML Path Language) expression, and return the matching nodes or values.

Math functions

These functions can be used for either types of numbers: **integers** and **floats**.

MATH FUNCTION	TASK
add	Return the result from adding two numbers.
div	Return the result from dividing two numbers.
max	Return the highest value from a set of numbers or an array.
min	Return the lowest value from a set of numbers or an array.
mod	Return the remainder from dividing two numbers.
mul	Return the product from multiplying two numbers.
rand	Return a random integer from a specified range.
range	Return an integer array that starts from a specified integer.

MATH FUNCTION	TASK
sub	Return the result from subtracting the second number from the first number.

Function reference

This section lists all the available functions in alphabetical order.

add

Return the result from adding two numbers.

```
add(<summand_1>, <summand_2>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*summand_1*>, <*summand_2*>	Yes	Integer, Float, or mixed	The numbers to add

RETURN VALUE	TYPE	DESCRIPTION
<*result-sum*>	Integer or Float	The result from adding the specified numbers

Example

This example adds the specified numbers:

```
add(1, 1.5)
```

And returns this result: `2.5`

addDays

Add a number of days to a timestamp.

```
addDays('<timestamp>', <days>, '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp
<*days*>	Yes	Integer	The positive or negative number of days to add

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*updated-timestamp*>	String	The timestamp plus the specified number of days

Example 1

This example adds 10 days to the specified timestamp:

```
addDays('2018-03-15T13:00:00Z', 10)
```

And returns this result: `"2018-03-25T00:00:0000000Z"`

Example 2

This example subtracts five days from the specified timestamp:

```
addDays('2018-03-15T00:00:00Z', -5)
```

And returns this result: `"2018-03-10T00:00:0000000Z"`

addHours

Add a number of hours to a timestamp.

```
addHours('<timestamp>', <hours>, '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp
<*hours*>	Yes	Integer	The positive or negative number of hours to add

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*updated-timestamp*>	String	The timestamp plus the specified number of hours

Example 1

This example adds 10 hours to the specified timestamp:

```
addHours('2018-03-15T00:00:00Z', 10)
```

And returns this result: `"2018-03-15T10:00:0000000Z"`

Example 2

This example subtracts five hours from the specified timestamp:

```
addHours('2018-03-15T15:00:00Z', -5)
```

And returns this result: `"2018-03-15T10:00:0000000Z"`

addMinutes

Add a number of minutes to a timestamp.

```
addMinutes('<timestamp>', <minutes>, '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp
<*minutes*>	Yes	Integer	The positive or negative number of minutes to add

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*updated-timestamp*>	String	The timestamp plus the specified number of minutes

Example 1

This example adds 10 minutes to the specified timestamp:

```
addMinutes('2018-03-15T00:10:00Z', 10)
```

And returns this result: `"2018-03-15T00:20:00.0000000Z"`

Example 2

This example subtracts five minutes from the specified timestamp:

```
addMinutes('2018-03-15T00:20:00Z', -5)
```

And returns this result: `"2018-03-15T00:15:00.0000000Z"`

addSeconds

Add a number of seconds to a timestamp.

```
addSeconds('<timestamp>', <seconds>, '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp
<*seconds*>	Yes	Integer	The positive or negative number of seconds to add

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*updated-timestamp*>	String	The timestamp plus the specified number of seconds

Example 1

This example adds 10 seconds to the specified timestamp:

```
addSeconds('2018-03-15T00:00:00Z', 10)
```

And returns this result: `"2018-03-15T00:00:10.0000000Z"`

Example 2

This example subtracts five seconds to the specified timestamp:

```
addSeconds('2018-03-15T00:00:30Z', -5)
```

And returns this result: `"2018-03-15T00:00:25.0000000Z"`

addToTime

Add a number of time units to a timestamp. See also getFutureTime().

```
addToTime('<timestamp>', <interval>, '<timeUnit>', '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp
<*interval*>	Yes	Integer	The number of specified time units to add
<*timeUnit*>	Yes	String	The unit of time to use with *interval*: "Second", "Minute", "Hour", "Day", "Week", "Month", "Year"

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*updated-timestamp*>	String	The timestamp plus the specified number of time units

Example 1

This example adds one day to the specified timestamp:

```
addToTime('2018-01-01T00:00:00Z', 1, 'Day')
```

And returns this result: `"2018-01-02T00:00:00.0000000Z"`

Example 2

This example adds one day to the specified timestamp:

```
addToTime('2018-01-01T00:00:00Z', 1, 'Day', 'D')
```

And returns the result using the optional "D" format: `"Tuesday, January 2, 2018"`

and

Check whether both expressions are true. Return true when both expressions are true, or return false when at least one expression is false.

```
and(<expression1>, <expression2>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*expression1*>, <*expression2*>	Yes	Boolean	The expressions to check

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when both expressions are true. Return false when at least one expression is false.

RETURN VALUE	TYPE	DESCRIPTION

Example 1

These examples check whether the specified Boolean values are both true:

```
and(true, true)
and(false, true)
and(false, false)
```

And returns these results:

- First example: Both expressions are true, so returns `true`.
- Second example: One expression is false, so returns `false`.
- Third example: Both expressions are false, so returns `false`.

Example 2

These examples check whether the specified expressions are both true:

```
and(equals(1, 1), equals(2, 2))
and(equals(1, 1), equals(1, 2))
and(equals(1, 2), equals(1, 3))
```

And returns these results:

- First example: Both expressions are true, so returns `true`.
- Second example: One expression is false, so returns `false`.
- Third example: Both expressions are false, so returns `false`.

array

Return an array from a single specified input. For multiple inputs, see createArray().

```
array('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The string for creating an array

RETURN VALUE	TYPE	DESCRIPTION
[<*value*>]	Array	An array that contains the single specified input

Example

This example creates an array from the "hello" string:

```
array('hello')
```

And returns this result: `["hello"]`

base64

Return the base64-encoded version for a string.

```
base64('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The input string

RETURN VALUE	TYPE	DESCRIPTION
<*base64-string*>	String	The base64-encoded version for the input string

Example

This example converts the "hello" string to a base64-encoded string:

```
base64('hello')
```

And returns this result: `"aGVsbG8="`

base64ToBinary

Return the binary version for a base64-encoded string.

```
base64ToBinary('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The base64-encoded string to convert

RETURN VALUE	TYPE	DESCRIPTION
<*binary-for-base64-string*>	String	The binary version for the base64-encoded string

Example

This example converts the "aGVsbG8=" base64-encoded string to a binary string:

```
base64ToBinary('aGVsbG8=')
```

And returns this result:

```
"0110000101000111010101100111001101100010010001110011100000111101"
```

base64ToString

Return the string version for a base64-encoded string, effectively decoding the base64 string. Use this function rather than decodeBase64(). Although both functions work the same way, `base64ToString()` is preferred.

```
base64ToString('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The base64-encoded string to decode

RETURN VALUE	TYPE	DESCRIPTION
<*decoded-base64-string*>	String	The string version for a base64-encoded string

Example

This example converts the "aGVsbG8=" base64-encoded string to just a string:

```
base64ToString('aGVsbG8=')
```

And returns this result: `"hello"`

binary

Return the binary version for a string.

```
binary('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The string to convert

RETURN VALUE	TYPE	DESCRIPTION
<*binary-for-input-value*>	String	The binary version for the specified string

Example

This example converts the "hello" string to a binary string:

```
binary('hello')
```

And returns this result:

`"0110100001100101011011000110110001101111"`

bool

Return the Boolean version for a value.

```
bool(<value>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	Any	The value to convert

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	The Boolean version for the specified value

Example

These examples convert the specified values to Boolean values:

```
bool(1)
bool(0)
```

And returns these results:

- First example: `true`
- Second example: `false`

coalesce

Return the first non-null value from one or more parameters. Empty strings, empty arrays, and empty objects are not null.

```
coalesce(<object_1>, <object_2>, ...)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*object_1*>, <*object_2*>, ...	Yes	Any, can mix types	One or more items to check for null

RETURN VALUE	TYPE	DESCRIPTION
<*first-non-null-item*>	Any	The first item or value that is not null. If all parameters are null, this function returns null.

Example

These examples return the first non-null value from the specified values, or null when all the values are null:

```
coalesce(null, true, false)
coalesce(null, 'hello', 'world')
coalesce(null, null, null)
```

And returns these results:

- First example: `true`
- Second example: `"hello"`
- Third example: `null`

concat

Combine two or more strings, and return the combined string.

```
concat('<text1>', '<text2>', ...)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*text1*>, <*text2*>, ...	Yes	String	At least two strings to combine

RETURN VALUE	TYPE	DESCRIPTION
<*text1text2...*>	String	The string created from the combined input strings

Example

This example combines the strings "Hello" and "World":

```
concat('Hello', 'World')
```

And returns this result: `"HelloWorld"`

contains

Check whether a collection has a specific item. Return true when the item is found, or return false when not found. This function is case-sensitive.

```
contains('<collection>', '<value>')
contains([<collection>], '<value>')
```

Specifically, this function works on these collection types:

- A *string* to find a *substring*
- An *array* to find a *value*
- A *dictionary* to find a *key*

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*collection*>	Yes	String, Array, or Dictionary	The collection to check
<*value*>	Yes	String, Array, or Dictionary, respectively	The item to find

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when the item is found. Return false when not found.

Example 1

This example checks the string "hello world" for the substring "world" and returns true:

```
contains('hello world', 'world')
```

Example 2

This example checks the string "hello world" for the substring "universe" and returns false:

```
contains('hello world', 'universe')
```

convertFromUtc

Convert a timestamp from Universal Time Coordinated (UTC) to the target time zone.

```
convertFromUtc('<timestamp>', '<destinationTimeZone>', '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*destinationTimeZone*>	Yes	String	The name for the target time zone. For time zone names, see Microsoft Time Zone Values, but you might have to remove any punctuation from the time zone name.
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*converted-timestamp*>	String	The timestamp converted to the target time zone

Example 1

This example converts a timestamp to the specified time zone:

```
convertFromUtc('2018-01-01T08:00:00.0000000Z', 'Pacific Standard Time')
```

And returns this result: `"2018-01-01T00:00:00Z"`

Example 2

This example converts a timestamp to the specified time zone and format:

```
convertFromUtc('2018-01-01T08:00:00.0000000Z', 'Pacific Standard Time', 'D')
```

And returns this result: `"Monday, January 1, 2018"`

convertTimeZone

Convert a timestamp from the source time zone to the target time zone.

```
convertTimeZone('<timestamp>', '<sourceTimeZone>', '<destinationTimeZone>', '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*sourceTimeZone*>	Yes	String	The name for the source time zone. For time zone names, see Microsoft Time Zone Values, but you might have to remove any punctuation from the time zone name.
<*destinationTimeZone*>	Yes	String	The name for the target time zone. For time zone names, see Microsoft Time Zone Values, but you might have to remove any punctuation from the time zone name.
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*converted-timestamp*>	String	The timestamp converted to the target time zone

Example 1

This example converts the source time zone to the target time zone:

```
convertTimeZone('2018-01-01T08:00:00.0000000Z', 'UTC', 'Pacific Standard Time')
```

And returns this result: `"2018-01-01T00:00:00.0000000"`

Example 2

This example converts a time zone to the specified time zone and format:

```
convertTimeZone('2018-01-01T08:00:00.0000000Z', 'UTC', 'Pacific Standard Time', 'D')
```

And returns this result: `"Monday, January 1, 2018"`

convertToUtc

Convert a timestamp from the source time zone to Universal Time Coordinated (UTC).

```
convertToUtc('<timestamp>', '<sourceTimeZone>', '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp
<*sourceTimeZone*>	Yes	String	The name for the source time zone. For time zone names, see Microsoft Time Zone Values, but you might have to remove any punctuation from the time zone name.
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*converted-timestamp*>	String	The timestamp converted to UTC

Example 1

This example converts a timestamp to UTC:

```
convertToUtc('01/01/2018 00:00:00', 'Pacific Standard Time')
```

And returns this result: `"2018-01-01T08:00:00.0000000Z"`

Example 2

This example converts a timestamp to UTC:

```
convertToUtc('01/01/2018 00:00:00', 'Pacific Standard Time', 'D')
```

And returns this result: `"Monday, January 1, 2018"`

createArray

Return an array from multiple inputs. For single input arrays, see array().

```
createArray('<object1>', '<object2>', ...)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*object1*>, <*object2*>, ...	Yes	Any, but not mixed	At least two items to create the array

RETURN VALUE	TYPE	DESCRIPTION
[<*object1*>, <*object2*>, ...]	Array	The array created from all the input items

Example

This example creates an array from these inputs:

```
createArray('h', 'e', 'l', 'l', 'o')
```

And returns this result: `["h", "e", "l", "l", "o"]`

dataUri

Return a data uniform resource identifier (URI) for a string.

```
dataUri('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The string to convert

RETURN VALUE	TYPE	DESCRIPTION
<*data-uri*>	String	The data URI for the input string

Example

This example creates a data URI for the "hello" string:

```
dataUri('hello')
```

And returns this result: `"data:text/plain;charset=utf-8; base64,aGVsbG8="`

dataUriToBinary

Return the binary version for a data uniform resource identifier (URI). Use this function rather than decodeDataUri(). Although both functions work the same way, `dataUriBinary()` is preferred.

```
dataUriToBinary('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The data URI to convert

RETURN VALUE	TYPE	DESCRIPTION
<*binary-for-data-uri*>	String	The binary version for the data URI

Example

This example creates a binary version for this data URI:

```
dataUriToBinary('data:text/plain;charset=utf-8; base64,aGVsbG8=')
```

And returns this result:

```
"01100100011000010111010001100001001110100111010001100101011110000111010000101111011100000
11011000110000101101001011011100011101101100011011010000110000101110010011100110110010101111
01000011110101110101011101000110011000101101001110000011101101100010011000010111001101100101
10011011000110100001011000110000101000111010101100111001101100010010001110011100000111101"
```

dataUriToString

Return the string version for a data uniform resource identifier (URI).

```
dataUriToString('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The data URI to convert

RETURN VALUE	TYPE	DESCRIPTION
<*string-for-data-uri*>	String	The string version for the data URI

Example

This example creates a string for this data URI:

```
dataUriToString('data:text/plain;charset=utf-8; base64,aGVsbG8=')
```

And returns this result: `"hello"`

dayOfMonth

Return the day of the month from a timestamp.

```
dayOfMonth('<timestamp>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp

RETURN VALUE	TYPE	DESCRIPTION
<*day-of-month*>	Integer	The day of the month from the specified timestamp

Example

This example returns the number for the day of the month from this timestamp:

```
dayOfMonth('2018-03-15T13:27:36Z')
```

And returns this result: `15`

dayOfWeek

Return the day of the week from a timestamp.

```
dayOfWeek('<timestamp>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp

RETURN VALUE	TYPE	DESCRIPTION
<*day-of-week*>	Integer	The day of the week from the specified timestamp where Sunday is 0, Monday is 1, and so on

Example

This example returns the number for the day of the week from this timestamp:

```
dayOfWeek('2018-03-15T13:27:36Z')
```

And returns this result: `3`

dayOfYear

Return the day of the year from a timestamp.

```
dayOfYear('<timestamp>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp

RETURN VALUE	TYPE	DESCRIPTION
<*day-of-year*>	Integer	The day of the year from the specified timestamp

Example

This example returns the number of the day of the year from this timestamp:

```
dayOfYear('2018-03-15T13:27:36Z')
```

And returns this result: `74`

decodeBase64

Return the string version for a base64-encoded string, effectively decoding the base64 string. Consider using base64ToString() rather than `decodeBase64()`. Although both functions work the same way, `base64ToString()` is preferred.

```
decodeBase64('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The base64-encoded string to decode

RETURN VALUE	TYPE	DESCRIPTION
<*decoded-base64-string*>	String	The string version for a base64-encoded string

Example

This example creates a string for a base64-encoded string:

```
decodeBase64('aGVsbG8=')
```

And returns this result: `"hello"`

decodeDataUri

Return the binary version for a data uniform resource identifier (URI). Consider using dataUriToBinary(), rather than `decodeDataUri()`. Although both functions work the same way, `dataUriToBinary()` is preferred.

```
decodeDataUri('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<value>	Yes	String	The data URI string to decode

RETURN VALUE	TYPE	DESCRIPTION
<binary-for-data-uri>	String	The binary version for a data URI string

Example

This example returns the binary version for this data URI:

```
decodeDataUri('data:text/plain;charset=utf-8; base64,aGVsbG8=')
```

And returns this result:

```
"01100100011000010111010001100001001110100111010001100101011110000111010000101111011100000
11011000110000101101001011011100011101101100011011010000110000101110010011100110110010101111
01000011110101110101011101000110011000101101001110000011101101100010011000010111001101100101
10011011000110100001011000110000101000111010101100111001101100010010001110011100000111101"
```

decodeUriComponent

Return a string that replaces escape characters with decoded versions.

```
decodeUriComponent('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<value>	Yes	String	The string with the escape characters to decode

RETURN VALUE	TYPE	DESCRIPTION
<decoded-uri>	String	The updated string with the decoded escape characters

Example

This example replaces the escape characters in this string with decoded versions:

```
decodeUriComponent('http%3A%2F%2Fcontoso.com')
```

And returns this result: `"https://contoso.com"`

div

Return the integer result from dividing two numbers. To get the remainder result, see mod().

```
div(<dividend>, <divisor>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*dividend*>	Yes	Integer or Float	The number to divide by the *divisor*
<*divisor*>	Yes	Integer or Float	The number that divides the *dividend*, but cannot be 0

RETURN VALUE	TYPE	DESCRIPTION
<*quotient-result*>	Integer	The integer result from dividing the first number by the second number

Example

Both examples divide the first number by the second number:

```
div(10, 5)
div(11, 5)
```

And return this result: `2`

encodeUriComponent

Return a uniform resource identifier (URI) encoded version for a string by replacing URL-unsafe characters with escape characters. Consider using uriComponent(), rather than `encodeUriComponent()`. Although both functions work the same way, `uriComponent()` is preferred.

```
encodeUriComponent('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The string to convert to URI-encoded format

RETURN VALUE	TYPE	DESCRIPTION
<*encoded-uri*>	String	The URI-encoded string with escape characters

Example

This example creates a URI-encoded version for this string:

```
encodeUriComponent('https://contoso.com')
```

And returns this result: `"http%3A%2F%2Fcontoso.com"`

empty

Check whether a collection is empty. Return true when the collection is empty, or return false when not empty.

```
empty('<collection>')
empty([<collection>])
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*collection*>	Yes	String, Array, or Object	The collection to check

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when the collection is empty. Return false when not empty.

Example

These examples check whether the specified collections are empty:

```
empty('')
empty('abc')
```

And returns these results:

- First example: Passes an empty string, so the function returns `true`.
- Second example: Passes the string "abc", so the function returns `false`.

endsWith

Check whether a string ends with a specific substring. Return true when the substring is found, or return false when not found. This function is not case-sensitive.

```
endsWith('<text>', '<searchText>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*text*>	Yes	String	The string to check
<*searchText*>	Yes	String	The ending substring to find

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when the ending substring is found. Return false when not found.

Example 1

This example checks whether the "hello world" string ends with the "world" string:

```
endsWith('hello world', 'world')
```

And returns this result: `true`

Example 2

This example checks whether the "hello world" string ends with the "universe" string:

```
endsWith('hello world', 'universe')
```

And returns this result: `false`

equals

Check whether both values, expressions, or objects are equivalent. Return true when both are equivalent, or return false when they're not equivalent.

```
equals('<object1>', '<object2>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*object1*>, <*object2*>	Yes	Various	The values, expressions, or objects to compare

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when both are equivalent. Return false when not equivalent.

Example

These examples check whether the specified inputs are equivalent.

```
equals(true, 1)
equals('abc', 'abcd')
```

And returns these results:

- First example: Both values are equivalent, so the function returns `true`.
- Second example: Both values aren't equivalent, so the function returns `false`.

first

Return the first item from a string or array.

```
first('<collection>')
first([<collection>])
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*collection*>	Yes	String or Array	The collection where to find the first item

RETURN VALUE	TYPE	DESCRIPTION
<*first-collection-item*>	Any	The first item in the collection

Example

These examples find the first item in these collections:

```
first('hello')
first(createArray(0, 1, 2))
```

And return these results:

- First example: `"h"`
- Second example: `0`

float

Convert a string version for a floating-point number to an actual floating point number.

```
float('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The string that has a valid floating-point number to convert

RETURN VALUE	TYPE	DESCRIPTION
<*float-value*>	Float	The floating-point number for the specified string

Example

This example creates a string version for this floating-point number:

```
float('10.333')
```

And returns this result: `10.333`

formatDateTime

Return a timestamp in the specified format.

```
formatDateTime('<timestamp>', '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*reformatted-timestamp*>	String	The updated timestamp in the specified format

Example

This example converts a timestamp to the specified format:

```
formatDateTime('03/15/2018 12:00:00', 'yyyy-MM-ddTHH:mm:ss')
```

And returns this result: `"2018-03-15T12:00:00"`

getFutureTime

Return the current timestamp plus the specified time units.

```
getFutureTime(<interval>, <timeUnit>, <format>?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<interval>	Yes	Integer	The number of specified time units to add
<timeUnit>	Yes	String	The unit of time to use with *interval*: "Second", "Minute", "Hour", "Day", "Week", "Month", "Year"
<format>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<updated-timestamp>	String	The current timestamp plus the specified number of time units

Example 1

Suppose the current timestamp is "2018-03-01T00:00:00.0000000Z". This example adds five days to that timestamp:

```
getFutureTime(5, 'Day')
```

And returns this result: `"2018-03-06T00:00:00.0000000Z"`

Example 2

Suppose the current timestamp is "2018-03-01T00:00:00.0000000Z". This example adds five days and converts the result to "D" format:

```
getFutureTime(5, 'Day', 'D')
```

And returns this result: `"Tuesday, March 6, 2018"`

getPastTime

Return the current timestamp minus the specified time units.

```
getPastTime(<interval>, <timeUnit>, <format>?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*interval*>	Yes	Integer	The number of specified time units to subtract
<*timeUnit*>	Yes	String	The unit of time to use with *interval*: "Second", "Minute", "Hour", "Day", "Week", "Month", "Year"
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*updated-timestamp*>	String	The current timestamp minus the specified number of time units

Example 1

Suppose the current timestamp is "2018-02-01T00:00:00.0000000Z". This example subtracts five days from that timestamp:

```
getPastTime(5, 'Day')
```

And returns this result: `"2018-01-27T00:00:00.0000000Z"`

Example 2

Suppose the current timestamp is "2018-02-01T00:00:00.0000000Z". This example subtracts five days and converts the result to "D" format:

```
getPastTime(5, 'Day', 'D')
```

And returns this result: `"Saturday, January 27, 2018"`

greater

Check whether the first value is greater than the second value. Return true when the first value is more, or return false when less.

```
greater(<value>, <compareTo>)
greater('<value>', '<compareTo>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<value>	Yes	Integer, Float, or String	The first value to check whether greater than the second value
<compareTo>	Yes	Integer, Float, or String, respectively	The comparison value

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when the first value is greater than the second value. Return false when the first value is equal to or less than the second value.

Example

These examples check whether the first value is greater than the second value:

```
greater(10, 5)
greater('apple', 'banana')
```

And return these results:

- First example: `true`
- Second example: `false`

greaterOrEquals

Check whether the first value is greater than or equal to the second value. Return true when the first value is greater or equal, or return false when the first value is less.

```
greaterOrEquals(<value>, <compareTo>)
greaterOrEquals('<value>', '<compareTo>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<value>	Yes	Integer, Float, or String	The first value to check whether greater than or equal to the second value
<compareTo>	Yes	Integer, Float, or String, respectively	The comparison value

RETURN VALUE	TYPE	DESCRIPTION

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when the first value is greater than or equal to the second value. Return false when the first value is less than the second value.

Example

These examples check whether the first value is greater or equal than the second value:

```
greaterOrEquals(5, 5)
greaterOrEquals('apple', 'banana')
```

And return these results:

- First example: `true`
- Second example: `false`

guid

Generate a globally unique identifier (GUID) as a string, for example, "c2ecc88d-88c8-4096-912c-d6f2e2b138ce":

```
guid()
```

Also, you can specify a different format for the GUID other than the default format, "D", which is 32 digits separated by hyphens.

```
guid('<format>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*format*>	No	String	A single format specifier for the returned GUID. By default, the format is "D", but you can use "N", "D", "B", "P", or "X".

RETURN VALUE	TYPE	DESCRIPTION
<*GUID-value*>	String	A randomly generated GUID

Example

This example generates the same GUID, but as 32 digits, separated by hyphens, and enclosed in parentheses:

```
guid('P')
```

And returns this result: `"(c2ecc88d-88c8-4096-912c-d6f2e2b138ce)"`

if

Check whether an expression is true or false. Based on the result, return a specified value.

```
if(<expression>, <valueIfTrue>, <valueIfFalse>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*expression*>	Yes	Boolean	The expression to check
<*valueIfTrue*>	Yes	Any	The value to return when the expression is true
<*valueIfFalse*>	Yes	Any	The value to return when the expression is false

RETURN VALUE	TYPE	DESCRIPTION
<*specified-return-value*>	Any	The specified value that returns based on whether the expression is true or false

Example

This example returns `"yes"` because the specified expression returns true. Otherwise, the example returns `"no"`:

```
if(equals(1, 1), 'yes', 'no')
```

indexOf

Return the starting position or index value for a substring. This function is not case-sensitive, and indexes start with the number 0.

```
indexOf('<text>', '<searchText>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*text*>	Yes	String	The string that has the substring to find
<*searchText*>	Yes	String	The substring to find

RETURN VALUE	TYPE	DESCRIPTION

RETURN VALUE	TYPE	DESCRIPTION
< *index-value* >	Integer	The starting position or index value for the specified substring. If the string is not found, return the number -1.

Example

This example finds the starting index value for the "world" substring in the "hello world" string:

```
indexOf('hello world', 'world')
```

And returns this result: `6`

int

Return the integer version for a string.

```
int('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
< *value* >	Yes	String	The string to convert

RETURN VALUE	TYPE	DESCRIPTION
< *integer-result* >	Integer	The integer version for the specified string

Example

This example creates an integer version for the string "10":

```
int('10')
```

And returns this result: `10`

json

Return the JavaScript Object Notation (JSON) type value or object for a string or XML.

```
json('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
< *value* >	Yes	String or XML	The string or XML to convert

PARAMETER	REQUIRED	TYPE	DESCRIPTION

RETURN VALUE	TYPE	DESCRIPTION
<JSON-result>	JSON native type or object	The JSON native type value or object for the specified string or XML. If the string is null, the function returns an empty object.

Example 1

This example converts this string to the JSON value:

```
json('[1, 2, 3]')
```

And returns this result: `[1, 2, 3]`

Example 2

This example converts this string to JSON:

```
json('{"fullName": "Sophia Owen"}')
```

And returns this result:

```
{
  "fullName": "Sophia Owen"
}
```

Example 3

This example converts this XML to JSON:

```
json(xml('<?xml version="1.0"?> <root> <person id='1'> <name>Sophia Owen</name>
<occupation>Engineer</occupation> </person> </root>'))
```

And returns this result:

```
{
   "?xml": { "@version": "1.0" },
   "root": {
      "person": [ {
         "@id": "1",
         "name": "Sophia Owen",
         "occupation": "Engineer"
      } ]
   }
}
```

intersection

Return a collection that has *only* the common items across the specified collections. To appear in the result, an item must appear in all the collections passed to this function. If one or more items have the same name, the last

item with that name appears in the result.

```
intersection([<collection1>], [<collection2>], ...)
intersection('<collection1>', '<collection2>', ...)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*collection1*>, <*collection2*>, ...	Yes	Array or Object, but not both	The collections from where you want *only* the common items

RETURN VALUE	TYPE	DESCRIPTION
<*common-items*>	Array or Object, respectively	A collection that has only the common items across the specified collections

Example

This example finds the common items across these arrays:

```
intersection(createArray(1, 2, 3), createArray(101, 2, 1, 10), createArray(6, 8, 1, 2))
```

And returns an array with *only* these items: `[1, 2]`

join

Return a string that has all the items from an array and has each character separated by a *delimiter*.

```
join([<collection>], '<delimiter>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*collection*>	Yes	Array	The array that has the items to join
<*delimiter*>	Yes	String	The separator that appears between each character in the resulting string

RETURN VALUE	TYPE	DESCRIPTION
<*char1*> <*delimiter*> <*char2*> <*delimiter*> ...	String	The resulting string created from all the items in the specified array

Example

This example creates a string from all the items in this array with the specified character as the delimiter:

```
join(createArray('a', 'b', 'c'), '.')
```

And returns this result: `"a.b.c"`

last

Return the last item from a collection.

```
last('<collection>')
last([<collection>])
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*collection*>	Yes	String or Array	The collection where to find the last item

RETURN VALUE	TYPE	DESCRIPTION
<*last-collection-item*>	String or Array, respectively	The last item in the collection

Example

These examples find the last item in these collections:

```
last('abcd')
last(createArray(0, 1, 2, 3))
```

And returns these results:

- First example: `"d"`
- Second example: `3`

lastIndexOf

Return the starting position or index value for the last occurrence of a substring. This function is not case-sensitive, and indexes start with the number 0.

```
lastIndexOf('<text>', '<searchText>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*text*>	Yes	String	The string that has the substring to find
<*searchText*>	Yes	String	The substring to find

RETURN VALUE	TYPE	DESCRIPTION
<*ending-index-value*>	Integer	The starting position or index value for the last occurrence of the specified substring. If the string is not found, return the number -1.

Example

This example finds the starting index value for the last occurrence of the "world" substring in the "hello world" string:

```
lastIndexOf('hello world', 'world')
```

And returns this result: `6`

length

Return the number of items in a collection.

```
length('<collection>')
length([<collection>])
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*collection*>	Yes	String or Array	The collection with the items to count

RETURN VALUE	TYPE	DESCRIPTION
<*length-or-count*>	Integer	The number of items in the collection

Example

These examples count the number of items in these collections:

```
length('abcd')
length(createArray(0, 1, 2, 3))
```

And return this result: `4`

less

Check whether the first value is less than the second value. Return true when the first value is less, or return false when the first value is more.

```
less(<value>, <compareTo>)
less('<value>', '<compareTo>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<value>	Yes	Integer, Float, or String	The first value to check whether less than the second value
<compareTo>	Yes	Integer, Float, or String, respectively	The comparison item

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when the first value is less than the second value. Return false when the first value is equal to or greater than the second value.

Example

These examples check whether the first value is less than the second value.

```
less(5, 10)
less('banana', 'apple')
```

And return these results:

- First example: `true`
- Second example: `false`

lessOrEquals

Check whether the first value is less than or equal to the second value. Return true when the first value is less than or equal, or return false when the first value is more.

```
lessOrEquals(<value>, <compareTo>)
lessOrEquals('<value>', '<compareTo>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<value>	Yes	Integer, Float, or String	The first value to check whether less than or equal to the second value
<compareTo>	Yes	Integer, Float, or String, respectively	The comparison item

RETURN VALUE	TYPE	DESCRIPTION

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when the first value is less than or equal to the second value. Return false when the first value is greater than the second value.

Example

These examples check whether the first value is less or equal than the second value.

```
lessOrEquals(10, 10)
lessOrEquals('apply', 'apple')
```

And return these results:

- First example: `true`
- Second example: `false`

max

Return the highest value from a list or array with numbers that is inclusive at both ends.

```
max(<number1>, <number2>, ...)
max([<number1>, <number2>, ...])
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*number1*>, <*number2*>, ...	Yes	Integer, Float, or both	The set of numbers from which you want the highest value
[<*number1*>, <*number2*>, ...]	Yes	Array - Integer, Float, or both	The array of numbers from which you want the highest value

RETURN VALUE	TYPE	DESCRIPTION
<*max-value*>	Integer or Float	The highest value in the specified array or set of numbers

Example

These examples get the highest value from the set of numbers and the array:

```
max(1, 2, 3)
max(createArray(1, 2, 3))
```

And return this result: `3`

min

Return the lowest value from a set of numbers or an array.

```
min(<number1>, <number2>, ...)
min([<number1>, <number2>, ...])
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*number1*>, <*number2*>, ...	Yes	Integer, Float, or both	The set of numbers from which you want the lowest value
[<*number1*>, <*number2*>, ...]	Yes	Array - Integer, Float, or both	The array of numbers from which you want the lowest value

RETURN VALUE	TYPE	DESCRIPTION
<*min-value*>	Integer or Float	The lowest value in the specified set of numbers or specified array

Example

These examples get the lowest value in the set of numbers and the array:

```
min(1, 2, 3)
min(createArray(1, 2, 3))
```

And return this result: `1`

mod

Return the remainder from dividing two numbers. To get the integer result, see div().

```
mod(<dividend>, <divisor>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*dividend*>	Yes	Integer or Float	The number to divide by the *divisor*
<*divisor*>	Yes	Integer or Float	The number that divides the *dividend*, but cannot be 0.

RETURN VALUE	TYPE	DESCRIPTION
<*modulo-result*>	Integer or Float	The remainder from dividing the first number by the second number

Example

This example divides the first number by the second number:

```
mod(3, 2)
```

And return this result: `1`

mul

Return the product from multiplying two numbers.

```
mul(<multiplicand1>, <multiplicand2>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*multiplicand1*>	Yes	Integer or Float	The number to multiply by *multiplicand2*
<*multiplicand2*>	Yes	Integer or Float	The number that multiples *multiplicand1*

RETURN VALUE	TYPE	DESCRIPTION
<*product-result*>	Integer or Float	The product from multiplying the first number by the second number

Example

These examples multiple the first number by the second number:

```
mul(1, 2)
mul(1.5, 2)
```

And return these results:

- First example: `2`
- Second example `3`

not

Check whether an expression is false. Return true when the expression is false, or return false when true.

```
not(<expression>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*expression*>	Yes	Boolean	The expression to check

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when the expression is false. Return false when the expression is true.

Example 1

These examples check whether the specified expressions are false:

```
not(false)
not(true)
```

And return these results:

- First example: The expression is false, so the function returns `true`.
- Second example: The expression is true, so the function returns `false`.

Example 2

These examples check whether the specified expressions are false:

```
not(equals(1, 2))
not(equals(1, 1))
```

And return these results:

- First example: The expression is false, so the function returns `true`.
- Second example: The expression is true, so the function returns `false`.

or

Check whether at least one expression is true. Return true when at least one expression is true, or return false when both are false.

```
or(<expression1>, <expression2>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*expression1*>, <*expression2*>	Yes	Boolean	The expressions to check

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when at least one expression is true. Return false when both expressions are false.

Example 1

These examples check whether at least one expression is true:

```
or(true, false)
or(false, false)
```

And return these results:

- First example: At least one expression is true, so the function returns `true`.
- Second example: Both expressions are false, so the function returns `false`.

Example 2

These examples check whether at least one expression is true:

```
or(equals(1, 1), equals(1, 2))
or(equals(1, 2), equals(1, 3))
```

And return these results:

- First example: At least one expression is true, so the function returns `true`.
- Second example: Both expressions are false, so the function returns `false`.

rand

Return a random integer from a specified range, which is inclusive only at the starting end.

```
rand(<minValue>, <maxValue>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*minValue*>	Yes	Integer	The lowest integer in the range
<*maxValue*>	Yes	Integer	The integer that follows the highest integer in the range that the function can return

RETURN VALUE	TYPE	DESCRIPTION
<*random-result*>	Integer	The random integer returned from the specified range

Example

This example gets a random integer from the specified range, excluding the maximum value:

```
rand(1, 5)
```

And returns one of these numbers as the result: `1`, `2`, `3`, or `4`

range

Return an integer array that starts from a specified integer.

```
range(<startIndex>, <count>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*startIndex*>	Yes	Integer	An integer value that starts the array as the first item
<*count*>	Yes	Integer	The number of integers in the array

RETURN VALUE	TYPE	DESCRIPTION
[<*range-result*>]	Array	The array with integers starting from the specified index

Example

This example creates an integer array that starts from the specified index and has the specified number of integers:

```
range(1, 4)
```

And returns this result: `[1, 2, 3, 4]`

replace

Replace a substring with the specified string, and return the result string. This function is case-sensitive.

```
replace('<text>', '<oldText>', '<newText>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*text*>	Yes	String	The string that has the substring to replace
<*oldText*>	Yes	String	The substring to replace
<*newText*>	Yes	String	The replacement string

RETURN VALUE	TYPE	DESCRIPTION
<*updated-text*>	String	The updated string after replacing the substring If the substring is not found, return the original string.

Example

This example finds the "old" substring in "the old string" and replaces "old" with "new":

```
replace('the old string', 'old', 'new')
```

And returns this result: `"the new string"`

skip

Remove items from the front of a collection, and return *all the other* items.

```
skip([<collection>], <count>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*collection*>	Yes	Array	The collection whose items you want to remove
<*count*>	Yes	Integer	A positive integer for the number of items to remove at the front

RETURN VALUE	TYPE	DESCRIPTION
[<*updated-collection*>]	Array	The updated collection after removing the specified items

Example

This example removes one item, the number 0, from the front of the specified array:

```
skip(createArray(0, 1, 2, 3), 1)
```

And returns this array with the remaining items: `[1,2,3]`

split

Return an array that contains substrings, separated by commas, based on the specified delimiter character in the original string.

```
split('<text>', '<delimiter>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*text*>	Yes	String	The string to separate into substrings based on the specified delimiter in the original string

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*delimiter*>	Yes	String	The character in the original string to use as the delimiter

RETURN VALUE	TYPE	DESCRIPTION
[<*substring1*>,<*substring2*>,...]	Array	An array that contains substrings from the original string, separated by commas

Example

This example creates an array with substrings from the specified string based on the specified character as the delimiter:

```
split('a_b_c', '_')
```

And returns this array as the result: `["a","b","c"]`

startOfDay

Return the start of the day for a timestamp.

```
startOfDay('<timestamp>', '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*updated-timestamp*>	String	The specified timestamp but starting at the zero-hour mark for the day

Example

This example finds the start of the day for this timestamp:

```
startOfDay('2018-03-15T13:30:30Z')
```

And returns this result: `"2018-03-15T00:00:00.0000000Z"`

startOfHour

Return the start of the hour for a timestamp.

```
startOfHour('<timestamp>', '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*updated-timestamp*>	String	The specified timestamp but starting at the zero-minute mark for the hour

Example

This example finds the start of the hour for this timestamp:

```
startOfHour('2018-03-15T13:30:30Z')
```

And returns this result: `"2018-03-15T13:00:00.0000000Z"`

startOfMonth

Return the start of the month for a timestamp.

```
startOfMonth('<timestamp>', '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*updated-timestamp*>	String	The specified timestamp but starting on the first day of the month at the zero-hour mark

Example

This example returns the start of the month for this timestamp:

```
startOfMonth('2018-03-15T13:30:30Z')
```

And returns this result: `"2018-03-01T00:00:00.0000000Z"`

startsWith

Check whether a string starts with a specific substring. Return true when the substring is found, or return false when not found. This function is not case-sensitive.

```
startsWith('<text>', '<searchText>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*text*>	Yes	String	The string to check
<*searchText*>	Yes	String	The starting string to find

RETURN VALUE	TYPE	DESCRIPTION
true or false	Boolean	Return true when the starting substring is found. Return false when not found.

Example 1

This example checks whether the "hello world" string starts with the "hello" substring:

```
startsWith('hello world', 'hello')
```

And returns this result: `true`

Example 2

This example checks whether the "hello world" string starts with the "greetings" substring:

```
startsWith('hello world', 'greetings')
```

And returns this result: `false`

string

Return the string version for a value.

```
string(<value>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	Any	The value to convert

RETURN VALUE	TYPE	DESCRIPTION
<*string-value*>	String	The string version for the specified value

Example 1

This example creates the string version for this number:

```
string(10)
```

And returns this result: `"10"`

Example 2

This example creates a string for the specified JSON object and uses the backslash character (\) as an escape character for the double-quotation mark (").

```
string( { "name": "Sophie Owen" } )
```

And returns this result: `"{ \\"name\\": \\"Sophie Owen\\" }"`

sub

Return the result from subtracting the second number from the first number.

```
sub(<minuend>, <subtrahend>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*minuend*>	Yes	Integer or Float	The number from which to subtract the *subtrahend*
<*subtrahend*>	Yes	Integer or Float	The number to subtract from the *minuend*

RETURN VALUE	TYPE	DESCRIPTION
<*result*>	Integer or Float	The result from subtracting the second number from the first number

Example

This example subtracts the second number from the first number:

```
sub(10.3, .3)
```

And returns this result: `10`

substring

Return characters from a string, starting from the specified position, or index. Index values start with the number 0.

```
substring('<text>', <startIndex>, <length>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*text*>	Yes	String	The string whose characters you want
<*startIndex*>	Yes	Integer	A positive number equal to or greater than 0 that you want to use as the starting position or index value
<*length*>	Yes	Integer	A positive number of characters that you want in the substring

RETURN VALUE	TYPE	DESCRIPTION
<*substring-result*>	String	A substring with the specified number of characters, starting at the specified index position in the source string

Example

This example creates a five-character substring from the specified string, starting from the index value 6:

```
substring('hello world', 6, 5)
```

And returns this result: `"world"`

subtractFromTime

Subtract a number of time units from a timestamp. See also getPastTime.

```
subtractFromTime('<timestamp>', <interval>, '<timeUnit>', '<format>'?)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string that contains the timestamp
<*interval*>	Yes	Integer	The number of specified time units to subtract
<*timeUnit*>	Yes	String	The unit of time to use with *interval*: "Second", "Minute", "Hour", "Day", "Week", "Month", "Year"
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*updated-timestamp*>	String	The timestamp minus the specified number of time units

Example 1

This example subtracts one day from this timestamp:

```
subtractFromTime('2018-01-02T00:00:00Z', 1, 'Day')
```

And returns this result: `"2018-01-01T00:00:00.0000000Z"`

Example 2

This example subtracts one day from this timestamp:

```
subtractFromTime('2018-01-02T00:00:00Z', 1, 'Day', 'D')
```

And returns this result using the optional "D" format: `"Monday, January, 1, 2018"`

take

Return items from the front of a collection.

```
take('<collection>', <count>)
take([<collection>], <count>)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*collection*>	Yes	String or Array	The collection whose items you want
<*count*>	Yes	Integer	A positive integer for the number of items that you want from the front

RETURN VALUE	TYPE	DESCRIPTION
<*subset*> or [<*subset*>]	String or Array, respectively	A string or array that has the specified number of items taken from the front of the original collection

Example

These examples get the specified number of items from the front of these collections:

```
take('abcde', 3)
take(createArray(0, 1, 2, 3, 4), 3)
```

And return these results:

- First example: `"abc"`
- Second example: `[0, 1, 2]`

ticks

Return the `ticks` property value for a specified timestamp. A *tick* is a 100-nanosecond interval.

```
ticks('<timestamp>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*timestamp*>	Yes	String	The string for a timestamp

RETURN VALUE	TYPE	DESCRIPTION
<*ticks-number*>	Integer	The number of ticks since the specified timestamp

toLower

Return a string in lowercase format. If a character in the string doesn't have a lowercase version, that character stays unchanged in the returned string.

```
toLower('<text>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*text*>	Yes	String	The string to return in lowercase format

RETURN VALUE	TYPE	DESCRIPTION
<*lowercase-text*>	String	The original string in lowercase format

Example

This example converts this string to lowercase:

```
toLower('Hello World')
```

And returns this result: `"hello world"`

toUpper

Return a string in uppercase format. If a character in the string doesn't have an uppercase version, that character stays unchanged in the returned string.

```
toUpper('<text>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*text*>	Yes	String	The string to return in uppercase format

RETURN VALUE	TYPE	DESCRIPTION
<*uppercase-text*>	String	The original string in uppercase format

Example

This example converts this string to uppercase:

```
toUpper('Hello World')
```

And returns this result: `"HELLO WORLD"`

trim

Remove leading and trailing whitespace from a string, and return the updated string.

```
trim('<text>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*text*>	Yes	String	The string that has the leading and trailing whitespace to remove

RETURN VALUE	TYPE	DESCRIPTION
<*updatedText*>	String	An updated version for the original string without leading or trailing whitespace

Example

This example removes the leading and trailing whitespace from the string " Hello World ":

```
trim(' Hello World  ')
```

And returns this result: `"Hello World"`

union

Return a collection that has *all* the items from the specified collections. To appear in the result, an item can appear in any collection passed to this function. If one or more items have the same name, the last item with that name appears in the result.

```
union('<collection1>', '<collection2>', ...)
union([<collection1>], [<collection2>], ...)
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*collection1*>, <*collection2*>, ...	Yes	Array or Object, but not both	The collections from where you want *all* the items

RETURN VALUE	TYPE	DESCRIPTION
<*updatedCollection*>	Array or Object, respectively	A collection with all the items from the specified collections - no duplicates

Example

This example gets *all* the items from these collections:

```
union(createArray(1, 2, 3), createArray(1, 2, 10, 101))
```

And returns this result: `[1, 2, 3, 10, 101]`

uriComponent

Return a uniform resource identifier (URI) encoded version for a string by replacing URL-unsafe characters with escape characters. Use this function rather than encodeUriComponent(). Although both functions work the same way, `uriComponent()` is preferred.

```
uriComponent('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The string to convert to URI-encoded format

RETURN VALUE	TYPE	DESCRIPTION
<*encoded-uri*>	String	The URI-encoded string with escape characters

Example

This example creates a URI-encoded version for this string:

```
uriComponent('https://contoso.com')
```

And returns this result: `"http%3A%2F%2Fcontoso.com"`

uriComponentToBinary

Return the binary version for a uniform resource identifier (URI) component.

```
uriComponentToBinary('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The URI-encoded string to convert

RETURN VALUE	TYPE	DESCRIPTION
<*binary-for-encoded-uri*>	String	The binary version for the URI-encoded string. The binary content is base64-encoded and represented by `$content`.

Example

This example creates the binary version for this URI-encoded string:

```
uriComponentToBinary('http%3A%2F%2Fcontoso.com')
```

And returns this result:

```
"0010001001101000011101000111010001110000001001010011001101000001001001010011001001000110001001010011001000100010010100110010010001100010010100110010001
1001100011011011110110111001110100110111101110011011011 1100101110011000110110111101101101001000100"
```

uriComponentToString

Return the string version for a uniform resource identifier (URI) encoded string, effectively decoding the URI-encoded string.

```
uriComponentToString('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The URI-encoded string to decode

RETURN VALUE	TYPE	DESCRIPTION
<*decoded-uri*>	String	The decoded version for the URI-encoded string

Example

This example creates the decoded string version for this URI-encoded string:

```
uriComponentToString('http%3A%2F%2Fcontoso.com')
```

And returns this result: `"https://contoso.com"`

utcNow

Return the current timestamp.

```
utcNow('<format>')
```

Optionally, you can specify a different format with the <*format*> parameter.

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*format*>	No	String	Either a single format specifier or a custom format pattern. The default format for the timestamp is "o" (yyyy-MM-ddTHH:mm:ss:fffffffK), which complies with ISO 8601 and preserves time zone information.

RETURN VALUE	TYPE	DESCRIPTION
<*current-timestamp*>	String	The current date and time

Example 1

Suppose today is April 15, 2018 at 1:00:00 PM. This example gets the current timestamp:

```
utcNow()
```

And returns this result: `"2018-04-15T13:00:00.0000000Z"`

Example 2

Suppose today is April 15, 2018 at 1:00:00 PM. This example gets the current timestamp using the optional "D" format:

```
utcNow('D')
```

And returns this result: `"Sunday, April 15, 2018"`

xml

Return the XML version for a string that contains a JSON object.

```
xml('<value>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*value*>	Yes	String	The string with the JSON object to convert The JSON object must have only one root property, which can't be an array. Use the backslash character (\) as an escape character for the double quotation mark (\").

RETURN VALUE	TYPE	DESCRIPTION
<*xml-version*>	Object	The encoded XML for the specified string or JSON object

Example 1

This example creates the XML version for this string, which contains a JSON object:

```
xml(json('{ \"name\": \"Sophia Owen\" }'))
```

And returns this result XML:

```
<name>Sophia Owen</name>
```

Example 2

Suppose you have this JSON object:

```
{
  "person": {
    "name": "Sophia Owen",
    "city": "Seattle"
  }
}
```

This example creates XML for a string that contains this JSON object:

```
xml(json('{\"person\": {\"name\": \"Sophia Owen\", \"city\": \"Seattle\"}}'))
```

And returns this result XML:

```
<person>
  <name>Sophia Owen</name>
  <city>Seattle</city>
<person>
```

xpath

Check XML for nodes or values that match an XPath (XML Path Language) expression, and return the matching

nodes or values. An XPath expression, or just "XPath", helps you navigate an XML document structure so that you can select nodes or compute values in the XML content.

```
xpath('<xml>', '<xpath>')
```

PARAMETER	REQUIRED	TYPE	DESCRIPTION
<*xml*>	Yes	Any	The XML string to search for nodes or values that match an XPath expression value
<*xpath*>	Yes	Any	The XPath expression used to find matching XML nodes or values

RETURN VALUE	TYPE	DESCRIPTION
<*xml-node*>	XML	An XML node when only a single node matches the specified XPath expression
<*value*>	Any	The value from an XML node when only a single value matches the specified XPath expression
[<*xml-node1*>, <*xml-node2*>, ...] -or- [<*value1*>, <*value2*>, ...]	Array	An array with XML nodes or values that match the specified XPath expression

Example 1

Following on Example 1, this example finds nodes that match the `<count></count>` node and adds those node values with the `sum()` function:

```
xpath(xml(parameters('items')), 'sum(/produce/item/count)')
```

And returns this result: `30`

Example 2

For this example, both expressions find nodes that match the `<location></location>` node, in the specified arguments, which include XML with a namespace. The expressions use the backslash character (\) as an escape character for the double quotation mark (").

- *Expression 1*

  ```
  xpath(xml(body('Http')), '/*[name()=\"file\"]/*[name()=\"location\"]')
  ```

- *Expression 2*

  ```
  xpath(xml(body('Http')), '/*[local-name()=\"file\" and namespace-uri()=\"http://contoso.com\"]/*
  [local-name()=\"location\"]')
  ```

Here are the arguments:

- This XML, which includes the XML document namespace, `xmlns="http://contoso.com"`:

```
<?xml version="1.0"?> <file xmlns="http://contoso.com"> <location>Paris</location> </file>
```

- Either XPath expression here:
  - `/*[name()=\"file\"]/*[name()=\"location\"]`
  - `/*[local-name()=\"file\" and namespace-uri()=\"http://contoso.com\"]/*[local-name()=\"location\"]`

Here is the result node that matches the `<location></location>` node:

```
<location xmlns="https://contoso.com">Paris</location>
```

Example 3

Following on Example 3, this example finds the value in the `<location></location>` node:

```
xpath(xml(body('Http')), 'string(/*[name()=\"file\"]/*[name()=\"location\"])')
```

And returns this result: `"Paris"`

Next steps

For a list of system variables you can use in expressions, see System variables.

System variables supported by Azure Data Factory and Azure Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article describes system variables supported by Azure Data Factory and Azure Synapse. You can use these variables in expressions when defining entities within either service.

Pipeline scope

These system variables can be referenced anywhere in the pipeline JSON.

VARIABLE NAME	DESCRIPTION
@pipeline().DataFactory	Name of the data or Synapse workspace the pipeline run is running in
@pipeline().Pipeline	Name of the pipeline
@pipeline().RunId	ID of the specific pipeline run
@pipeline().TriggerType	The type of trigger that invoked the pipeline (for example, `ScheduleTrigger`, `BlobEventsTrigger`). For a list of supported trigger types, see Pipeline execution and triggers. A trigger type of `Manual` indicates that the pipeline was triggered manually.
@pipeline().TriggerId	ID of the trigger that invoked the pipeline
@pipeline().TriggerName	Name of the trigger that invoked the pipeline
@pipeline().TriggerTime	Time of the trigger run that invoked the pipeline. This is the time at which the trigger **actually** fired to invoke the pipeline run, and it may differ slightly from the trigger's scheduled time.
@pipeline().GroupId	ID of the group to which pipeline run belongs.
@pipeline()?TriggeredByPipelineName	Name of the pipeline that triggers the pipeline run. Applicable when the pipeline run is triggered by an ExecutePipeline activity. Evaluate to *Null* when used in other circumstances. Note the question mark after @pipeline()
@pipeline()?TriggeredByPipelineRunId	Run ID of the pipeline that triggers the pipeline run. Applicable when the pipeline run is triggered by an ExecutePipeline activity. Evaluate to *Null* when used in other circumstances. Note the question mark after @pipeline()

> **NOTE**
> Trigger-related date/time system variables (in both pipeline and trigger scopes) return UTC dates in ISO 8601 format, for example, `2017-06-01T22:20:00.4061448Z`.

Schedule trigger scope

These system variables can be referenced anywhere in the trigger JSON for triggers of type ScheduleTrigger.

VARIABLE NAME	DESCRIPTION
@trigger().scheduledTime	Time at which the trigger was scheduled to invoke the pipeline run.
@trigger().startTime	Time at which the trigger **actually** fired to invoke the pipeline run. This may differ slightly from the trigger's scheduled time.

Tumbling window trigger scope

These system variables can be referenced anywhere in the trigger JSON for triggers of type TumblingWindowTrigger.

VARIABLE NAME	DESCRIPTION
@trigger().outputs.windowStartTime	Start of the window associated with the trigger run.
@trigger().outputs.windowEndTime	End of the window associated with the trigger run.
@trigger().scheduledTime	Time at which the trigger was scheduled to invoke the pipeline run.
@trigger().startTime	Time at which the trigger **actually** fired to invoke the pipeline run. This may differ slightly from the trigger's scheduled time.

Storage event trigger scope

These system variables can be referenced anywhere in the trigger JSON for triggers of type BlobEventsTrigger.

VARIABLE NAME	DESCRIPTION
@triggerBody().fileName	Name of the file whose creation or deletion caused the trigger to fire.
@triggerBody().folderPath	Path to the folder that contains the file specified by `@triggerBody().fileName`. The first segment of the folder path is the name of the Azure Blob Storage container.
@trigger().startTime	Time at which the trigger fired to invoke the pipeline run.

> **NOTE**
>
> If you are creating your pipeline and trigger in Azure Synapse Analytics, you must use `@trigger().outputs.body.fileName` and `@trigger().outputs.body.folderPath` as parameters. Those two properties capture blob information. Use those properties instead of using `@triggerBody().fileName` and `@triggerBody().folderPath`.

Custom event trigger scope

These system variables can be referenced anywhere in the trigger JSON for triggers of type CustomEventsTrigger.

> **NOTE**
>
> The service expects custom events to be formatted with Azure Event Grid event schema.

VARIABLE NAME	DESCRIPTION
@triggerBody().event.eventType	Type of events that triggered the Custom Event Trigger run. Event type is customer-defined field and take on any values of string type.
@triggerBody().event.subject	Subject of the custom event that caused the trigger to fire.
@triggerBody().event.data._keyName_	Data field in custom event is a free from JSON blob, which customer can use to send messages and data. Please use data.*keyName* to reference each field. For example, @triggerBody().event.data.callback returns the value for the *callback* field stored under *data*.
@trigger().startTime	Time at which the trigger fired to invoke the pipeline run.

Next steps

- For information about how these variables are used in expressions, see Expression language & functions.
- To use trigger scope system variables in pipeline, see Reference trigger metadata in pipeline

Parameterizing mapping data flows

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Mapping data flows in Azure Data Factory and Synapse pipelines support the use of parameters. Define parameters inside of your data flow definition and use them throughout your expressions. The parameter values are set by the calling pipeline via the Execute Data Flow activity. You have three options for setting the values in the data flow activity expressions:

- Use the pipeline control flow expression language to set a dynamic value
- Use the data flow expression language to set a dynamic value
- Use either expression language to set a static literal value

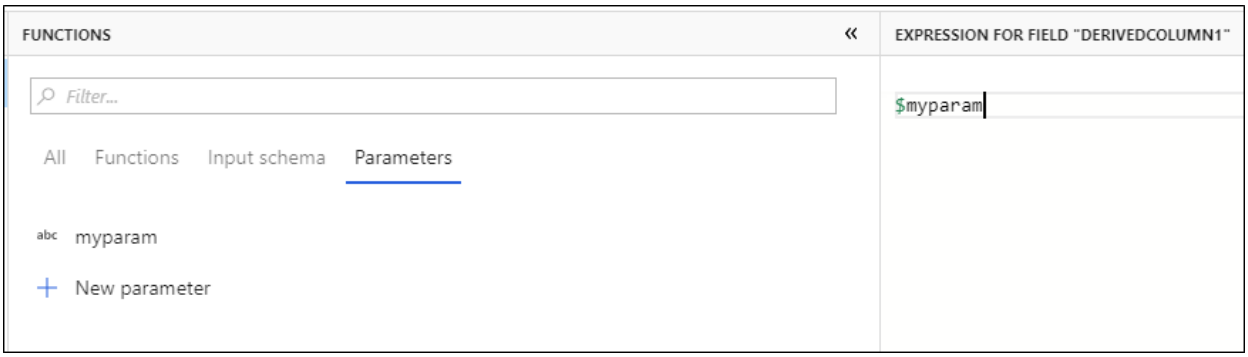
Use this capability to make your data flows general-purpose, flexible, and reusable. You can parameterize data flow settings and expressions with these parameters.

Create parameters in a mapping data flow

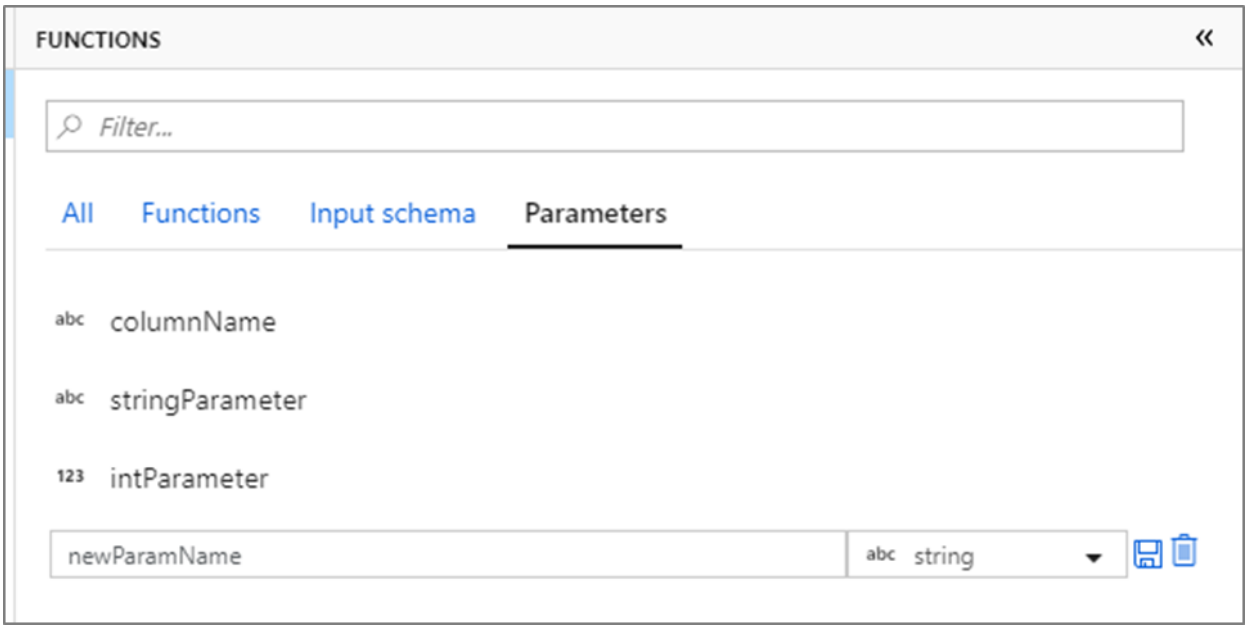
To add parameters to your data flow, click on the blank portion of the data flow canvas to see the general properties. In the settings pane, you will see a tab called **Parameter**. Select **New** to generate a new parameter. For each parameter, you must assign a name, select a type, and optionally set a default value.

Use parameters in a mapping data flow

Parameters can be referenced in any data flow expression. Parameters begin with $ and are immutable. You will find the list of available parameters inside of the Expression Builder under the **Parameters** tab.



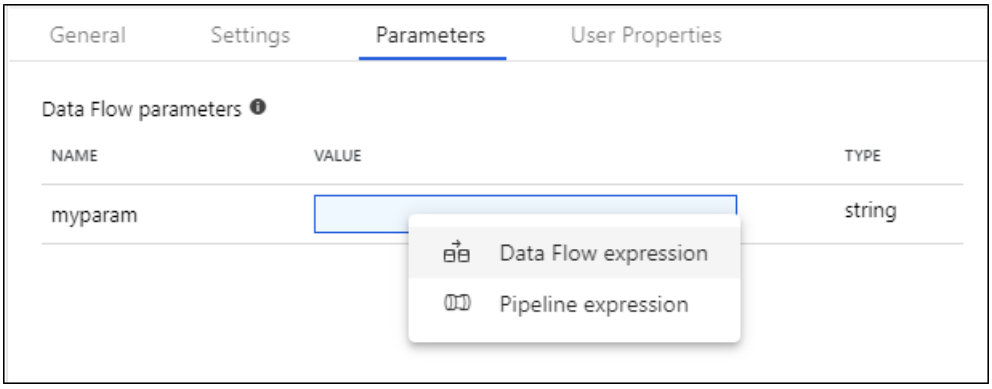
You can quickly add additional parameters by selecting **New parameter** and specifying the name and type.



Assign parameter values from a pipeline

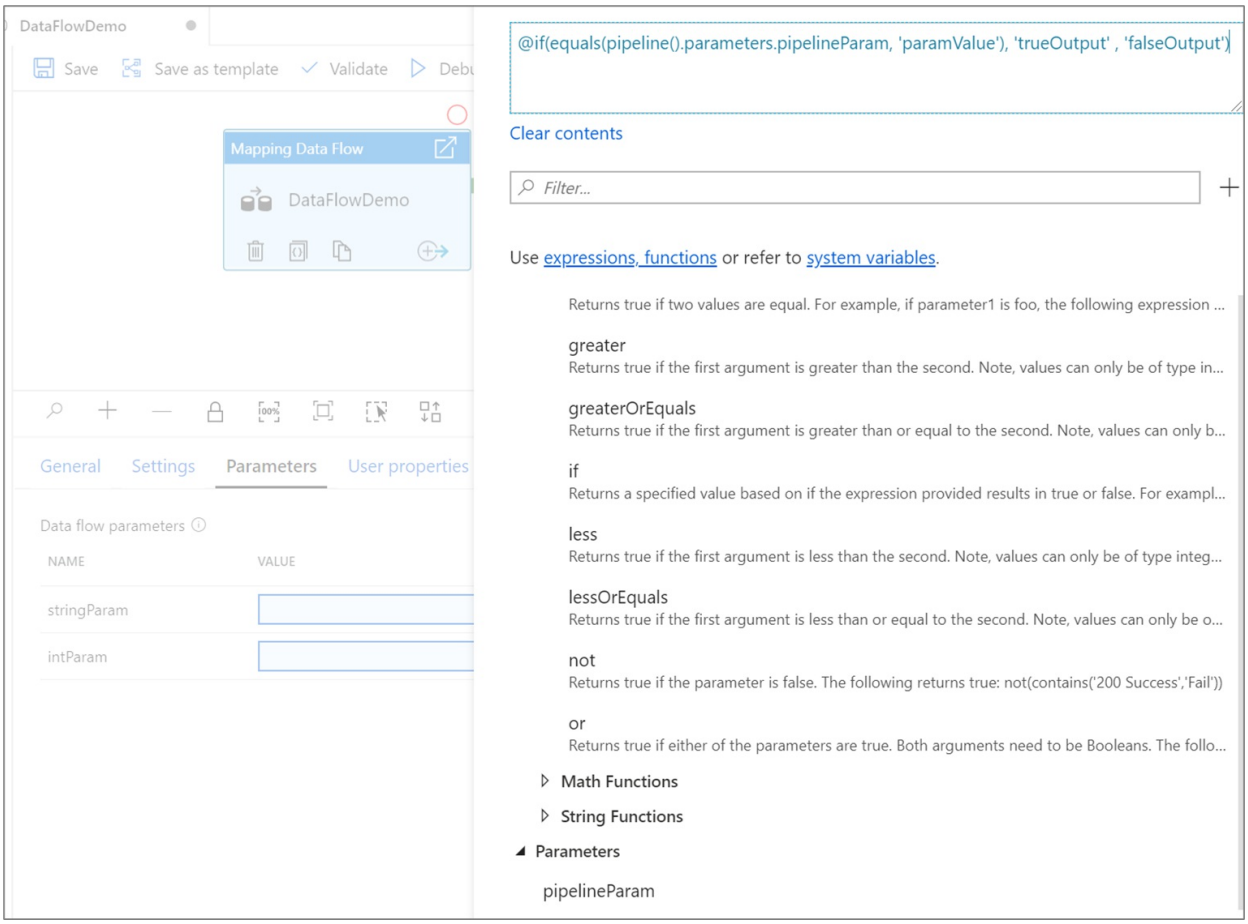
Once you've created a data flow with parameters, you can execute it from a pipeline with the Execute Data Flow Activity. After you add the activity to your pipeline canvas, you will be presented with the available data flow parameters in the activity's **Parameters** tab.

When assigning parameter values, you can use either the pipeline expression language or the data flow expression language based on spark types. Each mapping data flow can have any combination of pipeline and data flow expression parameters.



Pipeline expression parameters

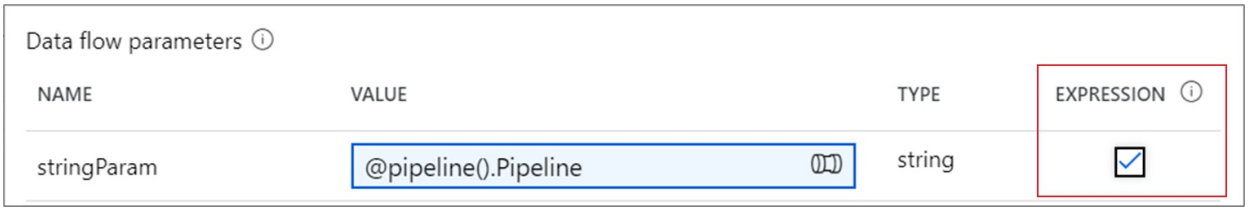
Pipeline expression parameters allow you to reference system variables, functions, pipeline parameters, and variables similar to other pipeline activities. When you click **Pipeline expression**, a side-nav will open allowing you to enter an expression using the expression builder.



When referenced, pipeline parameters are evaluated and then their value is used in the data flow expression language. The pipeline expression type doesn't need to match the data flow parameter type.

String literals vs expressions

When assigning a pipeline expression parameter of type string, by default quotes will be added and the value will be evaluated as a literal. To read the parameter value as a data flow expression, check the expression box next to the parameter.

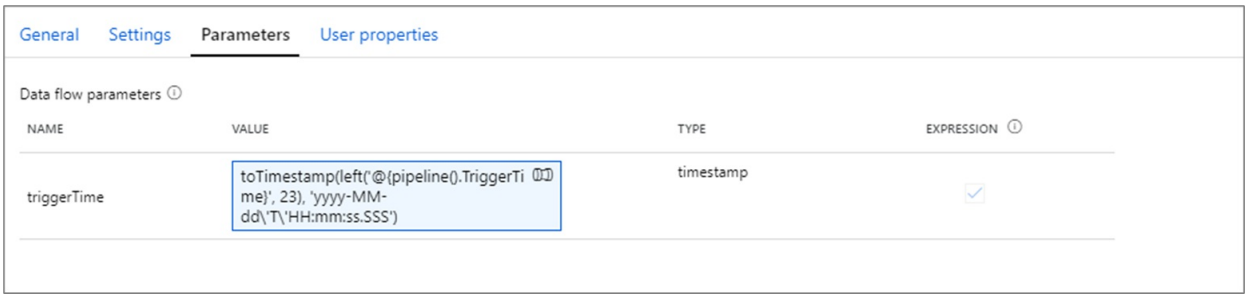


If data flow parameter `stringParam` references a pipeline parameter with value `upper(column1)`.

- If expression is checked, `$stringParam` evaluates to the value of column1 all uppercase.
- If expression is not checked (default behavior), `$stringParam` evaluates to `'upper(column1)'`

Passing in timestamps

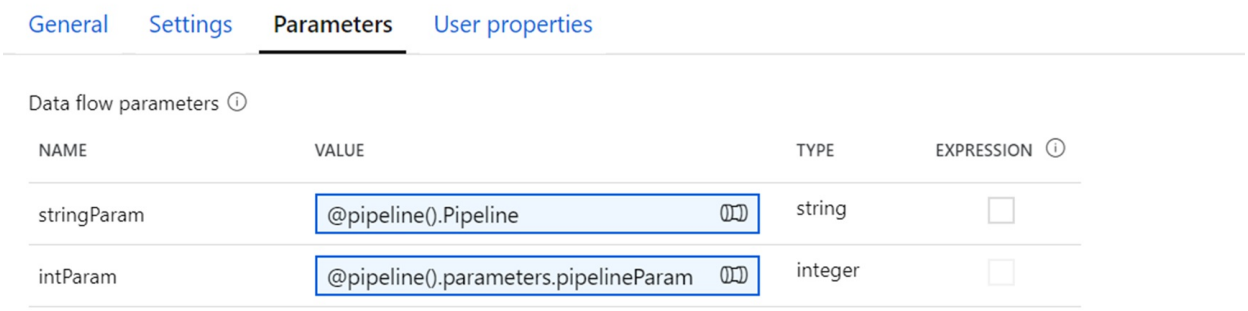
In the pipeline expression language, System variables such as `pipeline().TriggerTime` and functions like `utcNow()` return timestamps as strings in format 'yyyy-MM-dd'T'HH:mm:ss.SSSSSSZ'. To convert these into data flow parameters of type timestamp, use string interpolation to include the desired timestamp in a `toTimestamp()` function. For example, to convert the pipeline trigger time into a data flow parameter, you can use `toTimestamp(left('@{pipeline().TriggerTime}', 23), 'yyyy-MM-dd\'T\'HH:mm:ss.SSS')`.



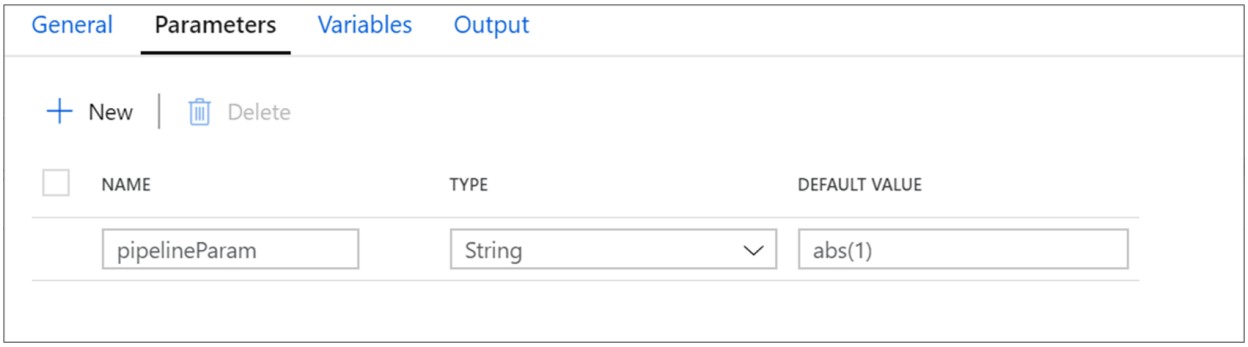
> **NOTE**
> Data Flows can only support up to 3 millisecond digits. The `left()` function is used trim off additional digits.

Pipeline parameter example

Say you have an integer parameter `intParam` that is referencing a pipeline parameter of type String, `@pipeline.parameters.pipelineParam`.



`@pipeline.parameters.pipelineParam` is assigned a value of `abs(1)` at runtime.



When `$intParam` is referenced in an expression such as a derived column, it will evaluate `abs(1)` return `1`.

Data flow expression parameters

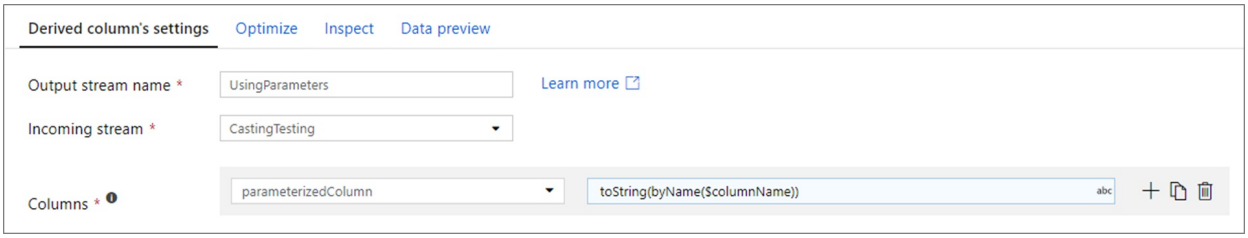
Select **Data flow expression** will open up the data flow expression builder. You will be able to reference functions, other parameters and any defined schema column throughout your data flow. This expression will be evaluated as is when referenced.

> **NOTE**
> If you pass in an invalid expression or reference a schema column that doesn't exist in that transformation, the parameter will evaluate to null.

Passing in a column name as a parameter

A common pattern is to pass in a column name as a parameter value. If the column is defined in the data flow schema, you can reference it directly as a string expression. If the column isn't defined in the schema, use the `byName()` function. Remember to cast the column to its appropriate type with a casting function such as `toString()`.

For example, if you wanted to map a string column based upon a parameter `columnName`, you can add a derived column transformation equal to `toString(byName($columnName))`.



Next steps

- Execute data flow activity
- Control flow expressions

Use Synapse Studio to monitor your workspace pipeline runs

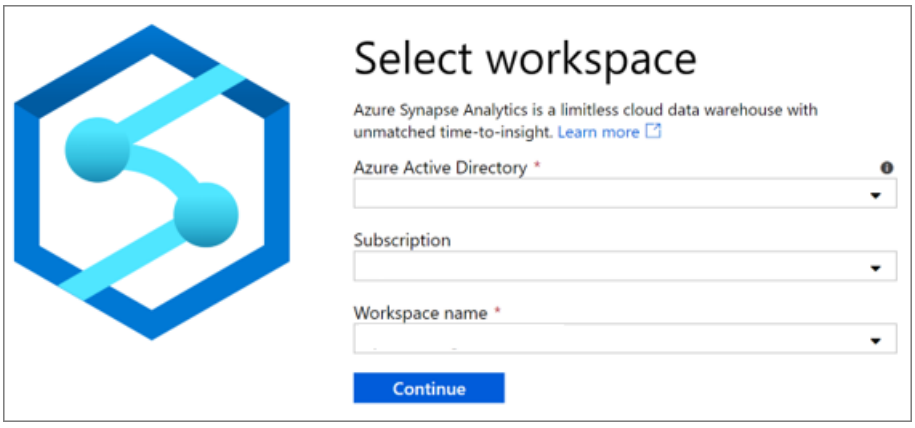
8/19/2022 • 2 minutes to read • Edit Online

With Azure Synapse Analytics, you can create complex pipelines that can automate and integrate your data movement, data transformation, and compute activities within your solution. You can author and monitor these pipelines using Synapse Studio.

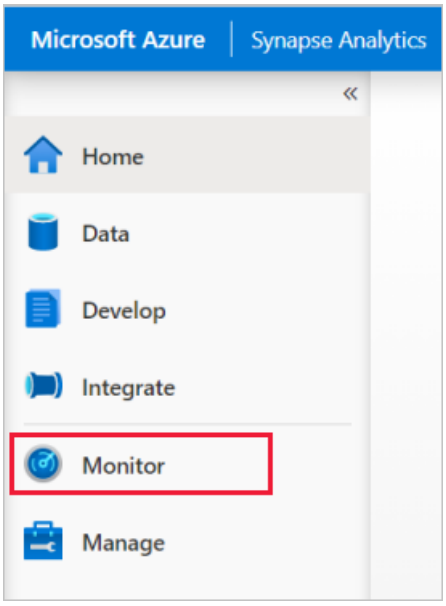
This article explains how to monitor your pipeline runs, which allows you to keep an eye on the latest status, issues, and progress of your pipelines.

Access pipeline runs list

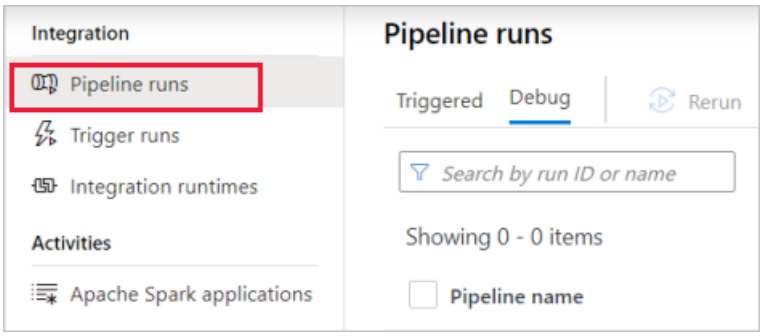
To see the list of pipeline runs in your workspace, first open the Synapse Studio and select your workspace.



Once you've opened your workspace, select the **Monitor** section on the left.



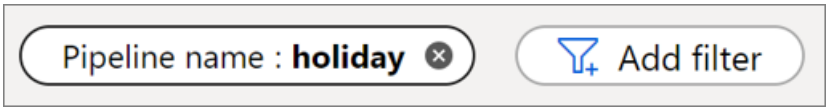
Select **Pipeline runs** to view the list of pipeline runs.



Filter your pipeline runs

You can filter the list of pipeline runs to the ones you're interested in. The filters at the top of the screen allow you to specify a field on which you'd like to filter.

For example, you can filter the view to see only the pipeline runs for the pipeline named "holiday":



View details about a specific pipeline run

To view details about your pipeline run, select the pipeline run. Then view the activity runs associated with the pipeline run. If the pipeline is still running, you can monitor the progress.

Next steps

To learn more about monitoring applications, see the Monitor Apache Spark applications article.

Send an email with an Azure Data Factory or Azure Synapse pipeline

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

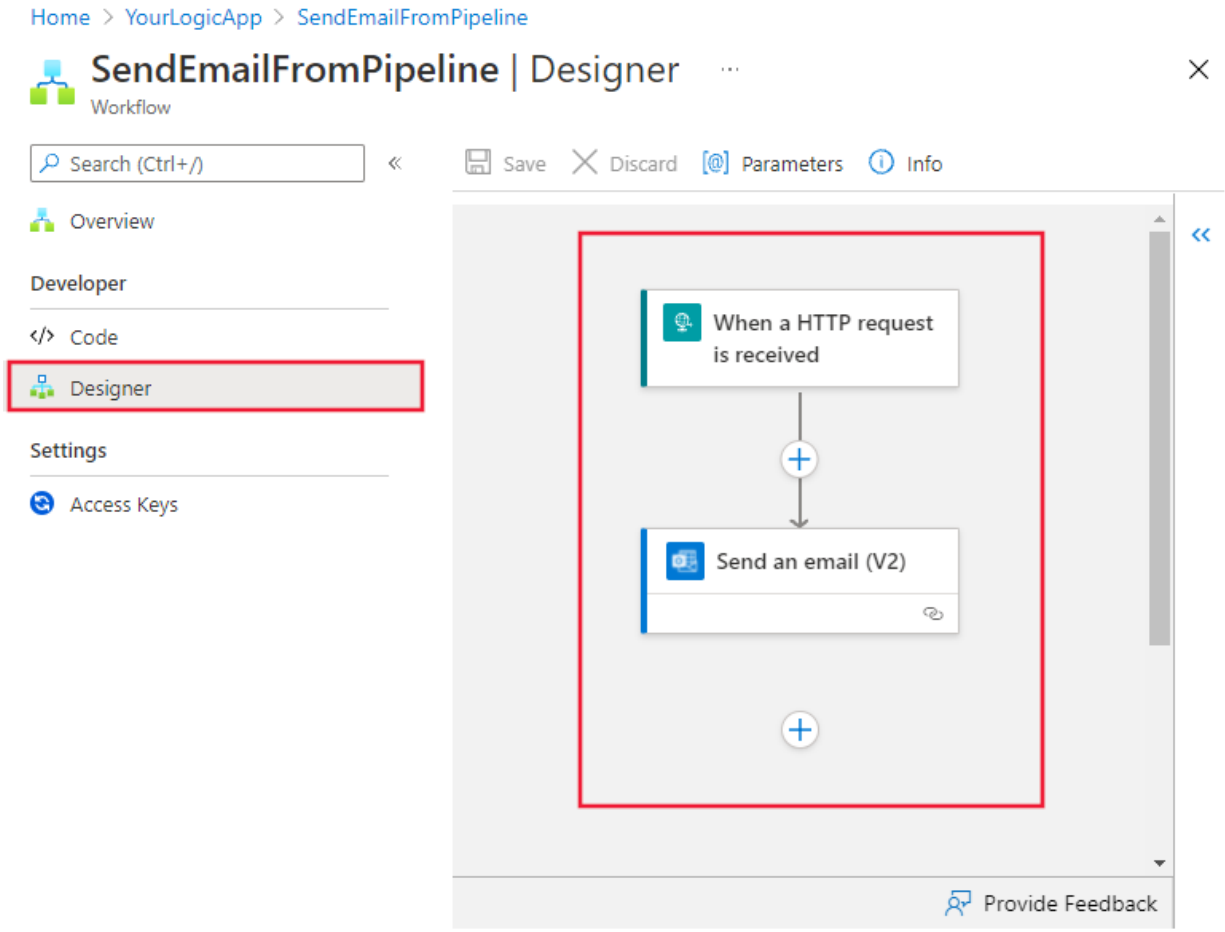
It's often necessary to send notifications during or after execution of a pipeline. Notification provides proactive alerting and reduces the need for reactive monitoring to discover issues. This article shows how to configure email notifications from an Azure Data Factory or Azure Synapse pipeline.

Prerequisites

- **Azure subscription**. If you don't have an Azure subscription, create a free account before you begin.
- **Logic App**. To trigger sending an email from the pipeline, you use Logic Apps to define the workflow. For details on creating a Logic App workflow, see How to create a logic app.

Create the email workflow in your Logic App

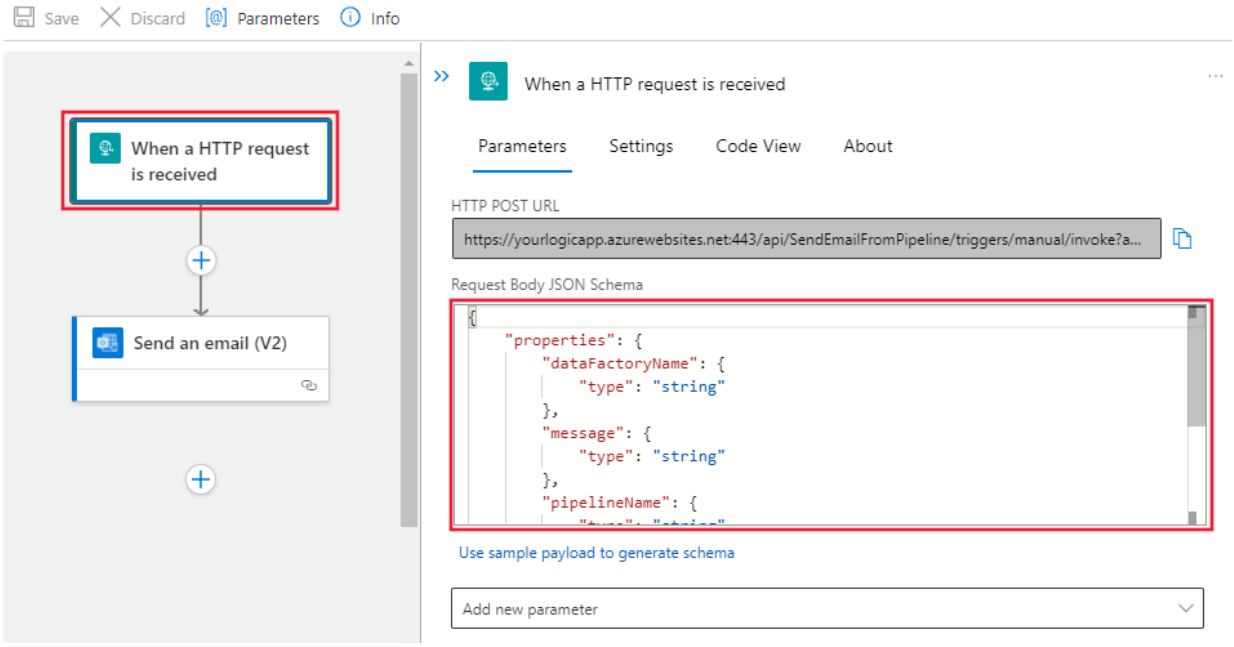
Create a Logic App workflow named `SendEmailFromPipeline`. Define the workflow trigger as `When an HTTP request is received`, and add an action of `Office 365 Outlook – Send an email (V2)`.



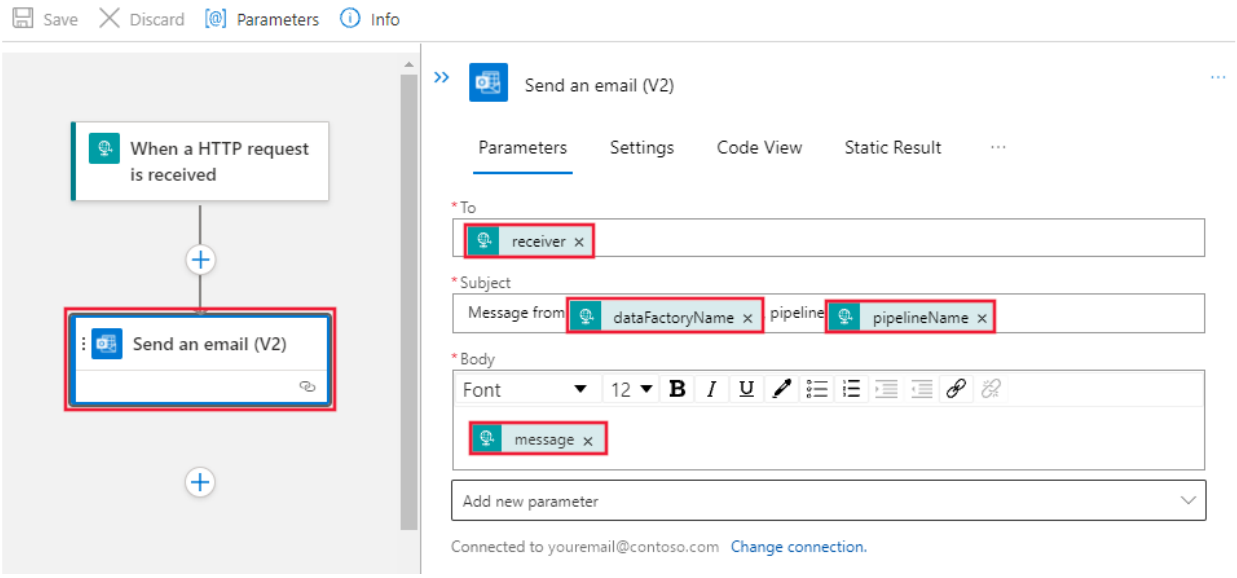
For the HTTP request trigger, provide this JSON for the `Request Body JSON Schema`:

```
{
    "properties": {
        "dataFactoryName": {
            "type": "string"
        },
        "message": {
            "type": "string"
        },
        "pipelineName": {
            "type": "string"
        },
        "receiver": {
            "type": "string"
        }
    },
    "type": "object"
}
```

The HTTP Request in the Logic App Designer should look like this:

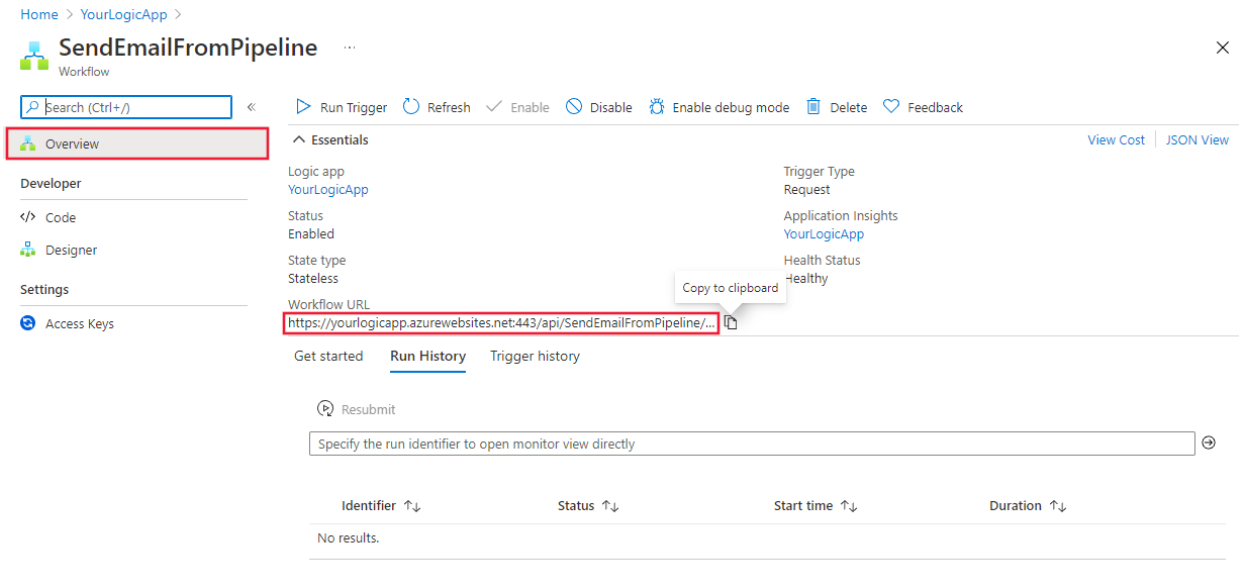


For the **Send Email (V2)** action, customize how you wish to format the email, using the properties from the request Body JSON schema:



Save the workflow. Browse to the Overview page for the workflow. Make a note of the Workflow URL for your

new workflow then, highlighted in the image below:



> **NOTE**
>
> To find the Workflow URL you must browse to the workflow itself, not just the logic app that contains it. From the Workflows page of your logic app instance, choose the workflow and then navigate to its Overview page.

Create a pipeline to trigger your Logic App email workflow

Once you create the Logic App workflow to send email, you can trigger it from a pipeline using a **Web** activity.

1. Create a new pipeline and find the **Web** activity under the **General** category, to drag it onto the editing canvas.

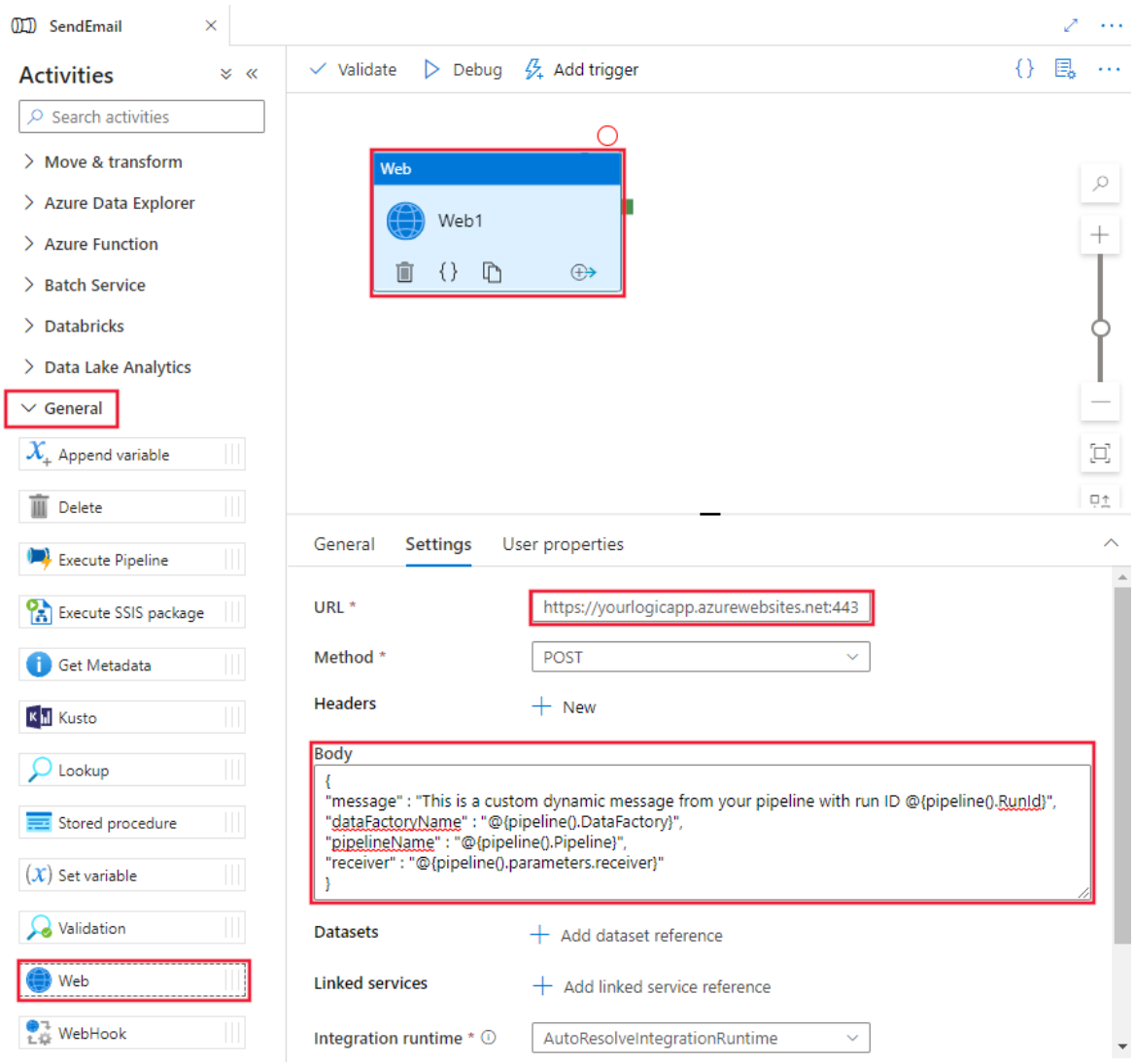
2. Select the new **Web1** activity, and then select the **Settings** tab.

   Provide the URL from the Logic App workflow you created previously in the **URL** field.

   Provide the following JSON for the **Body**:

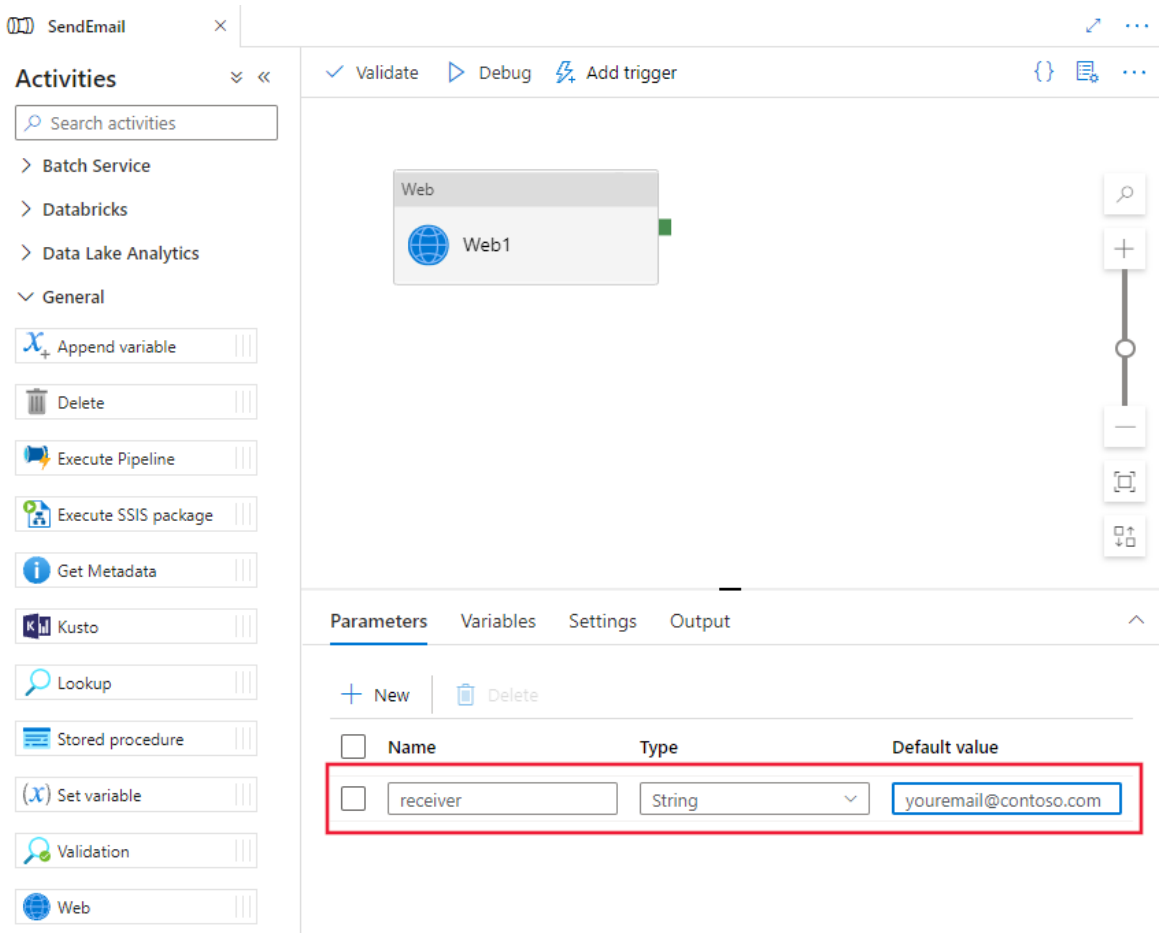
   ```
   {
    "message" : "This is a custom dynamic message from your pipeline with run ID
   @{pipeline().RunId}.",
    "dataFactoryName" : "@{pipeline().DataFactory}",
    "pipelineName" : "@{pipeline().Pipeline}",
    "receiver" : "@{pipeline().parameters.receiver}"
   }
   ```

   Use dynamic expressions to generate useful messages for events in your pipelines. Notice that the JSON format here matches the JSON format you defined in the Logic App, and you can also customize these as required.

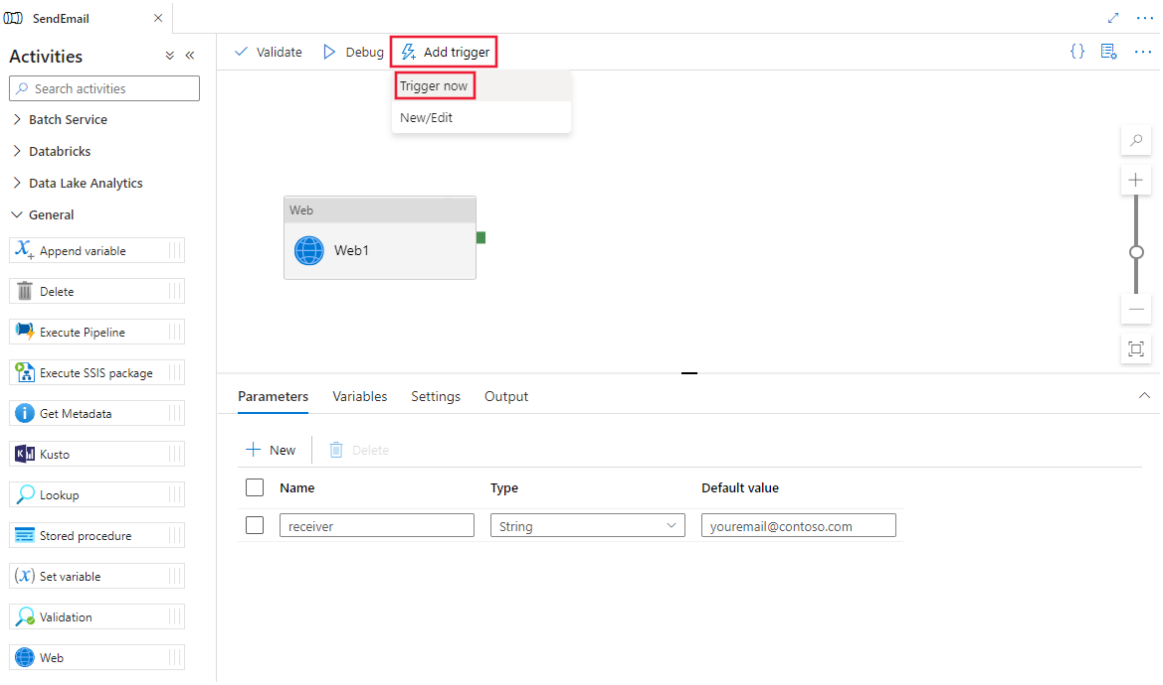


3. Select the background area of the pipeline designer to select the pipeline properties page and add a new parameter called receiver, providing an email address as its Default value.

   In this example, we provide the receiver email from a pipeline parameter we define arbitrarily. The receiver value could be taken from any expression, or even linked data sources.



4. Publish your pipeline, and then trigger it manually to confirm the email is sent as expected.



Add dynamic messages with system variables and expressions

You can use system variables and expressions to make your messages dynamic. For example:

- `@activity("CopyData").output.errors[0].Message`
- `@activity("DataFlow").error.Message`

The above expressions will return the relevant error messages from a Copy activity failure, which can be

redirected then to your Web activity that sends the email. Refer to the Copy activity output properties article for more details.

Next steps

How to send Teams notifications from a pipeline

Send notifications to a Microsoft Teams channel from an Azure Data Factory or Synapse Analytics pipeline

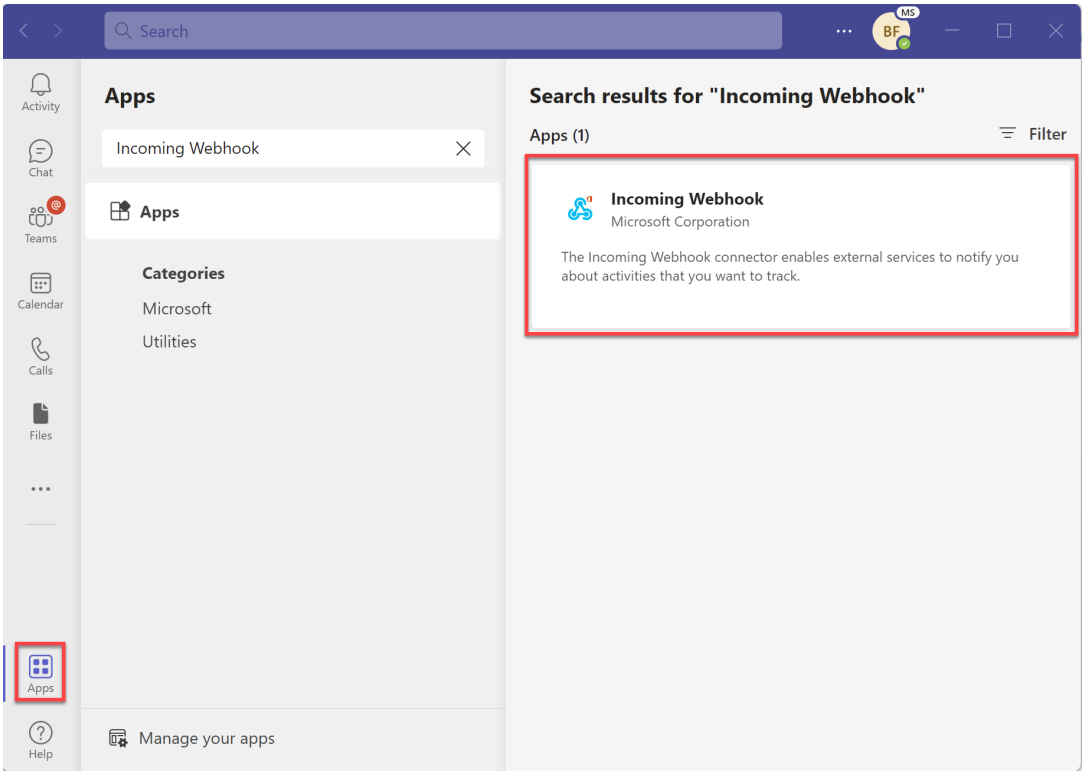
8/19/2022 • 11 minutes to read • Edit Online

It's often necessary to send notifications during or after execution of a pipeline. Notification provides proactive alerting and reduces the need for reactive monitoring to discover issues. You can learn about how to send email notifications using logic apps that a data factory or Synapse pipeline can invoke. Many enterprises are also increasingly using Microsoft Teams for collaboration. This article shows how to configure notifications from pipeline alerts into Microsoft Teams.

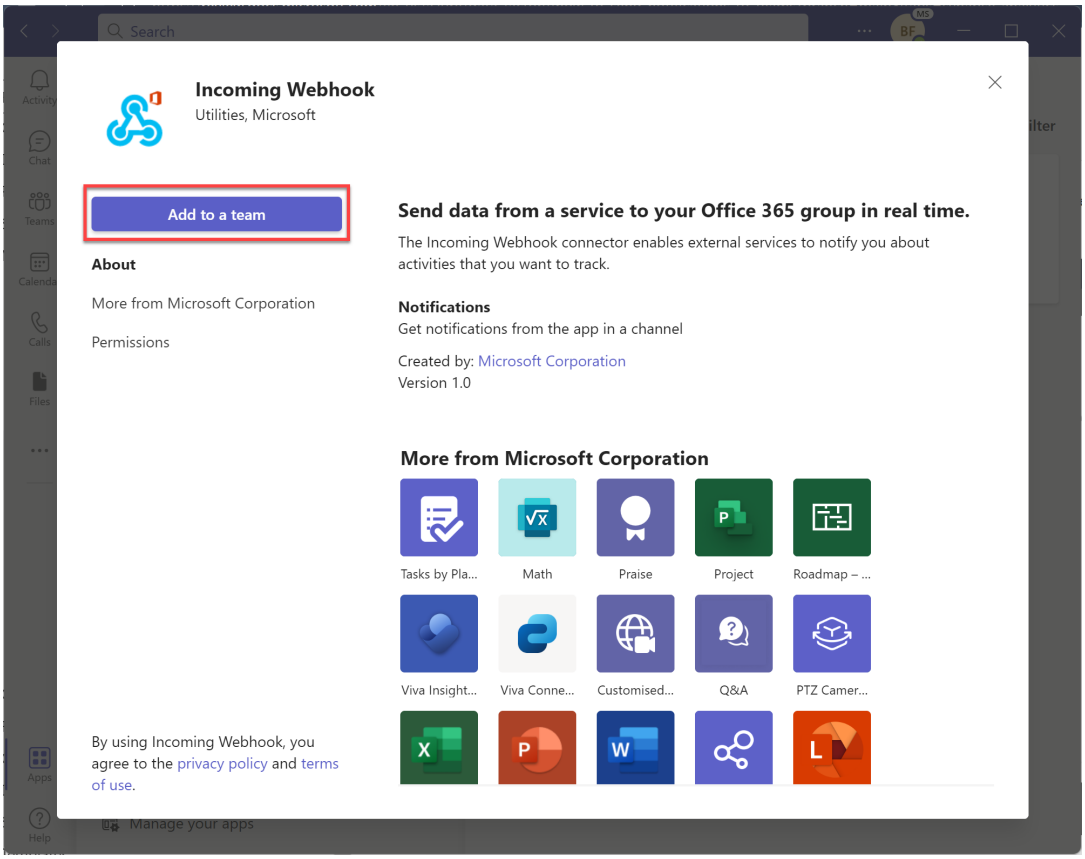
Prerequisites

Before you can send notifications to Teams from your pipelines, you must create an Incoming Webhook for your Teams channel. If you need to create a new Teams channel for this purpose, refer to the Teams documentation.

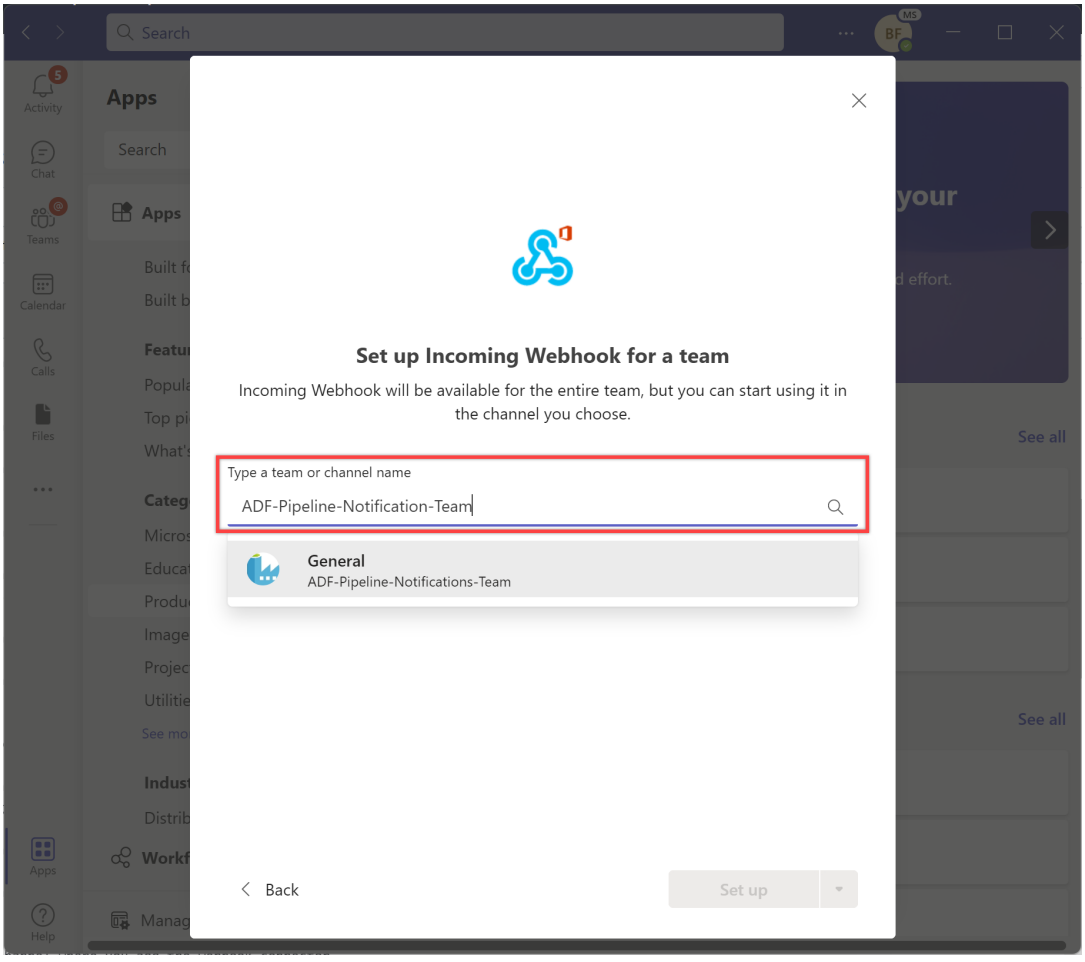
1. Open Microsoft Teams and go to the Apps tab. Search for "Incoming Webhook" and select the Incoming Webhook connector.



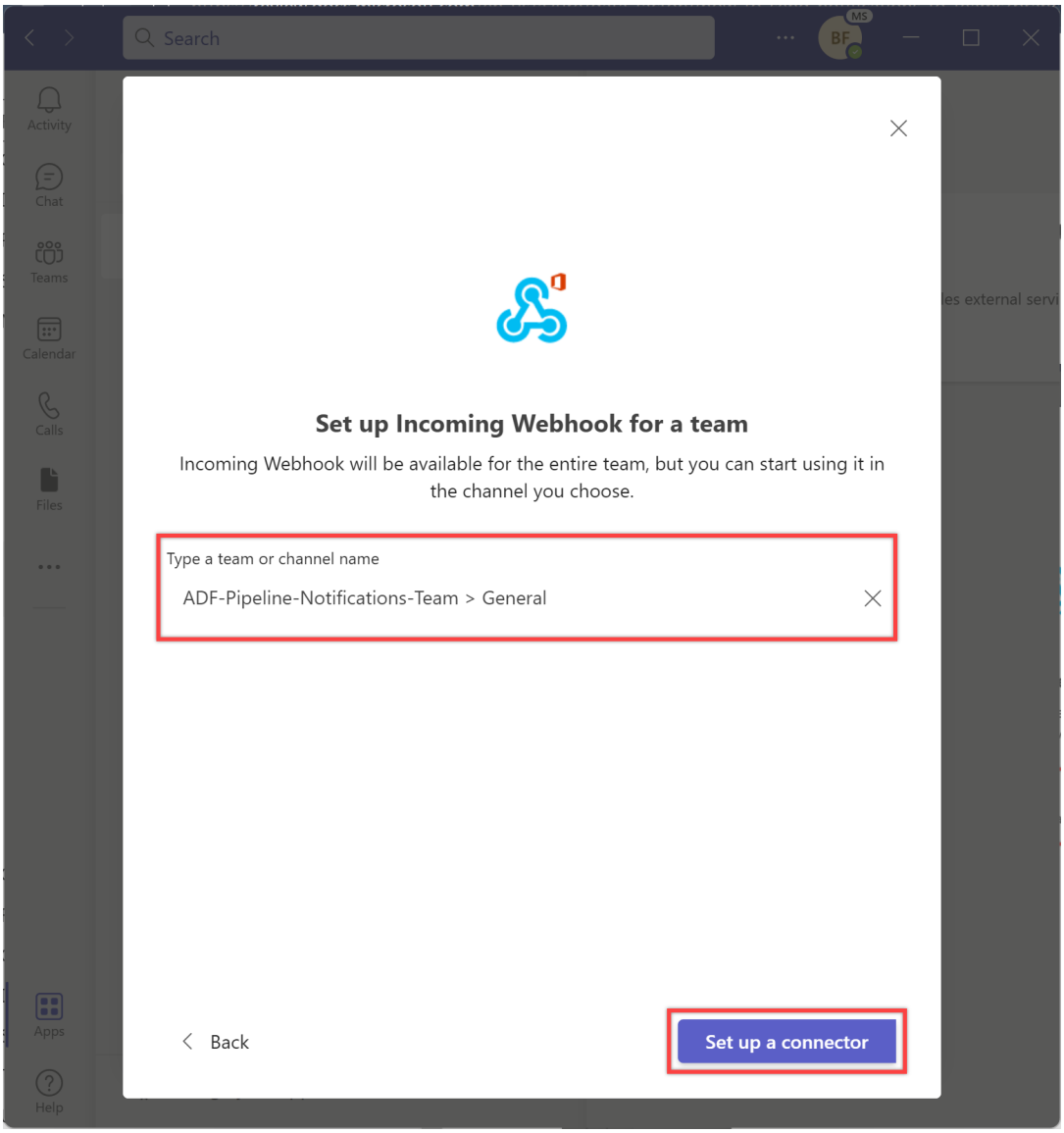
2. Select the "Add to a team" button to add the connector to the Team or Team channel name site where you want to send notifications.



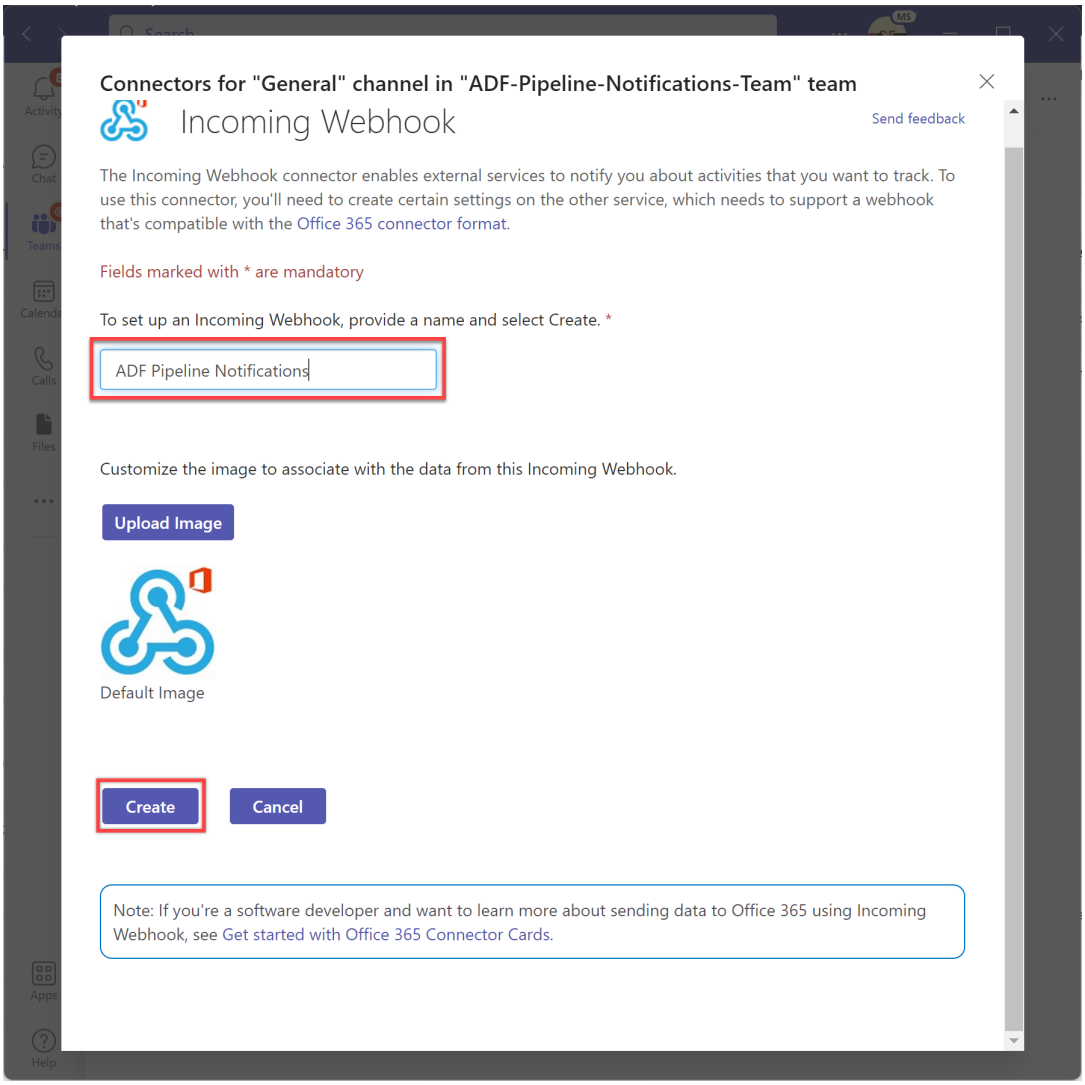
3. Type or select Team or Team channel name where you want to send the notifications.



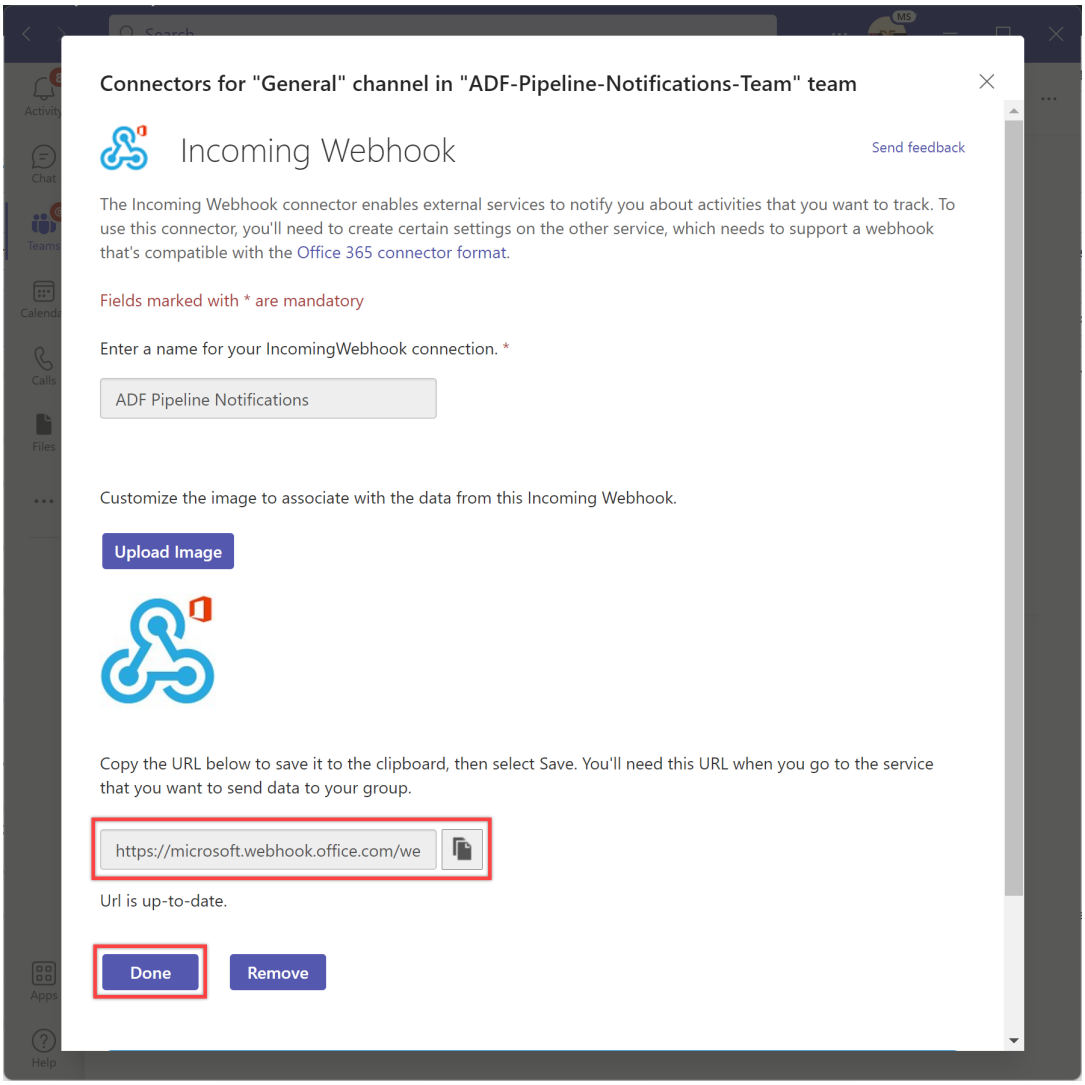
4. Select the "Set up a connector" button to set up the Incoming Webhook for the Team or Team channel name you selected in the previous step.



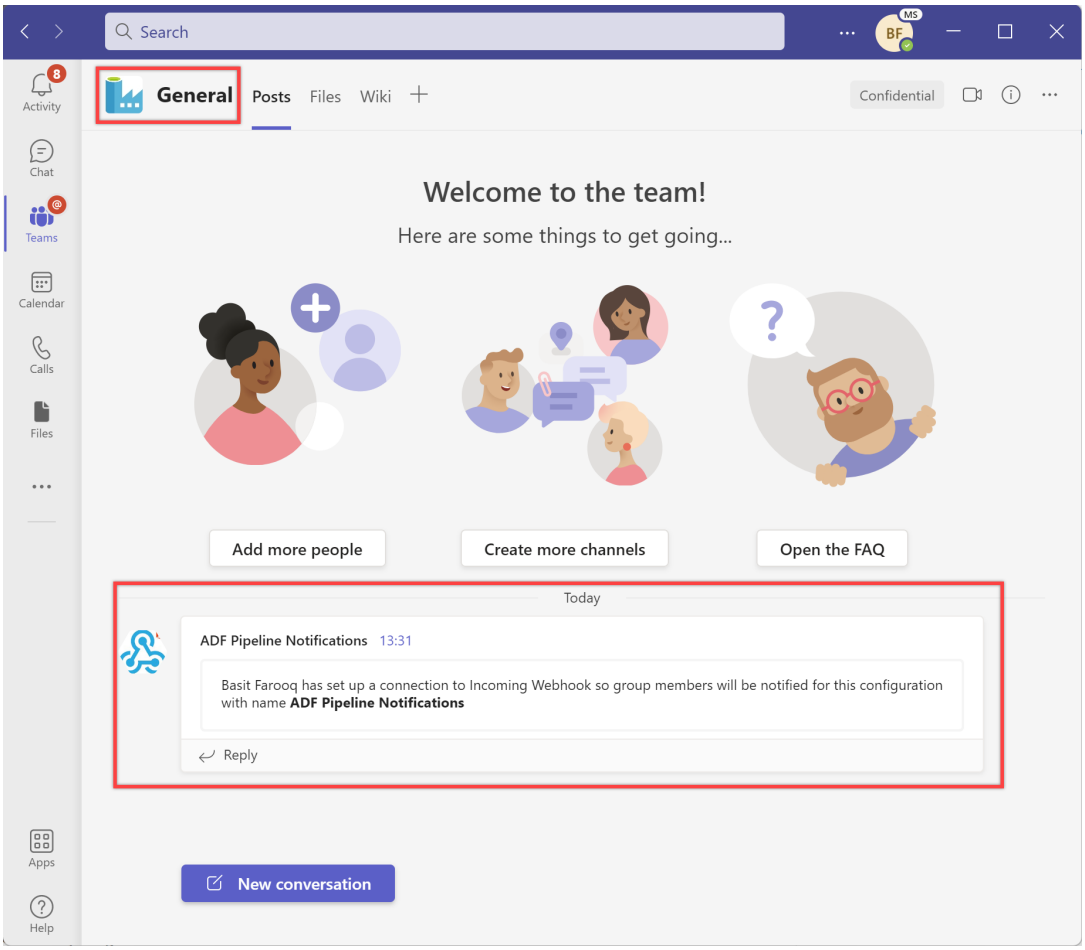
5. Name the Webhook as appropriate and optionally upload an icon to identify your messages. After that, select the "Create" button to create the Incoming Webhook.



6. Copy the Webhook URL that is generated on creation and save it for later use in pipeline. After that, select the "Done" button to complete the setup.



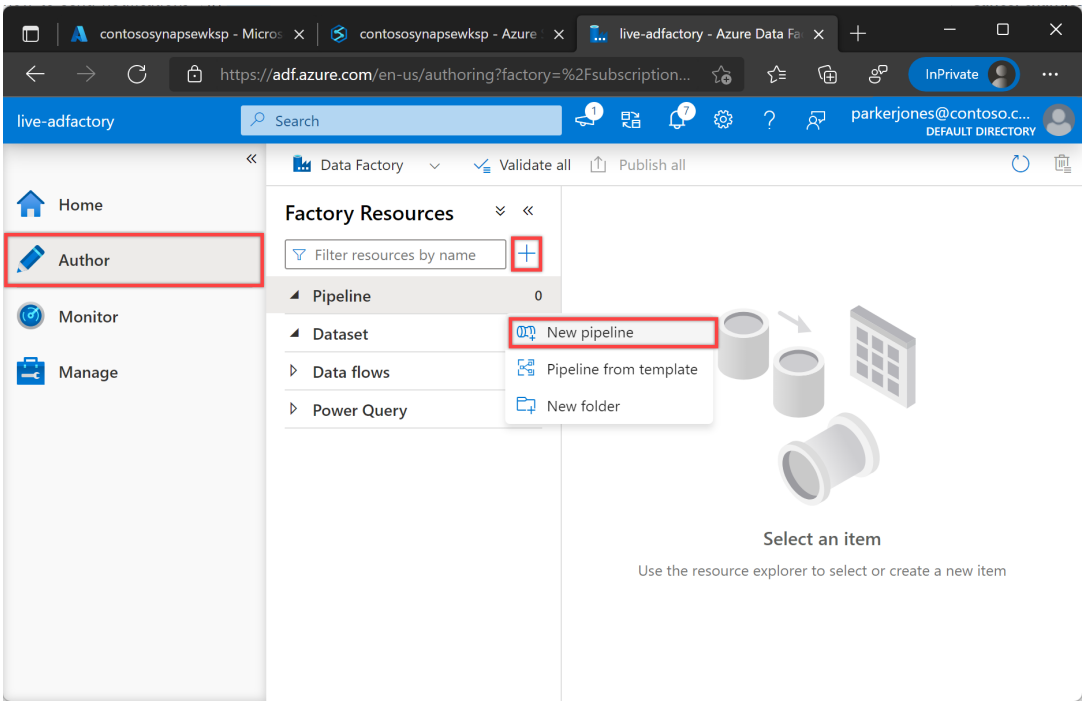
7. You can see the notification in the channel where you add the webhook connector.



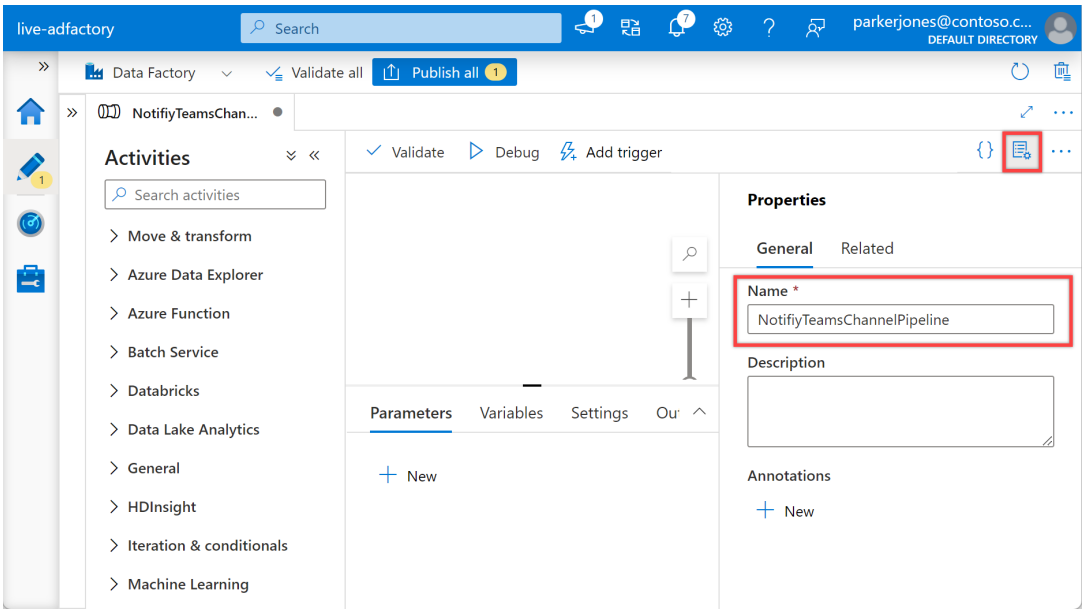
Steps to send notifications on Teams channel from a pipeline:

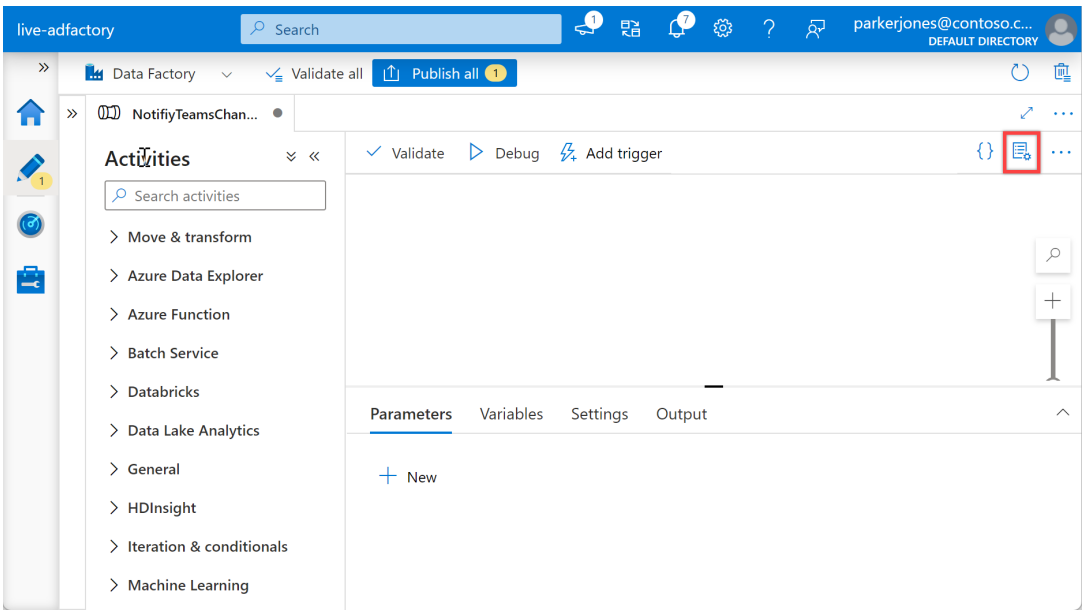
- Azure Data Factory
- Synapse Analytics

1. Select **Author** tab from the left pane.
2. Select the + (plus) button, and then select **New pipeline**.



3. In the "Properties" pane under "General", specify **NotifiyTeamsChannelPipeline** for **Name**. Then collapse the panel by clicking the **Properties** icon in the top-right corner.

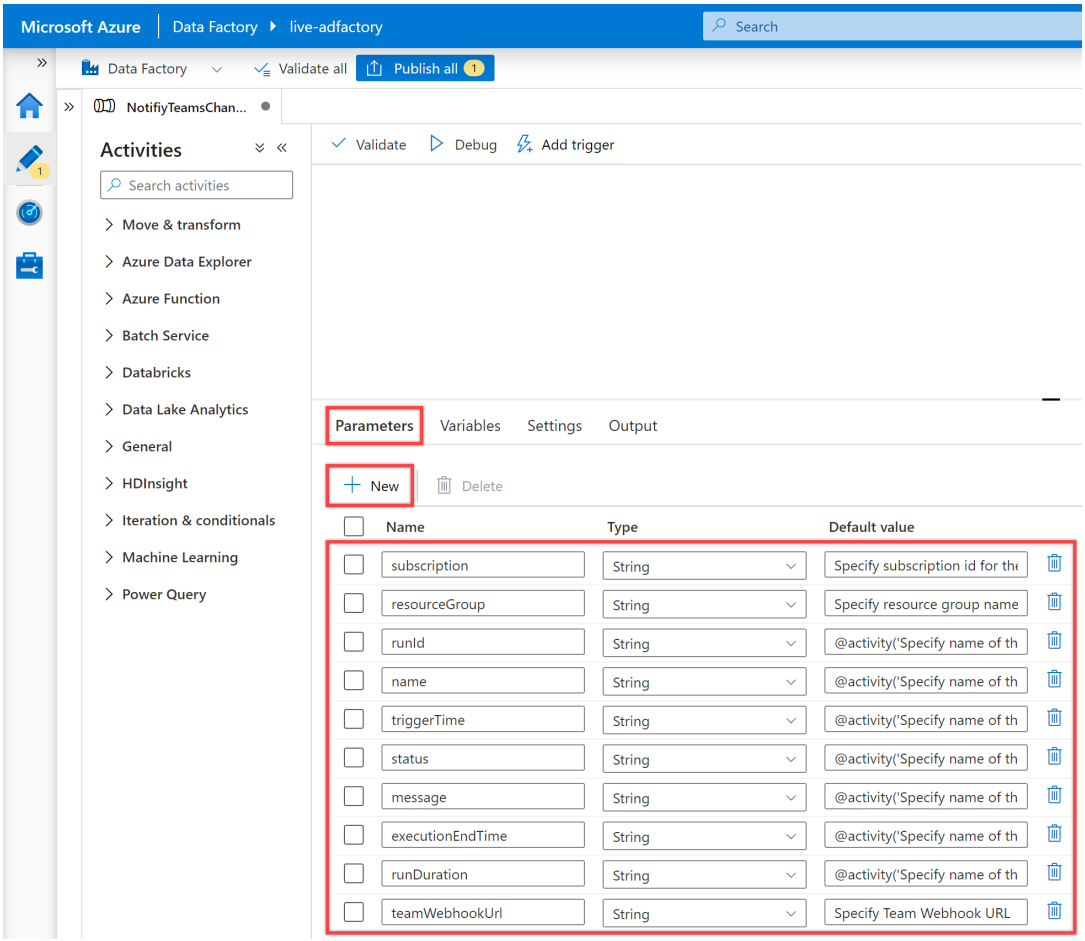




4. In the "Configurations" pane, select **Parameters**, and then select the **+ New** button define following parameters for your pipeline.

NAME	TYPE	DEFAULT VALUE
subscription	`String`	`Specify subscription id for the pipeline`
resourceGroup	`String`	`Specify resource group name for the pipeline`
runId	`String`	`@activity('Specify name of the calling pipeline').output['pipelineRunId']`
name	`String`	`@activity('Specify name of the calling pipeline').output['pipelineName']`
triggerTime	`String`	`@activity('Specify name of the calling pipeline').ExecutionStartTime`
status	`String`	`@activity('Specify name of the calling pipeline').Status`

NAME	TYPE	DEFAULT VALUE
message	`String`	`@activity('Specify name of the calling pipeline').Error['message']`
executionEndTime	`String`	`@activity('Specify name of the calling pipeline').ExecutionEndTime`
runDuration	`String`	`@activity('Specify name of the calling pipeline').Duration`
teamWebhookUrl	`String`	`Specify Team Webhook URL`



NOTE

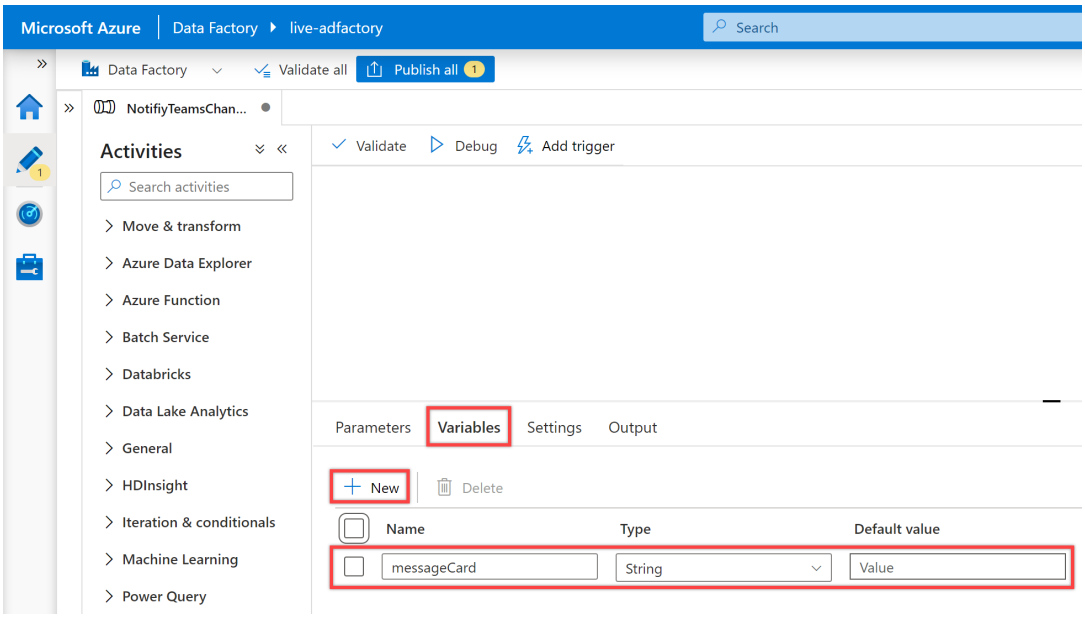
These parameters are used to construct the monitoring URL. Suppose you do not provide a valid subscription and resource group (of the same data factory where the pipelines belong). In that case, the notification will not contain a valid pipeline monitoring URL, but the messages will still work. Additionally, adding these parameters helps prevent the need to always pass those values from another pipeline. If you intend to control those values through a metadata-driven approach, then you should modify them accordingly.

TIP

We recommend adding the current Data Factory **Subscription ID**, **Resource Group**, and the **Teams webhook URL** (refer to prerequisites) for the default value of the relevant parameters.

5. In the "Configurations" pane, select **Variables**, and then select the **+ New** button define following variables for your pipeline.

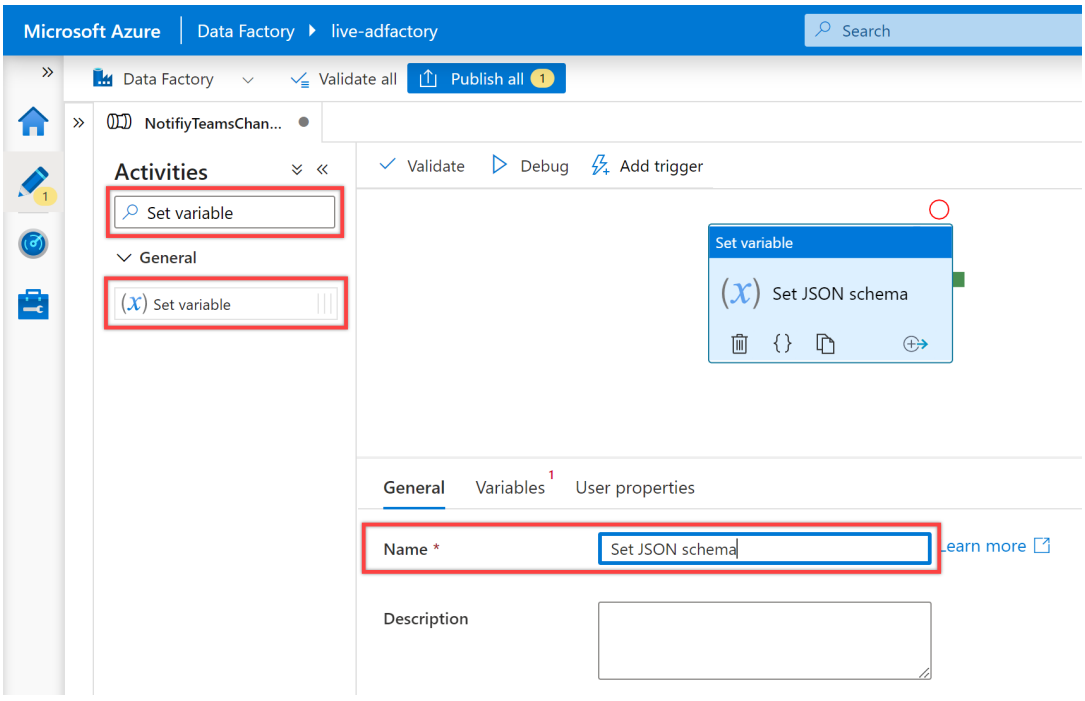
NAME	TYPE	DEFAULT VALUE
messageCard	`String`	



6. Search for "Set variable" in the pipeline "Activities" pane, and drag a **Set Variable** activity to the pipeline canvas.

7. Select the **Set Variable** activity on the canvas if it isn't already selected, and its "General" tab, to edit its details.

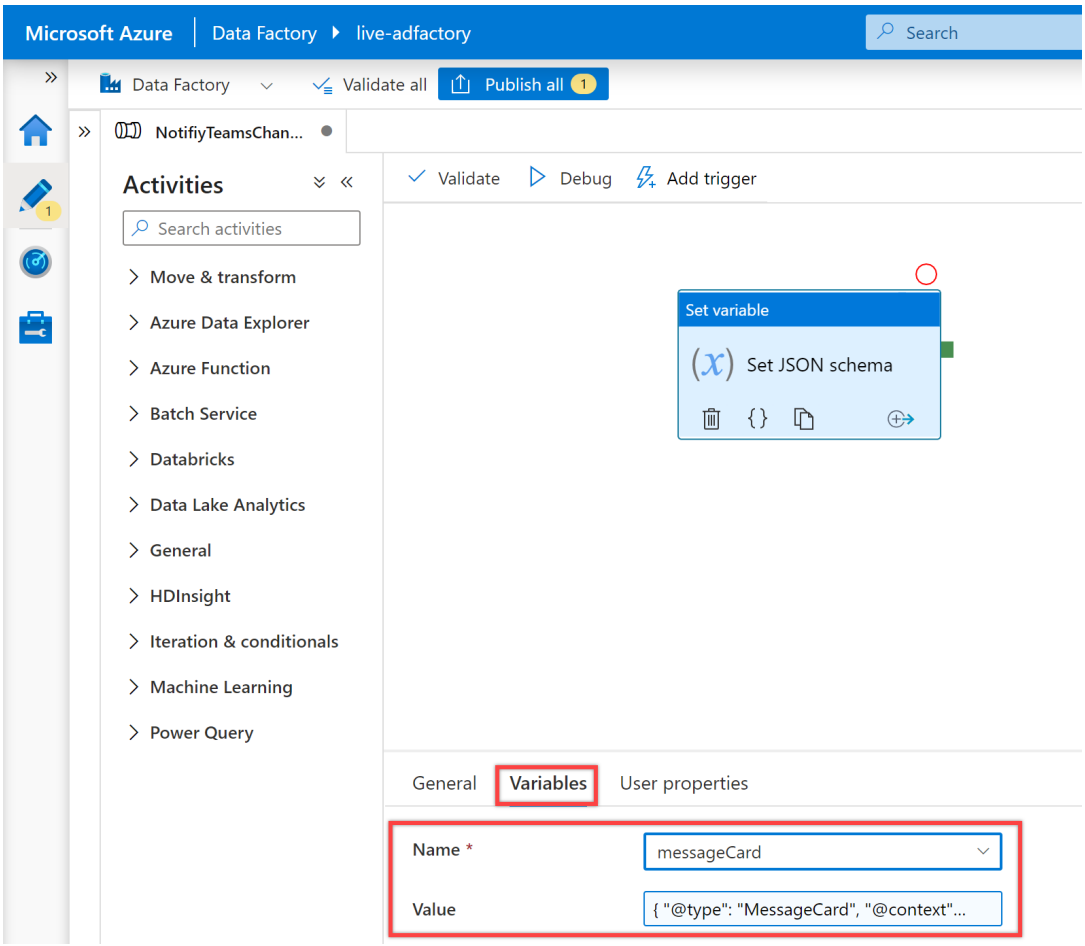
8. In the "General" tab, specify **Set JSON schema** for **Name** of the **Set Variable** activity.



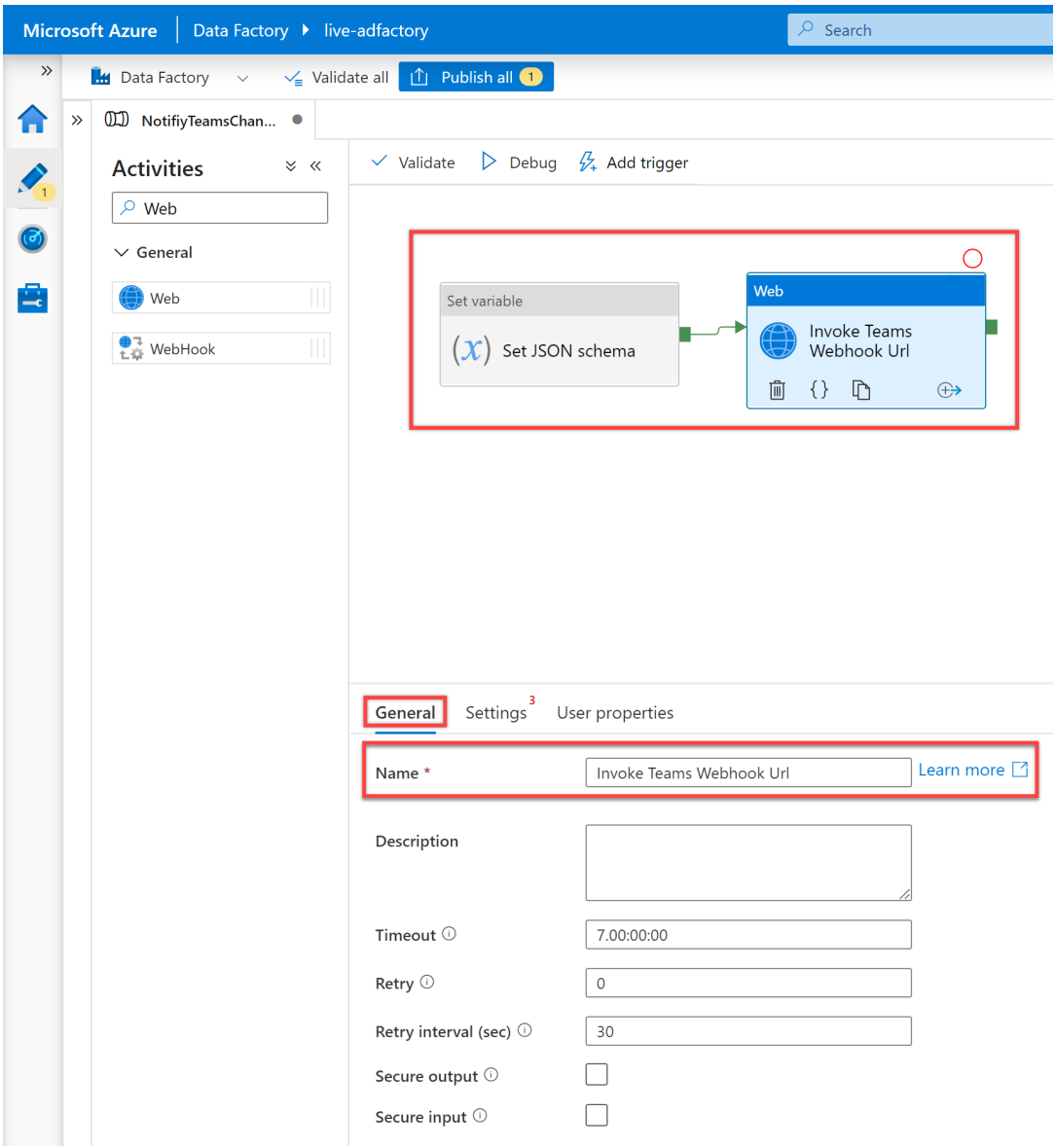
9. In the "Variables" tab, select **messageCard** variable for the **Name** property and enter the following **JSON** for its **Value** property:

```
{
    "@type": "MessageCard",
    "@context": "http://schema.org/extensions",
    "themeColor": "0076D7",
    "summary": "Pipeline status alert message",
    "sections": [
        {
```

```
            "activityTitle": "Pipeline execution alert",
            "facts": [
                {
                    "name": "Subscription Id:",
                    "value": "@{pipeline().parameters.subscription}"
                },
                {
                    "name": "Resource Group:",
                    "value": "@{pipeline().parameters.resourceGroup}"
                },
                {
                    "name": "Data Factory Name:",
                    "value": "@{pipeline().DataFactory}"
                },
                {
                    "name": "Pipeline RunId:",
                    "value": "@{pipeline().parameters.runId}"
                },
                {
                    "name": "Pipline Name:",
                    "value": "@{pipeline().parameters.name}"
                },
                {
                    "name": "Pipeline Status:",
                    "value": "@{pipeline().parameters.status}"
                },
                {
                    "name": "Execution Start Time (UTC):",
                    "value": "@{pipeline().parameters.triggerTime}"
                },
                {
                    "name": "Execution Finish Time (UTC):",
                    "value": "@{pipeline().parameters.executionEndTime}"
                },
                {
                    "name": "Execution Duration (s):",
                    "value": "@{pipeline().parameters.runDuration}"
                },
                {
                    "name": "Message:",
                    "value": "@{pipeline().parameters.message}"
                },
                {
                    "name": "Notification Time (UTC):",
                    "value": "@{utcnow()}"
                }
            ],
            "markdown": true
        }
    ],
    "potentialAction": [
        {
            "@type": "OpenUri",
            "name": "View pipeline run",
            "targets": [
                {
                    "os": "default",
                    "uri":
"@{concat('https://adf.azure.com/monitoring/pipelineruns/',pipeline().parameters.runId,'?
factory=/subscriptions/',pipeline().parameters.subscription,'/resourceGroups/',pipeline().parameters.
resourceGroup,'/providers/Microsoft.DataFactory/factories/',pipeline().DataFactory)}"
                }
            ]
        }
    ]
}
```

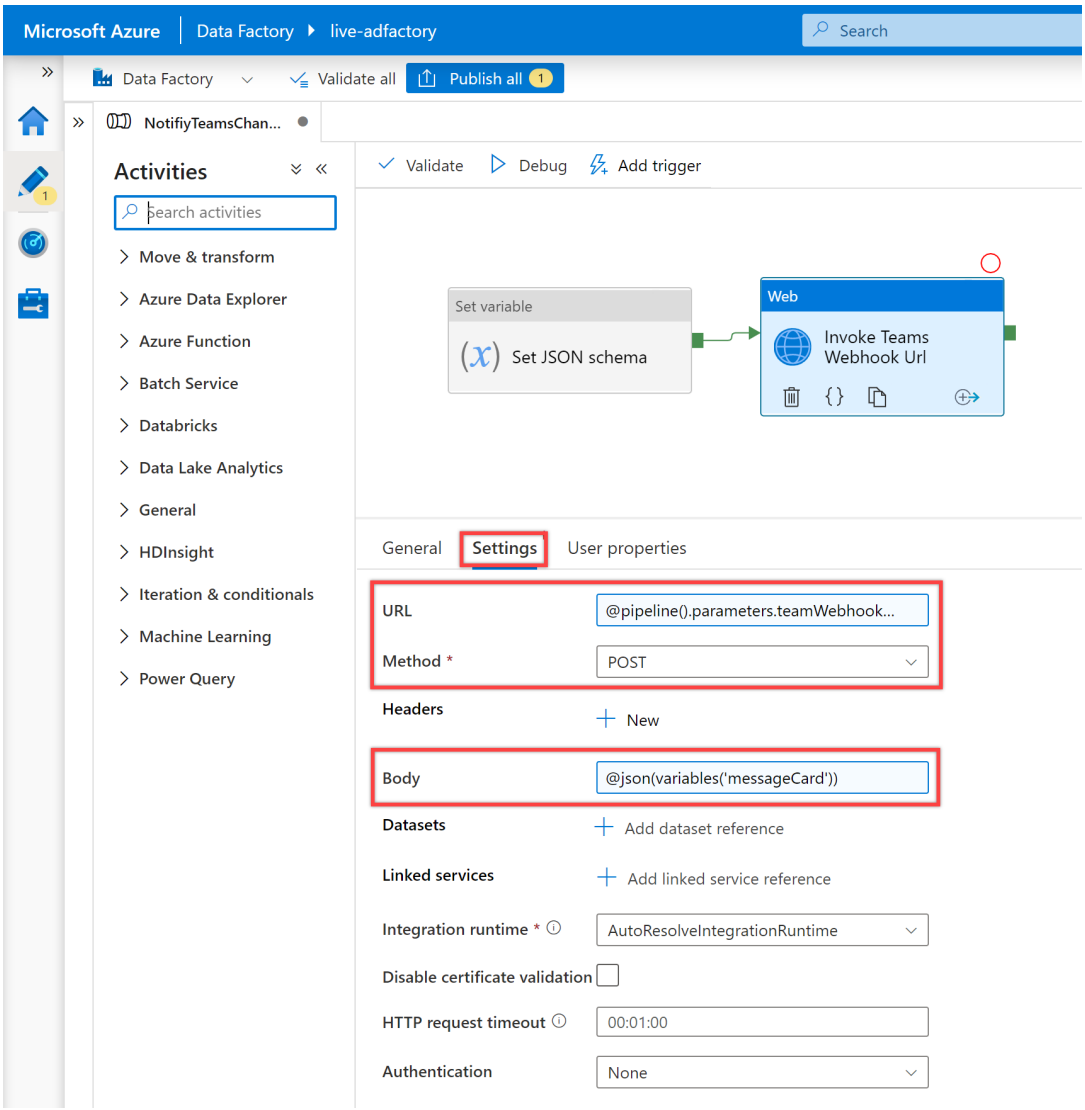


10. Search for "Web" in the pipeline "Activities" pane, and drag a **Web** activity to the pipeline canvas.
11. Create a dependency condition for the **Web** activity so that it only runs if the **Set Variable** activity succeeds. To create this dependency, select the green handle on the right side of the **Set Variable** activity, drag it, and connect it to the **Web** activity.
12. Select the new **Web** activity on the canvas if it isn't already selected, and its "General" tab, to edit its details.
13. In the "General" pane, specify **Invoke Teams Webhook Url** for **Name** of the **Web** activity.



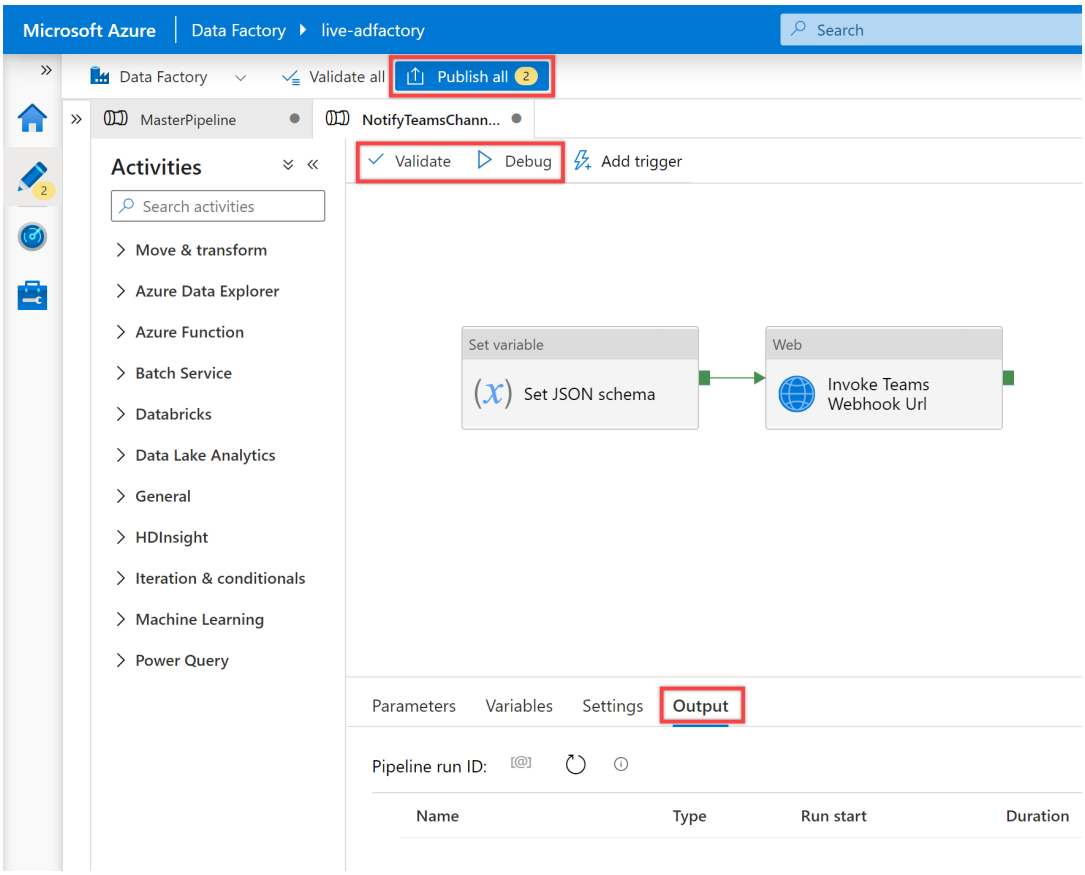
14. In the "Settings" pane, set following properties as follows:

PROPERTY	VALUE
URL	`@pipeline().parameters.teamWebhookUrl`
Method	`POST`
Body	`@json(variables('messageCard'))`



15. All set and now you're ready to validate, debug, and then publish your **NotifiyTeamsChannelPipeline** pipeline.

    - To validate the pipeline, select **Validate** from the tool bar.
    - To debug the pipeline, select **Debug** on the toolbar. You can see the status of the pipeline run in the "Output" tab at the bottom of the window.
    - Once the pipeline can run successfully, in the top toolbar, select **Publish all**. This action publishes entities you created to Data Factory. Wait until you see the **Successfully** published message.



Sample usage

In this sample usage scenario, we'll create a master pipeline with three **Execute Pipeline** activities. The first **Execute Pipeline** activity will invoke our ETL pipeline and the remaining two **Execute Pipeline** activities will invoke the "NotifiyTeamsChannelPipeline" pipeline to send relevant success or failure notifications to the Teams channel depending on the execution state of our ETL pipeline.

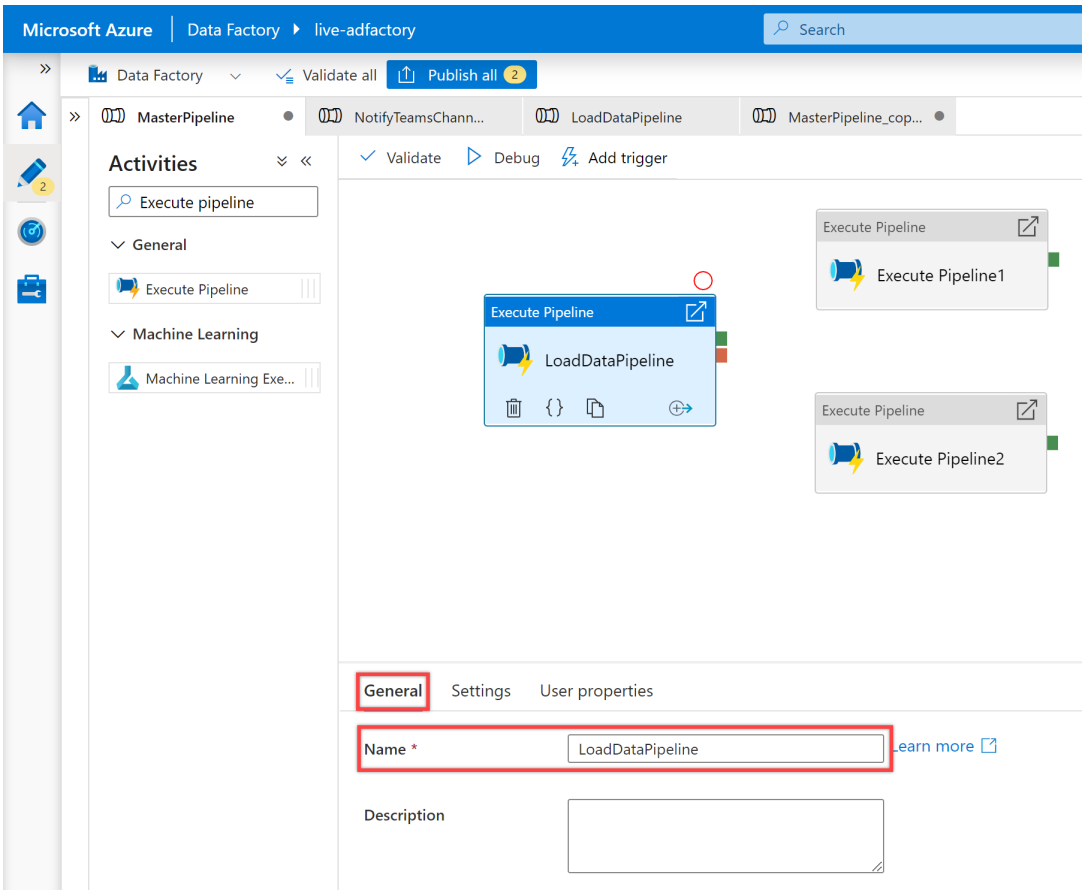
1. Select **Author** tab from the left pane in **Data Factory** or **Integrate** tab from the left pane in **Synapse Studio**. Next, select the + (plus) button, and then select **Pipeline** to create a new pipeline.

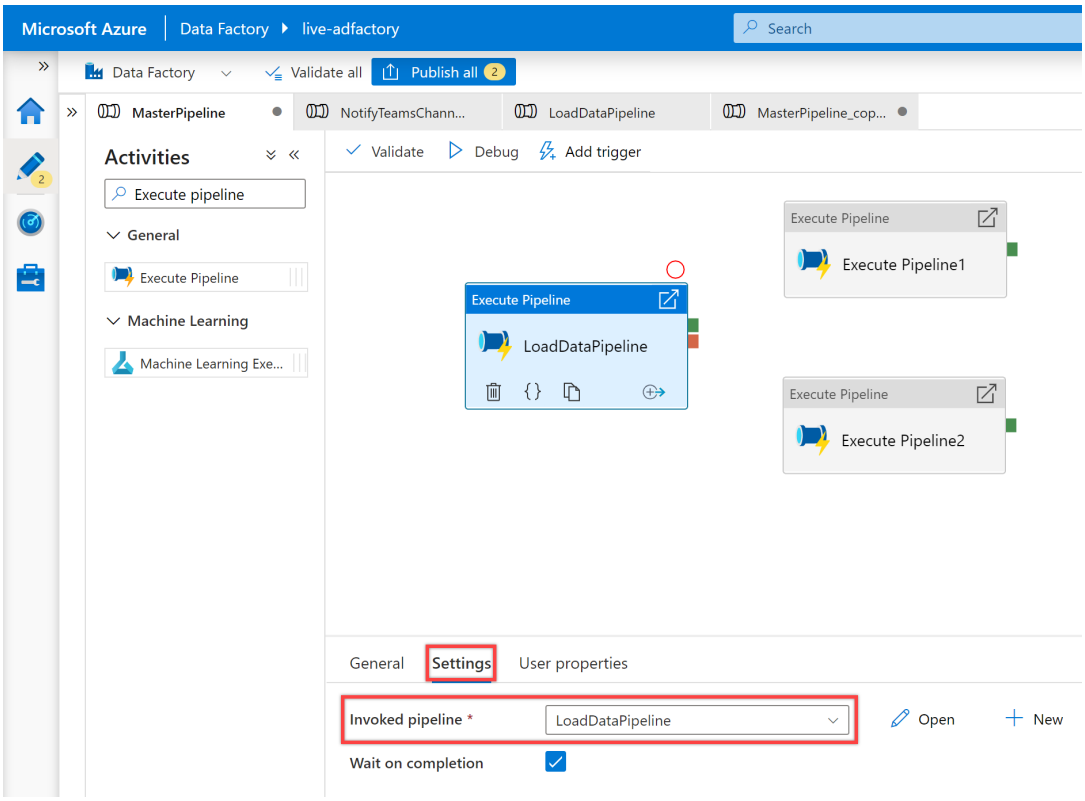
2. In the "General" panel under Properties, specify **MasterPipeline** for **Name**. Then collapse the panel by clicking the **Properties** icon in the top-right corner.

3. Search for pipeline in the pipeline "Activities" pane, and drag three **Execute Pipeline** activities to the pipeline canvas.

4. Select first **Execute Pipeline** activity on the canvas if it isn't already selected, and its "General" pane, to edit its details.

    - For **Name** property of the **Execute Pipeline** activity, we recommend using the name of your invoked ETL pipeline for which you want to send notifications. For example, we used **LoadDataPipeline** for the **Name** of our **Execute Pipeline** activity because it's the name of our invoked pipeline.
    - In the "Settings" pane, select an existing pipeline or create a new one using the **+ New** button for the **Invoked pipeline** property. For example, in our case, we selected **LoadDataPipeline** pipeline for the "Invoked pipeline" property. Select other options and configure any parameters for the pipeline as required to complete your configuration.

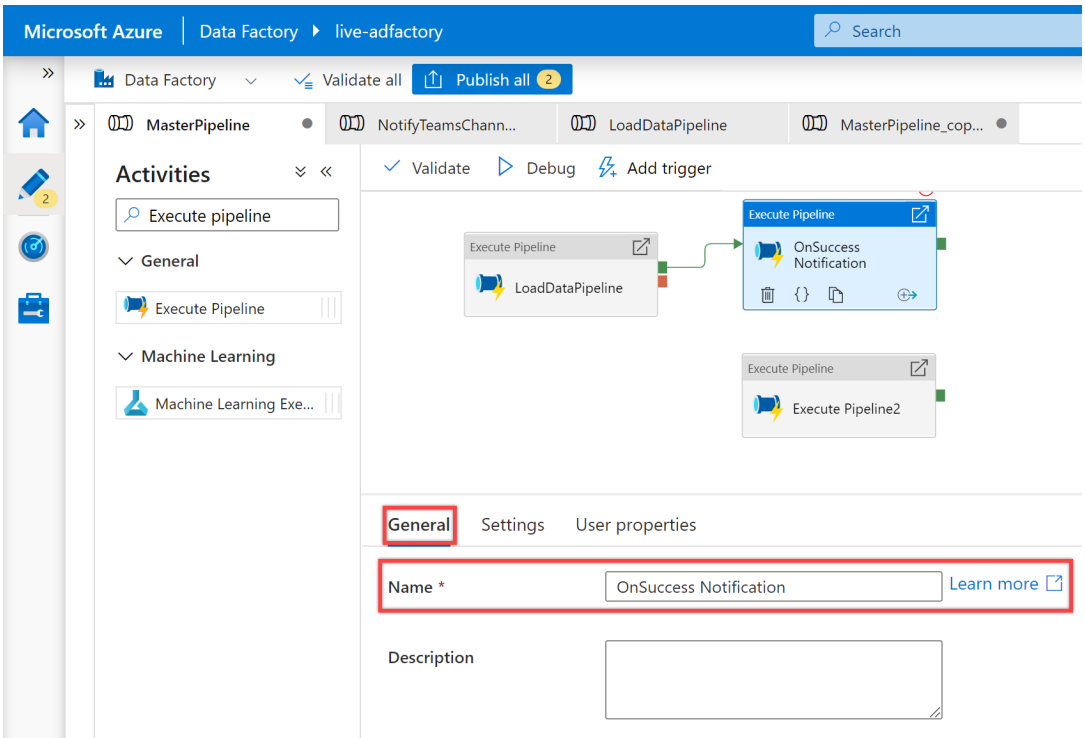


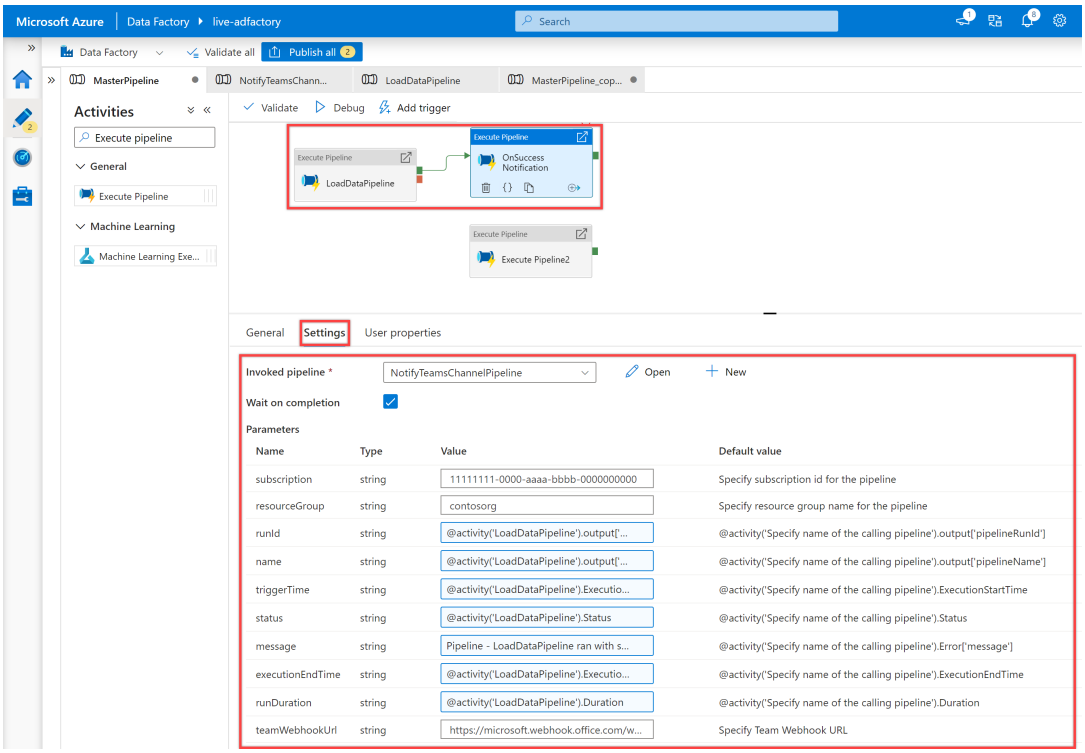


5. Select second **Execute Pipeline** activity on the canvas, and it's "General" pane, to edit its details.
   - Specify **OnSuccess Notification** for **Name** of the **Execute Pipeline** activity.
   - In the "Settings" pane, select **NotifiyTeamsChannelPipeline** pipeline, which we created earlier, for the **Invoked pipeline** property. Customize the parameters as required based on activity type. For example, I've customized the parameters as follows:

NAME	VALUE
subscription	`11111111-0000-aaaa-bbbb-0000000000`
resourceGroup	`contosorg`
runId	`@activity('LoadDataPipeline').output['pipelineRunId']`
name	`@activity('LoadDataPipeline').output['pipelineName']`
triggerTime	`@activity('LoadDataPipeline').ExecutionStartTime`
status	`@activity('LoadDataPipeline').Status`
message	`Pipeline - LoadDataPipeline ran with success.`
executionEndTime	`@activity('LoadDataPipeline').ExecutionEndTime`
runDuration	`@activity('LoadDataPipeline').Duration`
teamWebhookUrl	`https://microsoft.webhook.office.com/webhookb2/1234abcd-1x11-2ff1-a1234d0699a9e@72f988bf-32b1-41af-91ab-2d7cd011db47/IncomingWebhook/8212f66ad80040ab83cf68b554d9232a/17d52ed5c-44ed-98a0-35c12dd89a6d`

- Create a dependency condition for the second **Execute Pipeline** activity so that it only runs if the first **Execute Pipeline** activity succeeds. To create this dependency, select the green handle on the right side of the first **Execute Pipeline** activity, drag it, and connect it to the second **Execute Pipeline** activity.





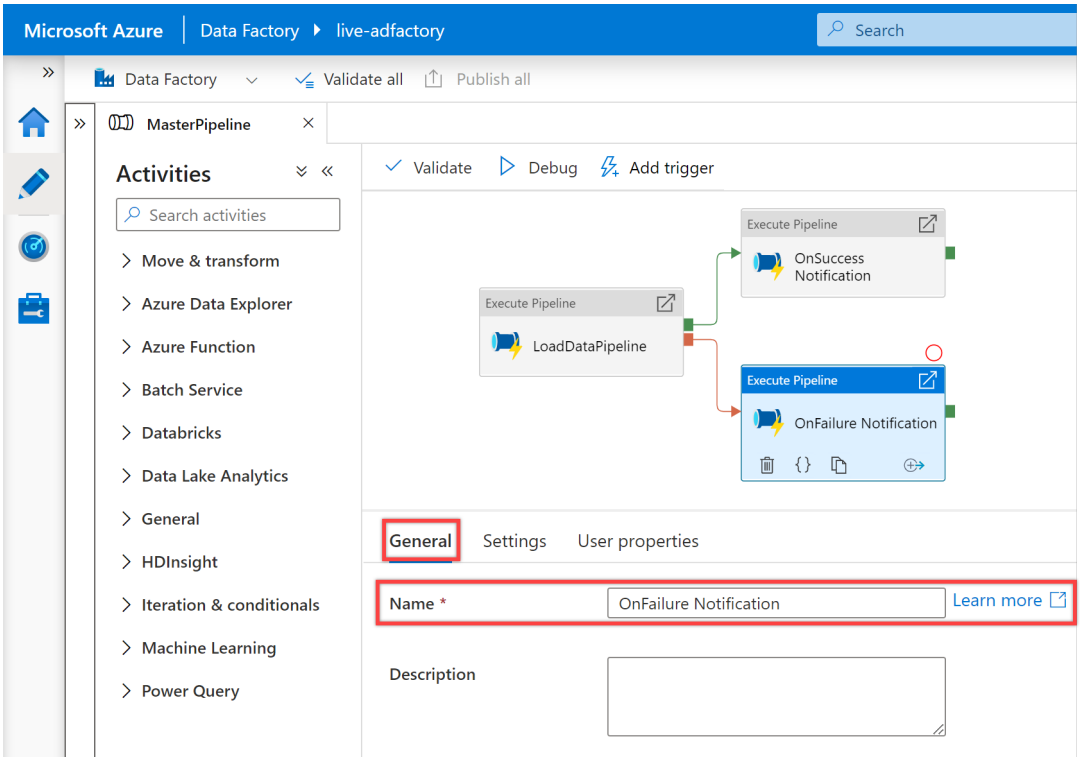
6. Select third **Execute Pipeline** activity on the canvas, and it's "General" pane, to edit its details.

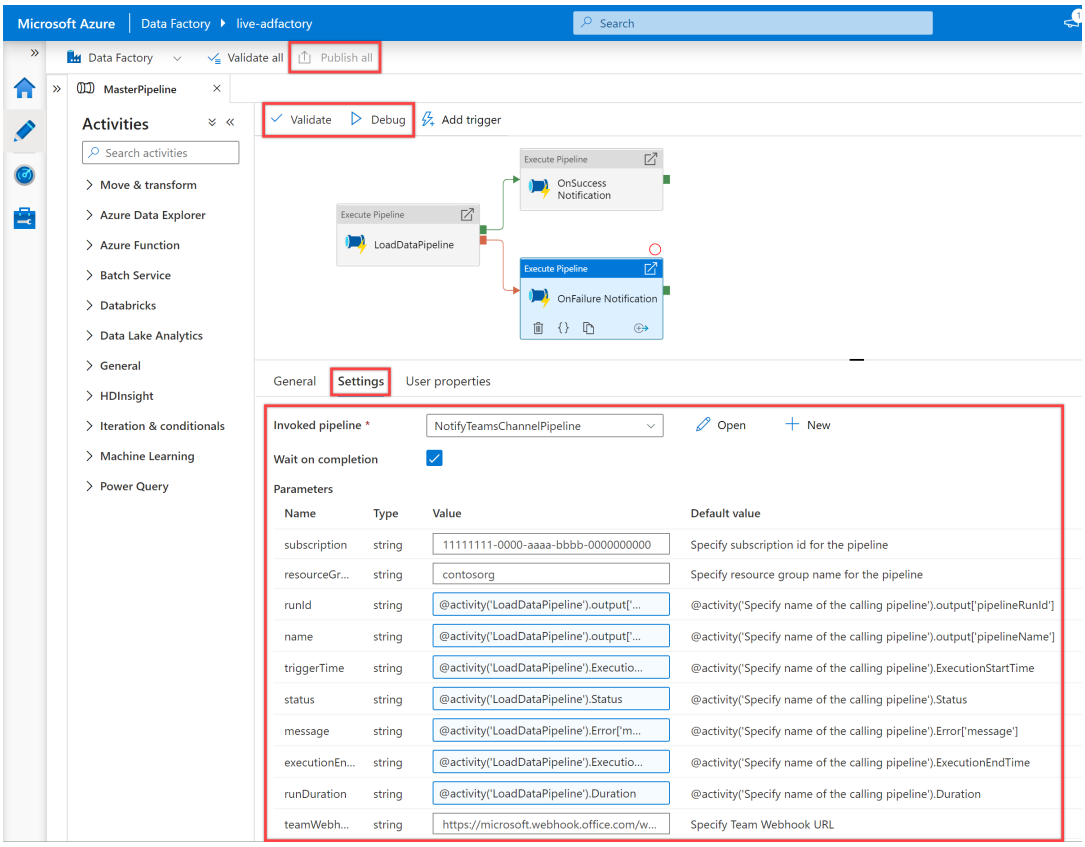
   - Specify **OnFailure Notification** for **Name** of the **Execute Pipeline** activity.
   - In the "Settings" pane, select **NotifiyTeamsChannelPipeline** pipeline for the **Invoked pipeline** property. Customize the parameters as required based on activity type. For example, I've customized the parameters this time as follows:

NAME	VALUE
subscription	`11111111-0000-aaaa-bbbb-0000000000`
resourceGroup	`contosorg`
runId	`@activity('LoadDataPipeline').output['pipelineRunId']`
name	`@activity('LoadDataPipeline').output['pipelineName']`
triggerTime	`@activity('LoadDataPipeline').ExecutionStartTime`
status	`@activity('LoadDataPipeline').Status`
message	`@activity('LoadDataPipeline').Error['message']`
executionEndTime	`@activity('LoadDataPipeline').ExecutionEndTime`
runDuration	`@activity('LoadDataPipeline').Duration`
teamWebhookUrl	`https://microsoft.webhook.office.com/webhookb2/1234abcd-1x11-2ff1-a1234d0699a9e@72f988bf-32b1-41af-91ab-2d7cd011db47/IncomingWebhook/8212f66ad80040ab83cf68b554d9232a/17d52ed5c-44ed-98a0-35c12dd89a6d`

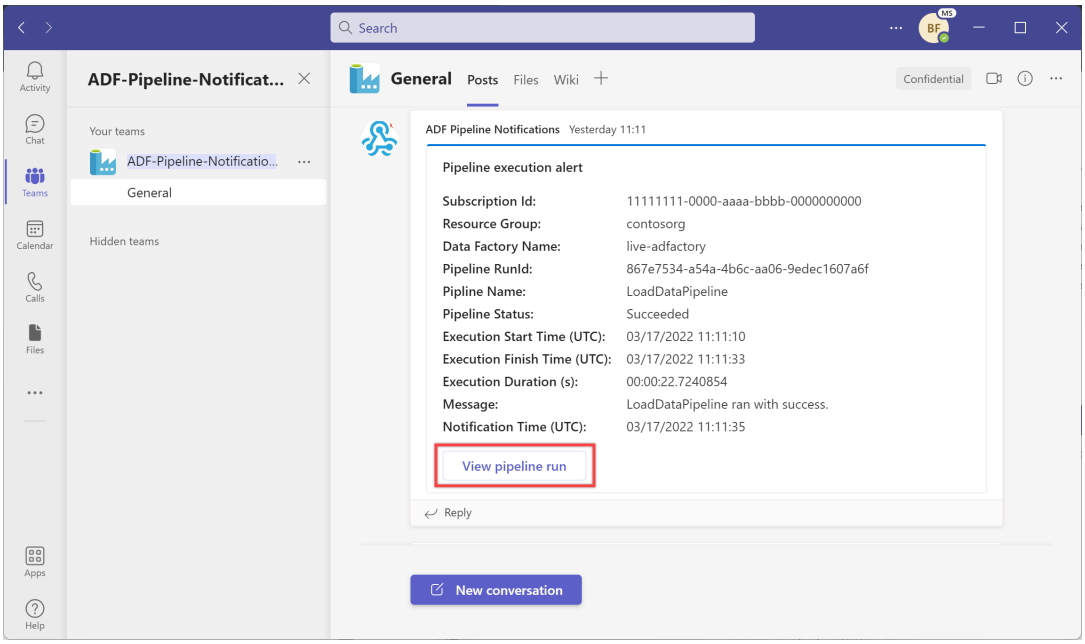
   - Create a dependency condition for the third **Execute Pipeline** activity so that it only runs if the first **Execute Pipeline** activity fails. To create this dependency, select the red handle on the right side of the first **Execute Pipeline** activity, drag it, and connect it to the third **Execute Pipeline** activity.

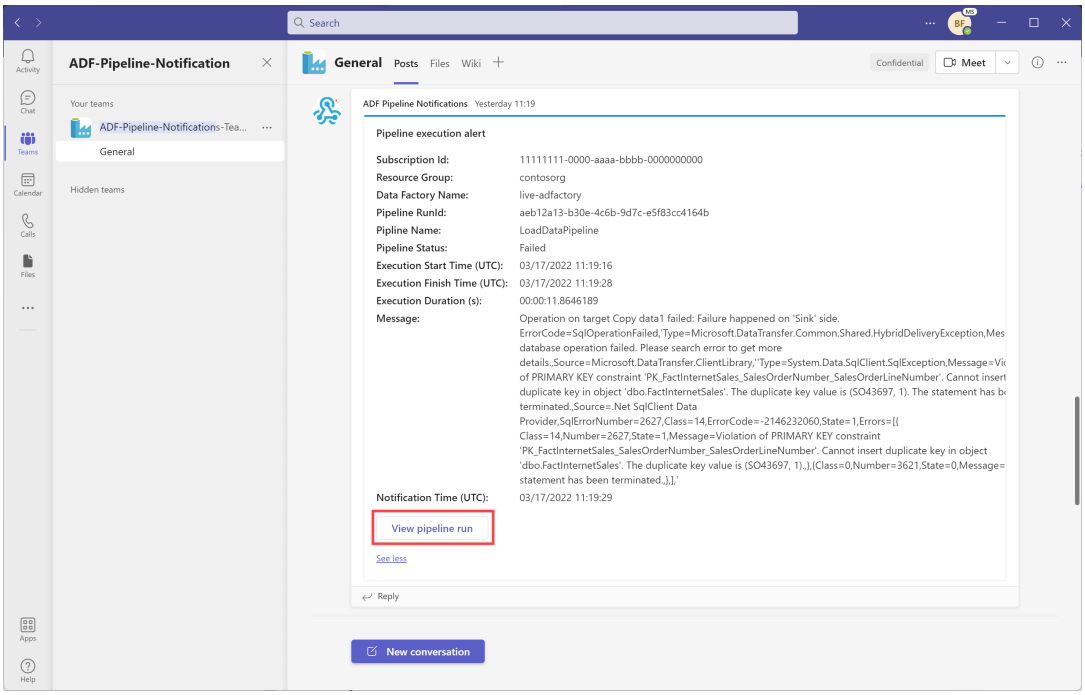
- Validate, debug, and then publish your **MasterPipeline** pipeline.





7. Run pipeline to receive notifications in Teams. For example, below are sample notifications, when my pipeline ran successfully and when it failed.





8. Select the "View pipeline run" button to view pipeline run.

Add dynamic messages with system variables and expressions

You can use system variables and expressions to make your messages dynamic. For example:

- `@activity("CopyData").output.errors[0].Message`
- `@activity("DataFlow").error.Message`

The above expressions will return the relevant error messages from a failure, which can be sent out as notification on a Teams channel. For more information about this topic, see the Copy activity output properties article.

We also encourage you to review the Microsoft Teams supported notification payload schema and further customize the above template to your needs.

Next steps

How to send email from a pipeline

How to create and configure Azure Integration Runtime

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

The Integration Runtime (IR) is the compute infrastructure used by Azure Data Factory and Synapse pipelines to provide data integration capabilities across different network environments. For more information about IR, see Integration runtime.

Azure IR provides a fully managed compute to natively perform data movement and dispatch data transformation activities to compute services like HDInsight. It is hosted in Azure environment and supports connecting to resources in public network environment with public accessible endpoints.

This document introduces how you can create and configure Azure Integration Runtime.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Default Azure IR

By default, each data factory or Synapse workspace has an Azure IR in the backend that supports operations on cloud data stores and compute services in public network. The location of that Azure IR is autoresolve. If **connectVia** property is not specified in the linked service definition, the default Azure IR is used. You only need to explicitly create an Azure IR when you would like to explicitly define the location of the IR, or if you would like to virtually group the activity executions on different IRs for management purpose.

Create Azure IR

To create and set up an Azure IR, you can use the following procedures.

Create an Azure IR via Azure PowerShell

Integration Runtime can be created using the **Set-AzDataFactoryV2IntegrationRuntime** PowerShell cmdlet. To create an Azure IR, you specify the name, location, and type to the command. Here is a sample command to create an Azure IR with location set to "West Europe":

```
Set-AzDataFactoryV2IntegrationRuntime -DataFactoryName "SampleV2DataFactory1" -Name "MySampleAzureIR" -
ResourceGroupName "ADFV2SampleRG" -Type Managed -Location "West Europe"
```

For Azure IR, the type must be set to **Managed**. You do not need to specify compute details because it is fully managed elastically in cloud. Specify compute details like node size and node count when you would like to create Azure-SSIS IR. For more information, see Create and Configure Azure-SSIS IR.

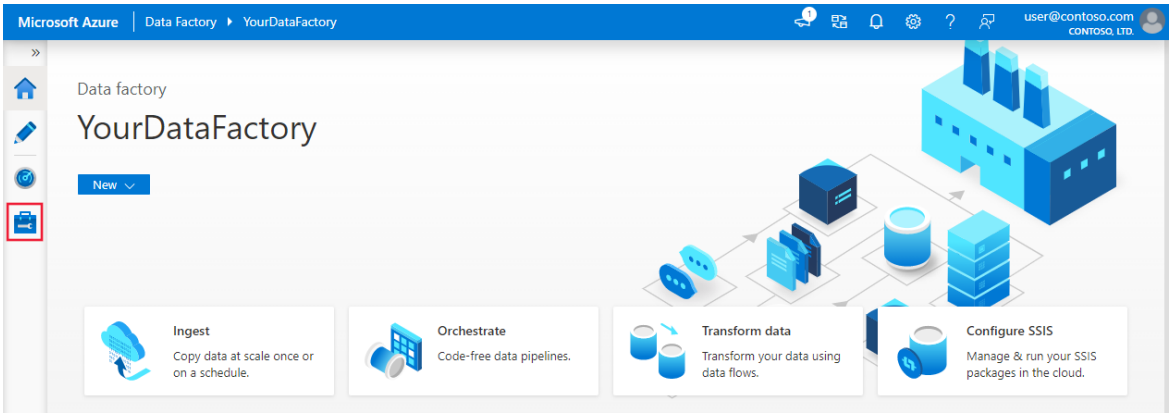
You can configure an existing Azure IR to change its location using the Set-AzDataFactoryV2IntegrationRuntime PowerShell cmdlet. For more information about the location of an Azure IR, see Introduction to integration runtime.

Create an Azure IR via UI

Use the following steps to create an Azure IR using UI.

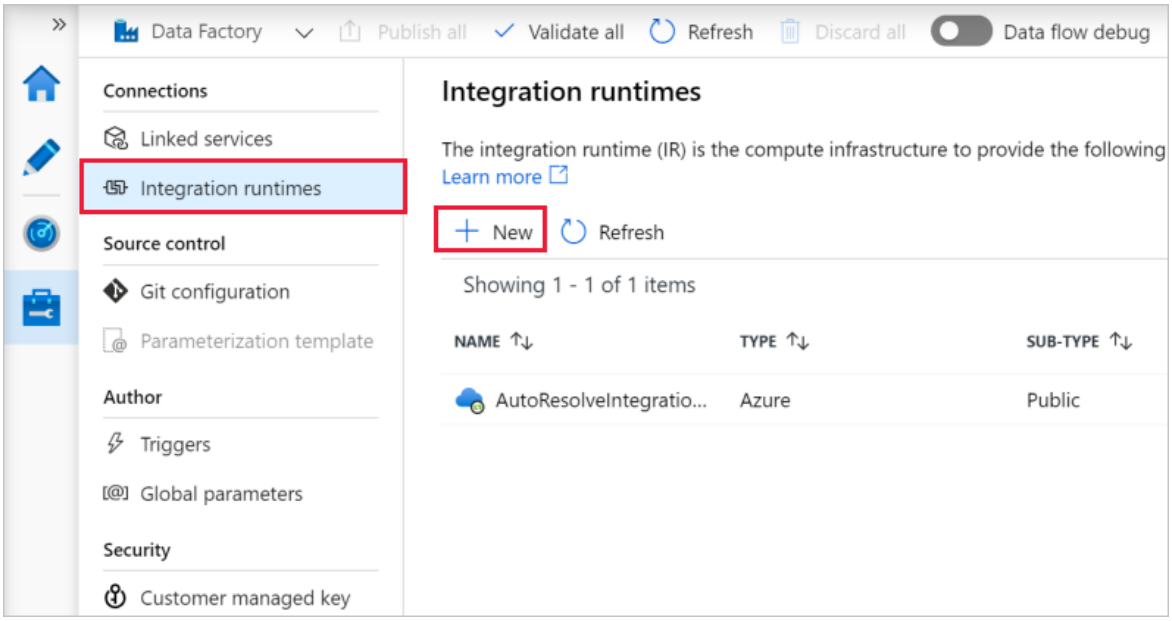
1. On the home page for the service, select the Manage tab from the leftmost pane.

   - Azure Data Factory
   - Azure Synapse



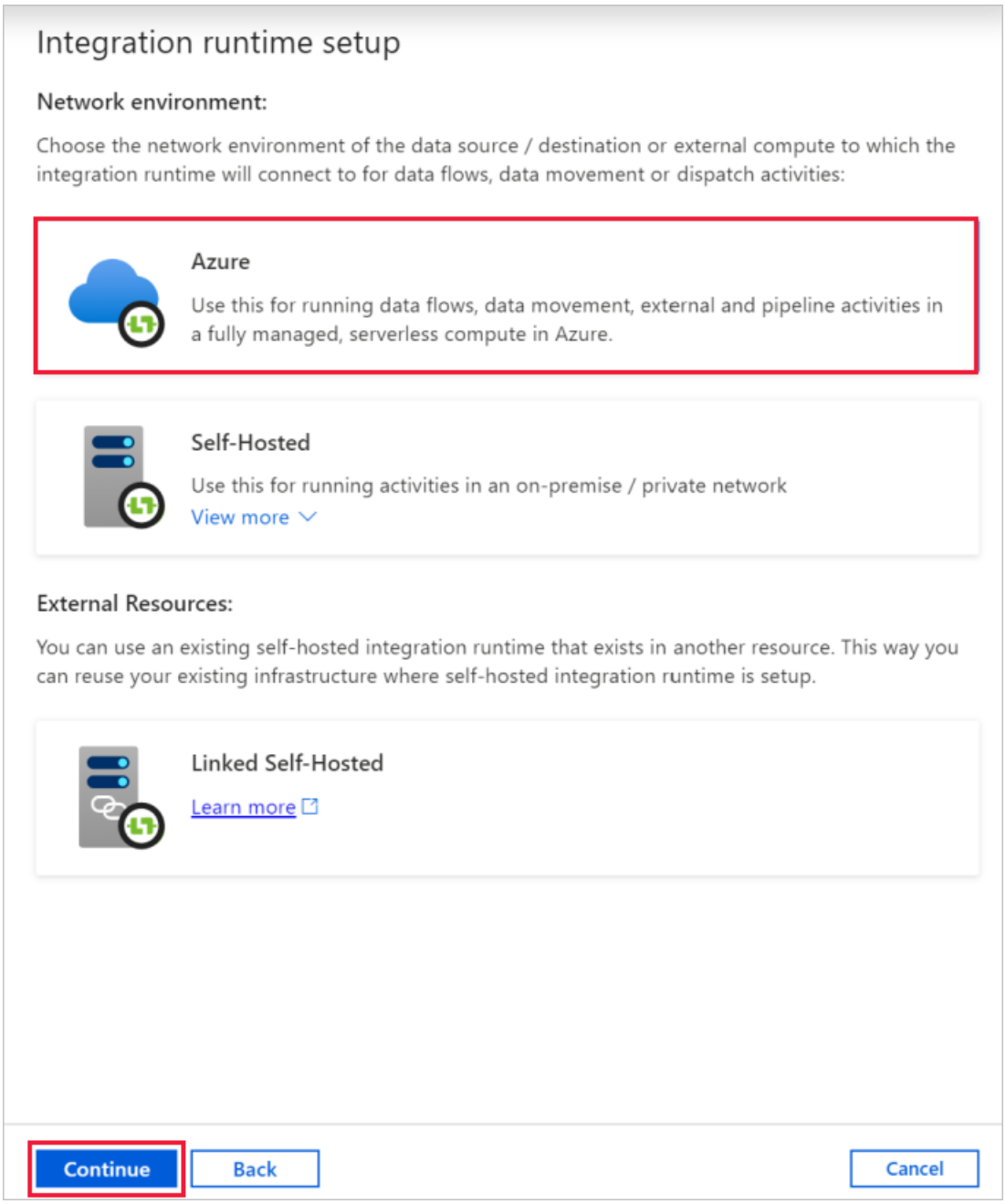
2. Select **Integration runtimes** on the left pane, and then select **+New**.

   - Azure Data Factory
   - Azure Synapse

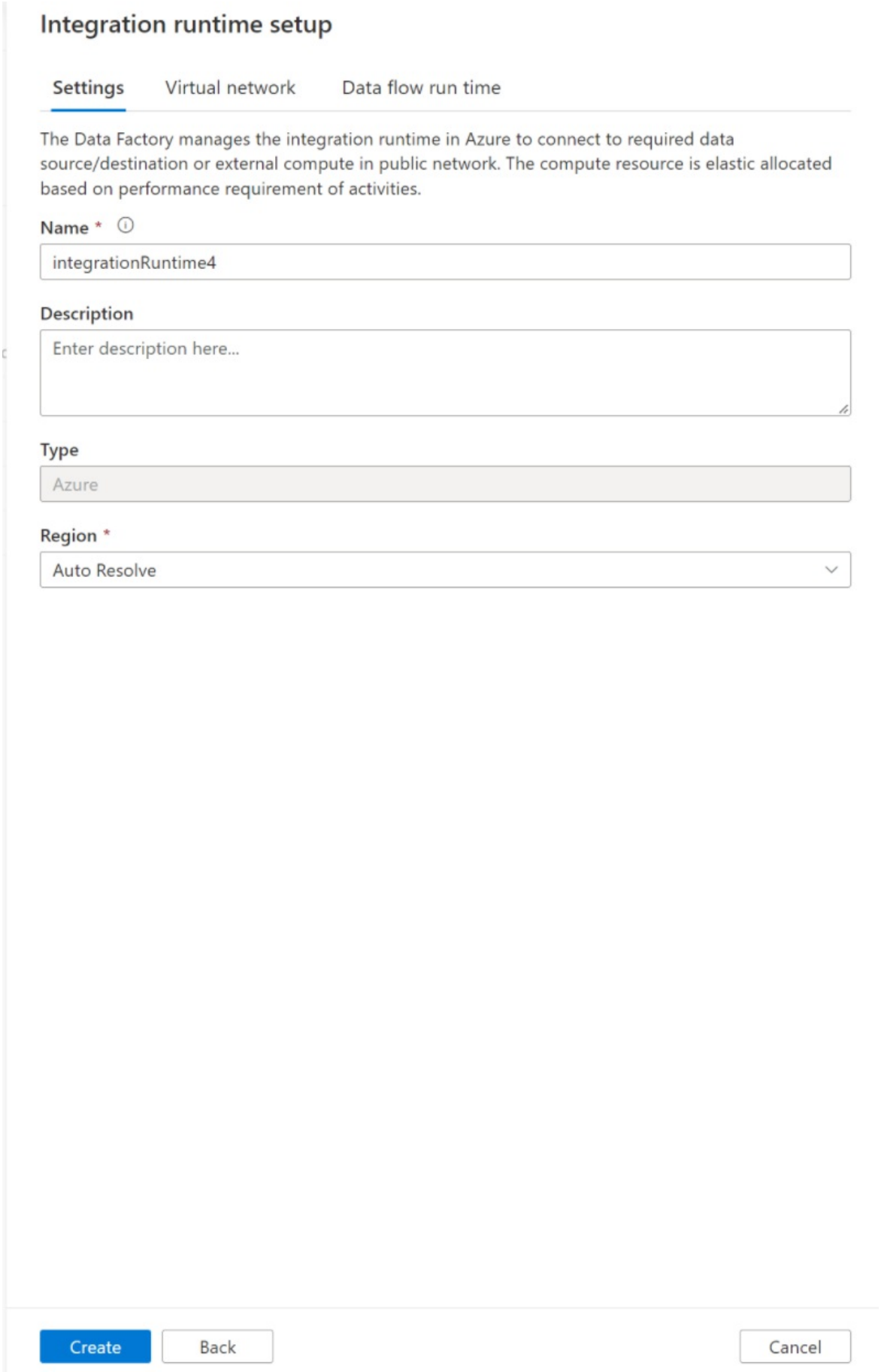


3. On the **Integration runtime setup** page, select **Azure, Self-Hosted**, and then select **Continue**.

4. On the following page, select **Azure** to create an Azure IR, and then select **Continue**.



5. Enter a name for your Azure IR, and select **Create**.



6. You'll see a pop-up notification when the creation completes. On the **Integration runtimes** page, make sure that you see the newly created IR in the list.

> **NOTE**
>
> If you want to enable managed virtual network on Azure IR, please see How to enable managed virtual network

Use Azure IR

Once an Azure IR is created, you can reference it in your Linked Service definition. Below is a sample of how you can reference the Azure Integration Runtime created above from an Azure Storage Linked Service:

```
{
    "name": "MyStorageLinkedService",
    "properties": {
      "type": "AzureStorage",
      "typeProperties": {
        "connectionString": "DefaultEndpointsProtocol=https;AccountName=myaccountname;AccountKey=..."
      },
      "connectVia": {
        "referenceName": "MySampleAzureIR",
        "type": "IntegrationRuntimeReference"
      }
    }
}
```

Next steps

See the following articles on how to create other types of integration runtimes:

- Create self-hosted integration runtime
- Create Azure-SSIS integration runtime

Create and configure a self-hosted integration runtime

8/19/2022 • 24 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

The integration runtime (IR) is the compute infrastructure that Azure Data Factory and Synapse pipelines use to provide data-integration capabilities across different network environments. For details about IR, see Integration runtime overview.

A self-hosted integration runtime can run copy activities between a cloud data store and a data store in a private network. It also can dispatch transform activities against compute resources in an on-premises network or an Azure virtual network. The installation of a self-hosted integration runtime needs an on-premises machine or a virtual machine inside a private network.

This article describes how you can create and configure a self-hosted IR.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Considerations for using a self-hosted IR

- You can use a single self-hosted integration runtime for multiple on-premises data sources. You can also share it with another data factory within the same Azure Active Directory (Azure AD) tenant. For more information, see Sharing a self-hosted integration runtime.
- You can install only one instance of a self-hosted integration runtime on any single machine. If you have two data factories or Synapse workspaces that need to access on-premises data sources, either use the self-hosted IR sharing feature to share the self-hosted IR, or install the self-hosted IR on two on-premises computers, one for each data factory or Synapse workspace.
- The self-hosted integration runtime doesn't need to be on the same machine as the data source. However, having the self-hosted integration runtime close to the data source reduces the time for the self-hosted integration runtime to connect to the data source. We recommend that you install the self-hosted integration runtime on a machine that differs from the one that hosts the on-premises data source. When the self-hosted integration runtime and data source are on different machines, the self-hosted integration runtime doesn't compete with the data source for resources.
- You can have multiple self-hosted integration runtimes on different machines that connect to the same on-premises data source. For example, if you have two self-hosted integration runtimes that serve two data factories, the same on-premises data source can be registered with both data factories.
- Use a self-hosted integration runtime to support data integration within an Azure virtual network.
- Treat your data source as an on-premises data source that is behind a firewall, even when you use Azure ExpressRoute. Use the self-hosted integration runtime to connect the service to the data source.
- Use the self-hosted integration runtime even if the data store is in the cloud on an Azure Infrastructure as a Service (IaaS) virtual machine.
- Tasks might fail in a self-hosted integration runtime that you installed on a Windows server for which FIPS-compliant encryption is enabled. To work around this problem, you have two options: store credentials/secret

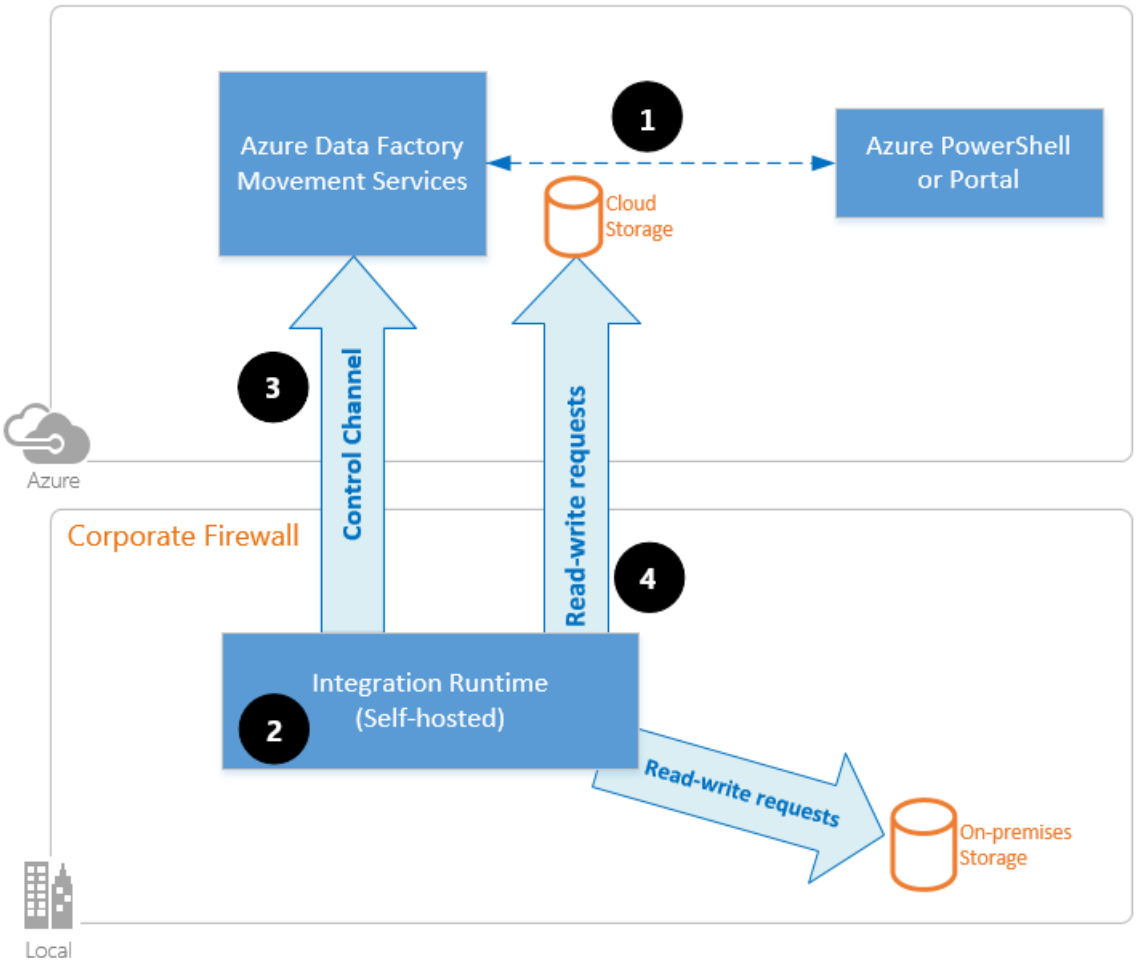
values in an Azure Key Vault or disable FIPS-compliant encryption on the server. To disable FIPS-compliant encryption, change the following registry subkey's value from 1 (enabled) to 0 (disabled): `HKLM\System\CurrentControlSet\Control\Lsa\FIPSAlgorithmPolicy\Enabled`. If you use the self-hosted integration runtime as a proxy for SSIS integration runtime, FIPS-compliant encryption can be enabled and will be used when moving data from on premises to Azure Blob Storage as a staging area.

> **NOTE**
> Currently self-hosted integration runtime can only be shared with multiple data factories, it can't be shared across Synapse workspaces or between data factory and Synapse workspace.

Command flow and data flow

When you move data between on-premises and the cloud, the activity uses a self-hosted integration runtime to transfer the data between an on-premises data source and the cloud.

Here is a high-level summary of the data-flow steps for copying with a self-hosted IR:



1. A data developer first creates a self-hosted integration runtime within an Azure data factory or Synapse workspace by using the Azure portal or the PowerShell cmdlet. Then the data developer creates a linked service for an on-premises data store, specifying the self-hosted integration runtime instance that the service should use to connect to data stores.

2. The self-hosted integration runtime node encrypts the credentials by using Windows Data Protection Application Programming Interface (DPAPI) and saves the credentials locally. If multiple nodes are set for high availability, the credentials are further synchronized across other nodes. Each node encrypts the credentials by using DPAPI and stores them locally. Credential synchronization is transparent to the data

developer and is handled by the self-hosted IR.

3. Azure Data Factory and Synapse pipelines communicate with the self-hosted integration runtime to schedule and manage jobs. Communication is via a control channel that uses a shared Azure Relay connection. When an activity job needs to be run, the service queues the request along with any credential information. It does so in case credentials aren't already stored on the self-hosted integration runtime. The self-hosted integration runtime starts the job after it polls the queue.

4. The self-hosted integration runtime copies data between an on-premises store and cloud storage. The direction of the copy depends on how the copy activity is configured in the data pipeline. For this step, the self-hosted integration runtime directly communicates with cloud-based storage services like Azure Blob storage over a secure HTTPS channel.

Prerequisites

- The supported versions of Windows are:
  - Windows 8.1
  - Windows 10
  - Windows 11
  - Windows Server 2012
  - Windows Server 2012 R2
  - Windows Server 2016
  - Windows Server 2019
  - Windows Server 2022

Installation of the self-hosted integration runtime on a domain controller isn't supported.

- Self-hosted integration runtime requires a 64-bit Operating System with .NET Framework 4.7.2 or above. See .NET Framework System Requirements for details.
- The recommended minimum configuration for the self-hosted integration runtime machine is a 2-GHz processor with 4 cores, 8 GB of RAM, and 80 GB of available hard drive space. For the details of system requirements, see Download.
- If the host machine hibernates, the self-hosted integration runtime doesn't respond to data requests. Configure an appropriate power plan on the computer before you install the self-hosted integration runtime. If the machine is configured to hibernate, the self-hosted integration runtime installer prompts with a message.
- You must be an administrator on the machine to successfully install and configure the self-hosted integration runtime.
- Copy-activity runs happen with a specific frequency. Processor and RAM usage on the machine follows the same pattern with peak and idle times. Resource usage also depends heavily on the amount of data that is moved. When multiple copy jobs are in progress, you see resource usage go up during peak times.
- Tasks might fail during extraction of data in Parquet, ORC, or Avro formats. For more on Parquet, see Parquet format in Azure Data Factory. File creation runs on the self-hosted integration machine. To work as expected, file creation requires the following prerequisites:
  - Visual C++ 2010 Redistributable Package (x64)
  - Java Runtime (JRE) version 8 from a JRE provider such as Adopt OpenJDK. Ensure that the JAVA_HOME environment variable is set to the JDK folder (and not just the JRE folder).

> **NOTE**
>
> It might be necessary to adjust the Java settings if memory errors occur, as described in the Parquet format documentation.

> **NOTE**
>
> If you are running in government cloud, please review Connect to government cloud.

Setting up a self-hosted integration runtime

To create and set up a self-hosted integration runtime, use the following procedures.

Create a self-hosted IR via Azure PowerShell

1. You can use Azure PowerShell for this task. Here is an example:

```
Set-AzDataFactoryV2IntegrationRuntime -ResourceGroupName $resourceGroupName -DataFactoryName
$dataFactoryName -Name $selfHostedIntegrationRuntimeName -Type SelfHosted -Description "selfhosted IR
description"
```

2. Download and install the self-hosted integration runtime on a local machine.

3. Retrieve the authentication key and register the self-hosted integration runtime with the key. Here is a PowerShell example:

```
Get-AzDataFactoryV2IntegrationRuntimeKey -ResourceGroupName $resourceGroupName -DataFactoryName
$dataFactoryName -Name $selfHostedIntegrationRuntimeName
```

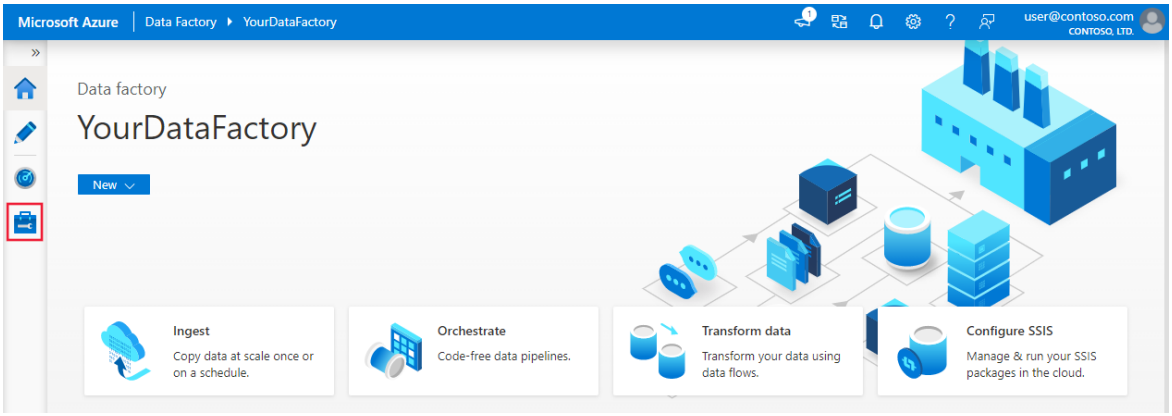
> **NOTE**
>
> Run PowerShell command in Azure government, please see Connect to Azure Government with PowerShell.

Create a self-hosted IR via UI

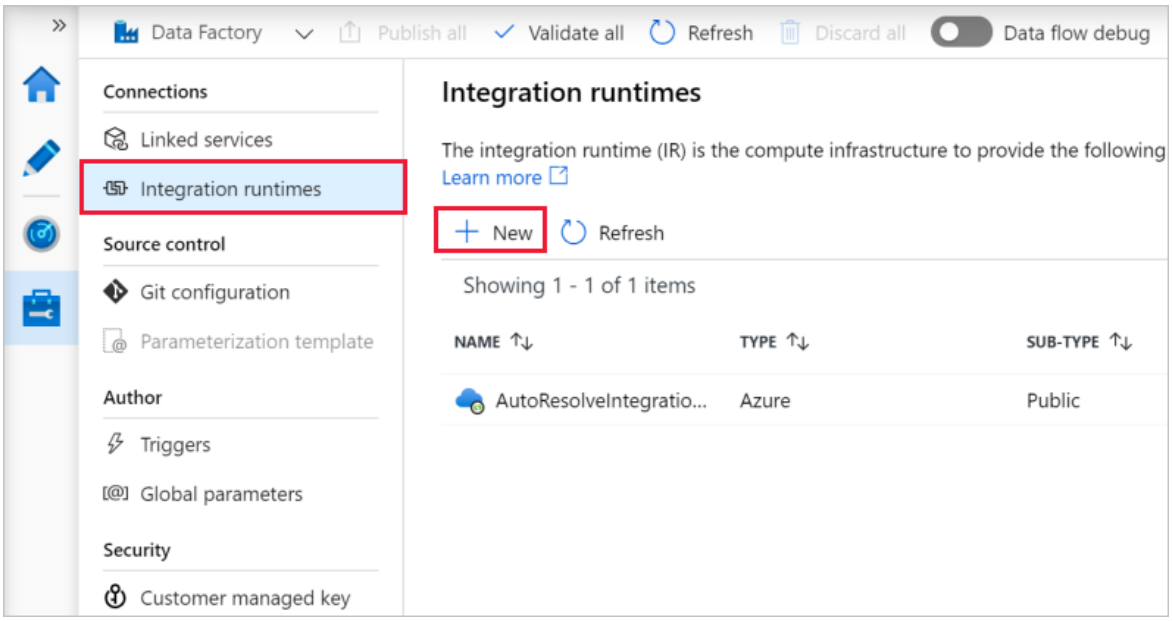
Use the following steps to create a self-hosted IR using the Azure Data Factory or Azure Synapse UI.

- Azure Data Factory
- Azure Synapse

1. On the home page of the Azure Data Factory UI, select the Manage tab from the leftmost pane.

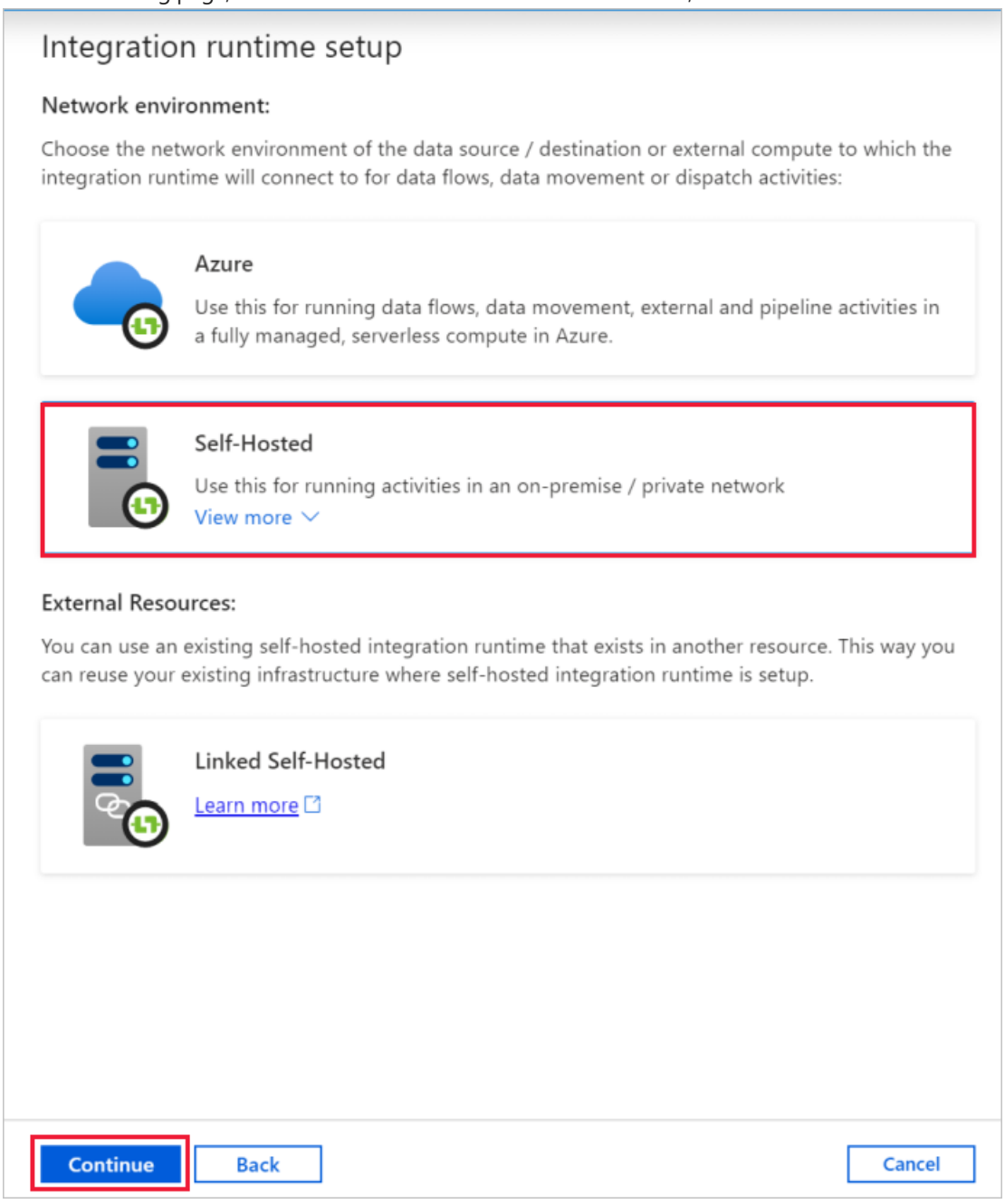


2. Select **Integration runtimes** on the left pane, and then select **+New**.



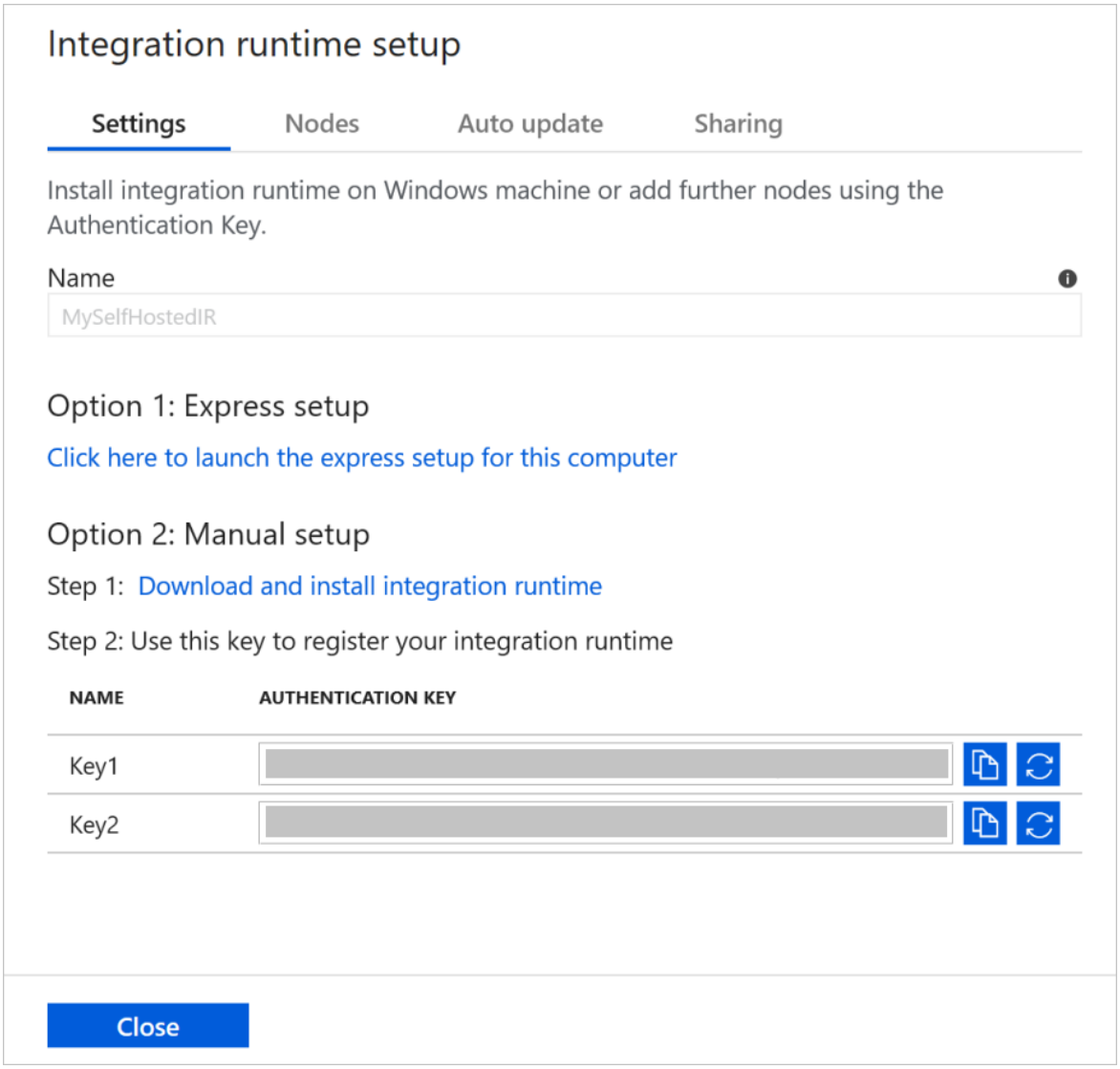
3. On the **Integration runtime setup** page, select **Azure, Self-Hosted**, and then select **Continue**.

4. On the following page, select **Self-Hosted** to create a Self-Hosted IR, and then select **Continue**.



Configure a self-hosted IR via UI

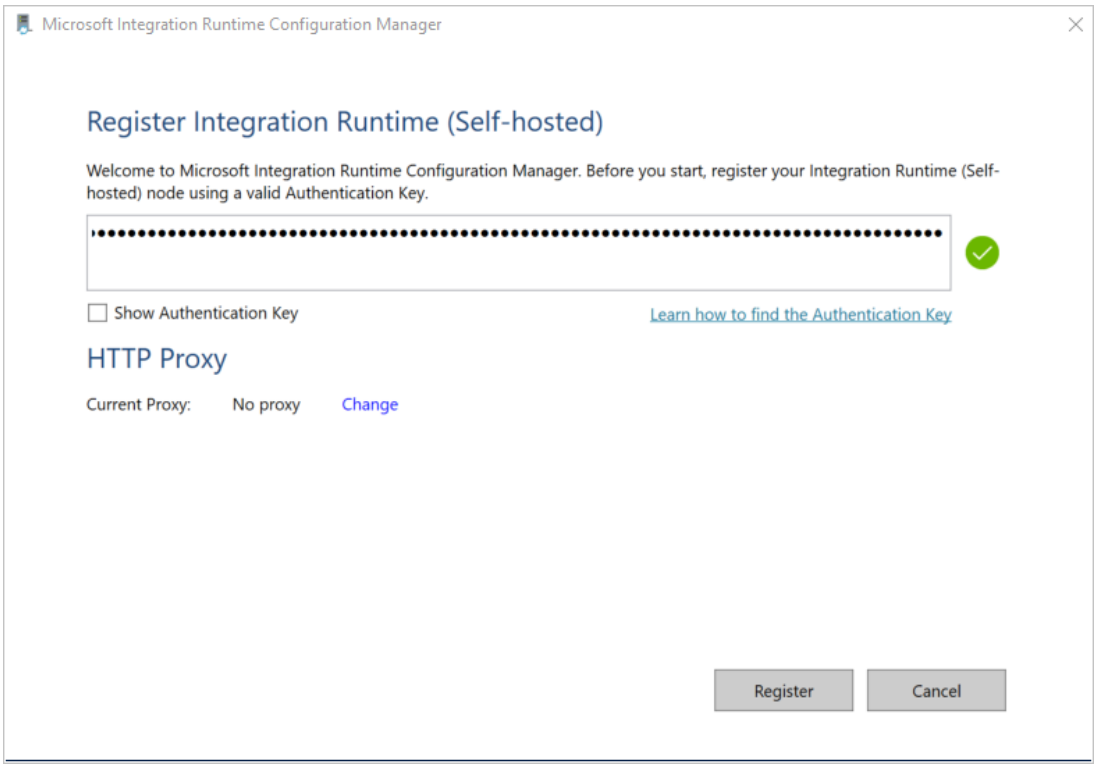
1. Enter a name for your IR, and select **Create**.
2. On the **Integration runtime setup** page, select the link under **Option 1** to open the express setup on your computer. Or follow the steps under **Option 2** to set up manually. The following instructions are based on manual setup:



   a. Copy and paste the authentication key. Select **Download and install integration runtime**.

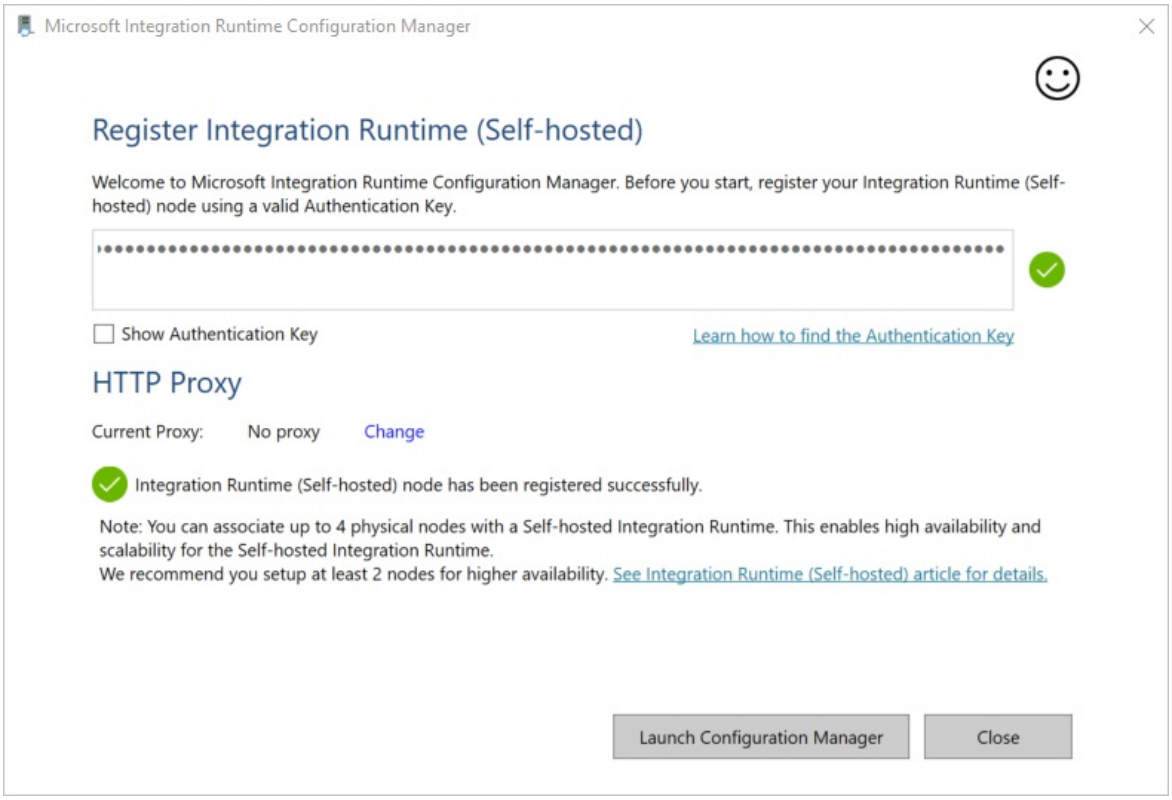
   b. Download the self-hosted integration runtime on a local Windows machine. Run the installer.

   c. On the **Register Integration Runtime (Self-hosted)** page, paste the key you saved earlier, and select **Register**.



d. On the **New Integration Runtime (Self-hosted) Node** page, select **Finish**.

3. After the self-hosted integration runtime is registered successfully, you see the following window:



Set up a self-hosted IR on an Azure VM via an Azure Resource Manager template

You can automate self-hosted IR setup on an Azure virtual machine by using the Create self host IR template. The template provides an easy way to have a fully functional self-hosted IR inside an Azure virtual network. The IR has high-availability and scalability features, as long as you set the node count to 2 or higher.

Set up an existing self-hosted IR via local PowerShell

You can use a command line to set up or manage an existing self-hosted IR. This usage can especially help to automate the installation and registration of self-hosted IR nodes.

Dmgcmd.exe is included in the self-hosted installer. It's typically located in the C:\Program Files\Microsoft Integration Runtime\4.0\Shared\ folder. This application supports various parameters and can be invoked via a command line using batch scripts for automation.

Use the application as follows:

```
dmgcmd ACTION args...
```

Here are details of the application's actions and arguments:

ACTION	ARGS	DESCRIPTION
`-rn`, `-RegisterNewNode`	"`<AuthenticationKey>`" ["`<NodeName>`"]	Register a self-hosted integration runtime node with the specified authentication key and node name.
`-era`, `-EnableRemoteAccess`	"`<port>`" ["`<thumbprint>`"]	Enable remote access on the current node to set up a high-availability cluster. Or enable setting credentials directly against the self-hosted IR without going through an Azure Data Factory or Azure Synapse workspace. You do the latter by using the **New-AzDataFactoryV2LinkedServiceEncryptedCredential** cmdlet from a remote machine in the same network.
`-erac`, `-EnableRemoteAccessInContainer`	"`<port>`" ["`<thumbprint>`"]	Enable remote access to the current node when the node runs in a container.
`-dra`, `-DisableRemoteAccess`		Disable remote access to the current node. Remote access is needed for multinode setup. The **New-AzDataFactoryV2LinkedServiceEncryptedCredential** PowerShell cmdlet still works even when remote access is disabled. This behavior is true as long as the cmdlet is executed on the same machine as the self-hosted IR node.
`-k`, `-Key`	"`<AuthenticationKey>`"	Overwrite or update the previous authentication key. Be careful with this action. Your previous self-hosted IR node can go offline if the key is of a new integration runtime.
`-gbf`, `-GenerateBackupFile`	"`<filePath>`" "`<password>`"	Generate a backup file for the current node. The backup file includes the node key and data-store credentials.
`-ibf`, `-ImportBackupFile`	"`<filePath>`" "`<password>`"	Restore the node from a backup file.
`-r`, `-Restart`		Restart the self-hosted integration runtime host service.

ACTION	ARGS	DESCRIPTION
`-s`, `-Start`		Start the self-hosted integration runtime host service.
`-t`, `-Stop`		Stop the self-hosted integration runtime host service.
`-sus`, `-StartUpgradeService`		Start the self-hosted integration runtime upgrade service.
`-tus`, `-StopUpgradeService`		Stop the self-hosted integration runtime upgrade service.
`-tonau`, `-TurnOnAutoUpdate`		Turn on the self-hosted integration runtime auto-update.
`-toffau`, `-TurnOffAutoUpdate`		Turn off the self-hosted integration runtime auto-update.
`-ssa`, `-SwitchServiceAccount`	" `<domain\user>` " [" `<password>` "]	Set DIAHostService to run as a new account. Use the empty password "" for system accounts and virtual accounts.
`-elma`, `-EnableLocalMachineAccess`		Enable local machine access (localhost, private IP) on the current self-hosted IR node. In self-hosted IR High Availability scenario, the action needs to be invoked on every self-hosted IR node.
`-dlma`, `-DisableLocalMachineAccess`		Disable local machine access (localhost, private IP) on the current self-hosted IR node. In self-hosted IR High Availability scenario, the action needs to be invoked on every self-hosted IR node.

Install and register a self-hosted IR from Microsoft Download Center

1. Go to the Microsoft integration runtime download page.
2. Select **Download**, select the 64-bit version, and select **Next**. The 32-bit version isn't supported.
3. Run the MSI file directly, or save it to your hard drive and run it.
4. On the **Welcome** window, select a language and select **Next**.
5. Accept the Microsoft Software License Terms and select **Next**.
6. Select **folder** to install the self-hosted integration runtime, and select **Next**.
7. On the **Ready to install** page, select **Install**.
8. Select **Finish** to complete installation.
9. Get the authentication key by using PowerShell. Here's a PowerShell example for retrieving the

authentication key:

```
Get-AzDataFactoryV2IntegrationRuntimeKey -ResourceGroupName $resourceGroupName -DataFactoryName
$dataFactoryName -Name $selfHostedIntegrationRuntimeName
```

10. On the **Register Integration Runtime (Self-hosted)** window of Microsoft Integration Runtime Configuration Manager running on your machine, take the following steps:

    a. Paste the authentication key in the text area.

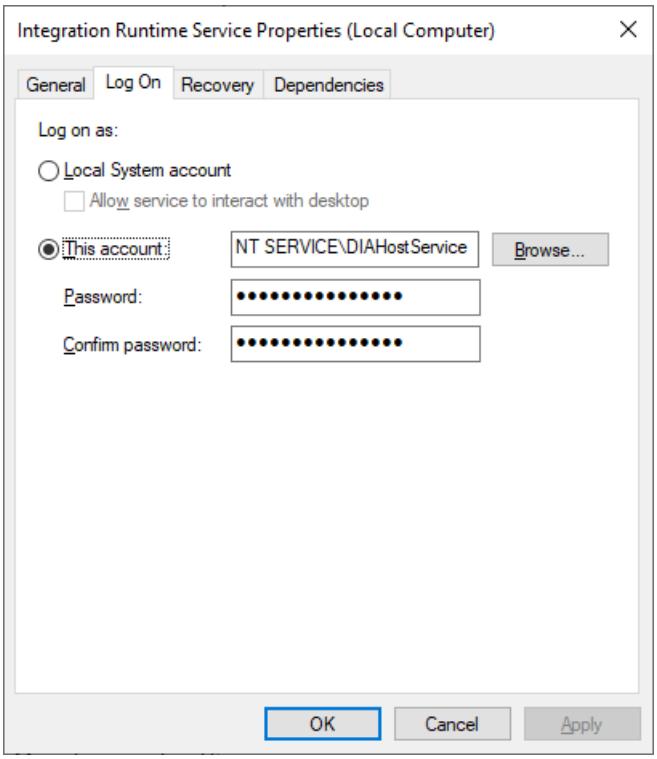
    b. Optionally, select **Show authentication key** to see the key text.

    c. Select **Register**.

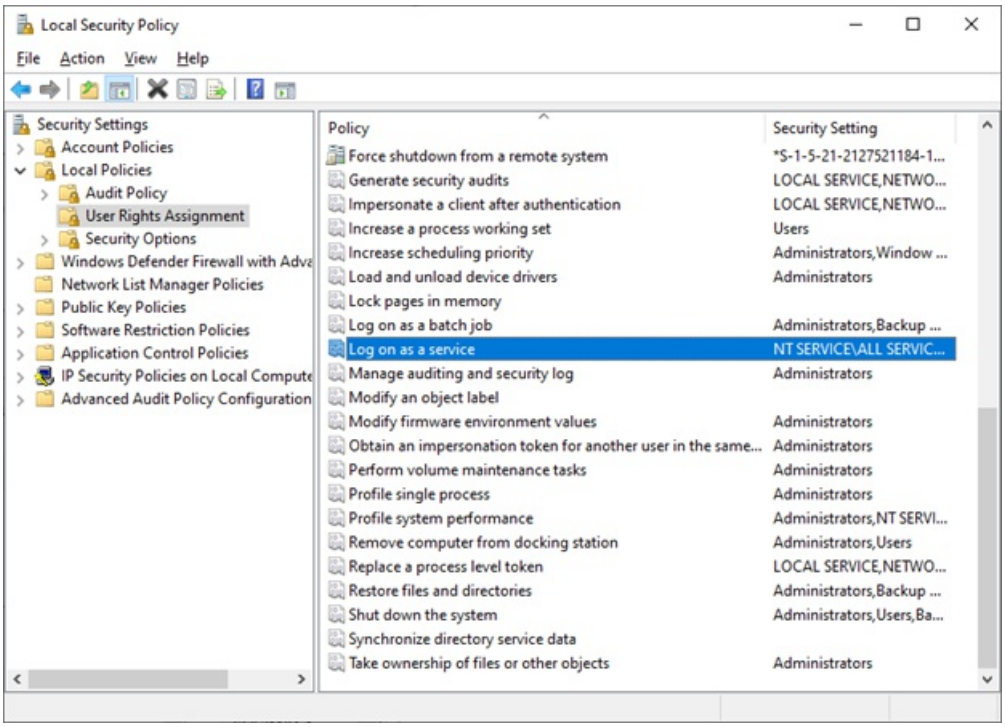
> **NOTE**
> Release Notes are available on the same Microsoft integration runtime download page.

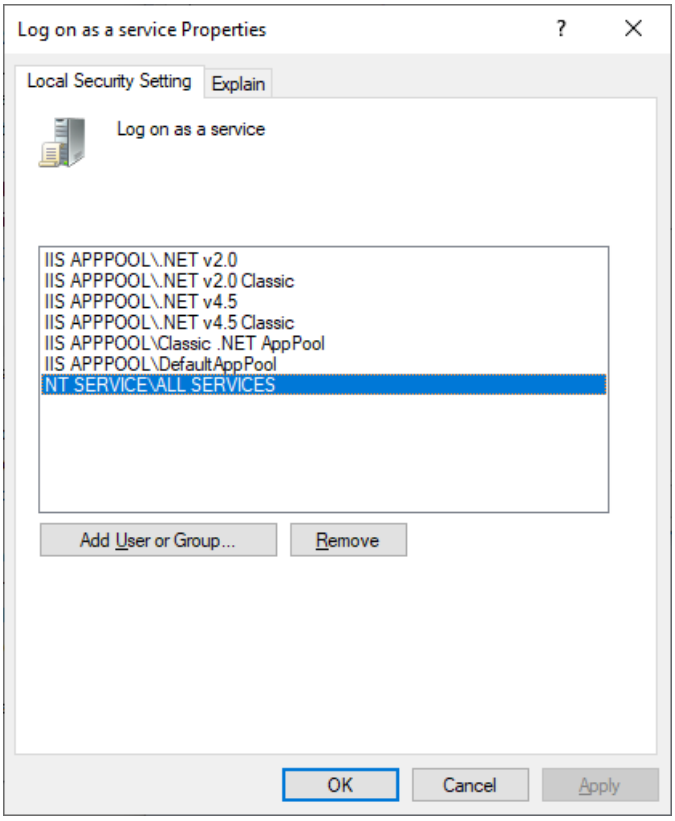
Service account for Self-hosted integration runtime

The default log on service account of Self-hosted integration runtime is **NT SERVICE\DIAHostService**. You can see it in **Services -> Integration Runtime Service -> Properties -> Log on**.



Make sure the account has the permission of Log on as a service. Otherwise self-hosted integration runtime can't start successfully. You can check the permission in **Local Security Policy -> Security Settings -> Local Policies -> User Rights Assignment -> Log on as a service**





Notification area icons and notifications

If you move your cursor over the icon or message in the notification area, you can see details about the state of the self-hosted integration runtime.

High availability and scalability

You can associate a self-hosted integration runtime with multiple on-premises machines or virtual machines in

Azure. These machines are called nodes. You can have up to four nodes associated with a self-hosted integration runtime. The benefits of having multiple nodes on on-premises machines that have a gateway installed for a logical gateway are:

- Higher availability of the self-hosted integration runtime so that it's no longer the single point of failure in your big data solution or cloud data integration. This availability helps ensure continuity when you use up to four nodes.
- Improved performance and throughput during data movement between on-premises and cloud data stores. Get more information on performance comparisons.

You can associate multiple nodes by installing the self-hosted integration runtime software from Download Center. Then, register it by using either of the authentication keys that were obtained from the **New-AzDataFactoryV2IntegrationRuntimeKey** cmdlet, as described in the tutorial.

> **NOTE**
> You don't need to create a new self-hosted integration runtime to associate each node. You can install the self-hosted integration runtime on another machine and register it by using the same authentication key.

> **NOTE**
> Before you add another node for high availability and scalability, ensure that the **Remote access to intranet** option is enabled on the first node. To do so, select **Microsoft Integration Runtime Configuration Manager** > **Settings** > **Remote access to intranet**.

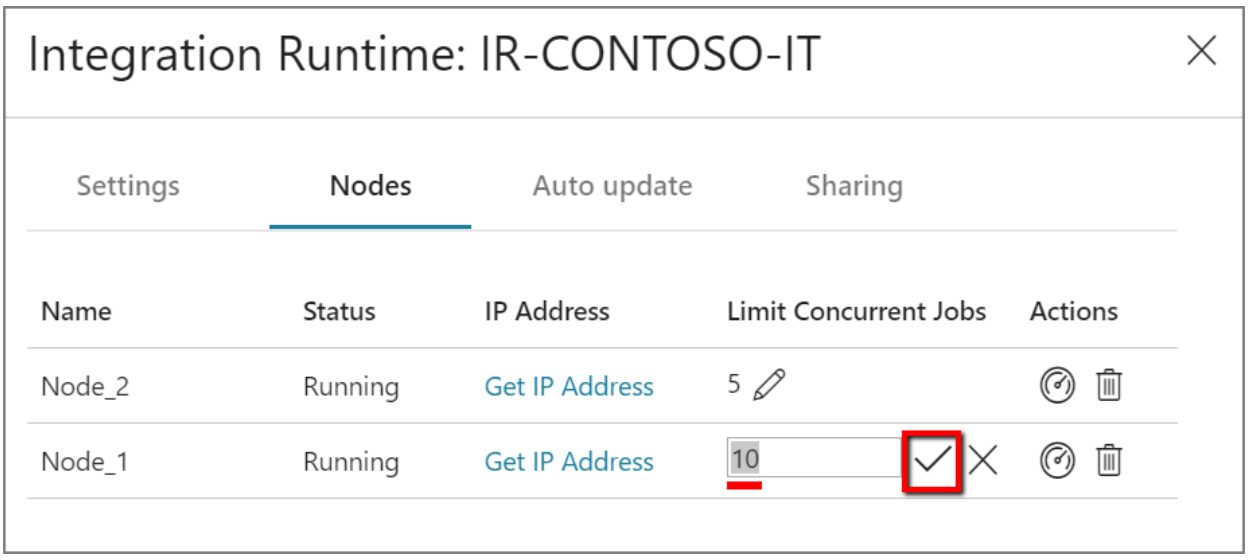
Scale considerations

Scale out

When processor usage is high and available memory is low on the self-hosted IR, add a new node to help scale out the load across machines. If activities fail because they time out or the self-hosted IR node is offline, it helps if you add a node to the gateway.

Scale up

When the processor and available RAM aren't well utilized, but the execution of concurrent jobs reaches a node's limits, scale up by increasing the number of concurrent jobs that a node can run. You might also want to scale up when activities time out because the self-hosted IR is overloaded. As shown in the following image, you can increase the maximum capacity for a node:



TLS/SSL certificate requirements

If you want to enable remote access from intranet with TLS/SSL certificate (Advanced) to secure communication between integration runtime nodes, you can follow steps in Enable remote access from intranet with TLS/SSL certificate.

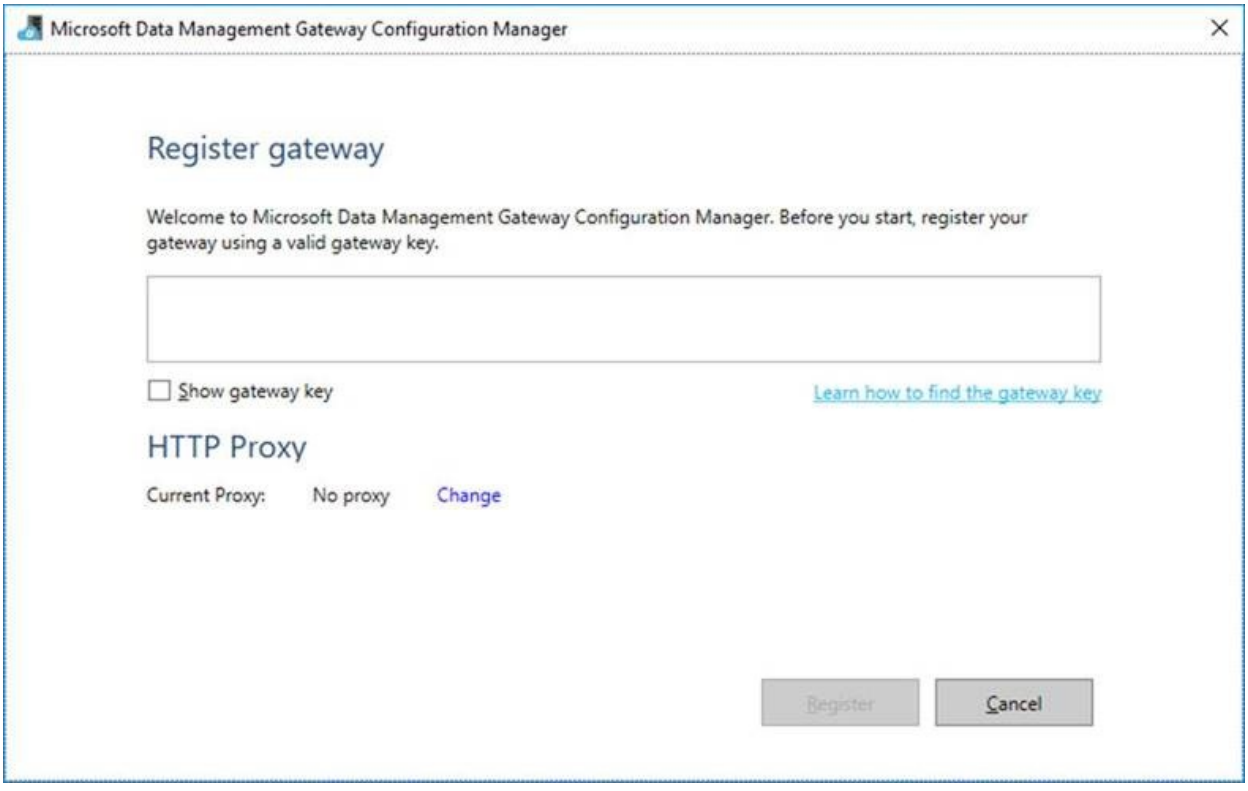
> **NOTE**
>
> This certificate is used:
>
> - To encrypt ports on a self-hosted IR node.
> - For node-to-node communication for state synchronization, which includes credentials synchronization of linked services across nodes.
> - When a PowerShell cmdlet is used for linked-service credential settings from within a local network.
>
> We suggest you use this certificate if your private network environment is not secure or if you want to secure the communication between nodes within your private network.
>
> Data movement in transit from a self-hosted IR to other data stores always happens within an encrypted channel, regardless of whether or not this certificate is set.

Credential Sync

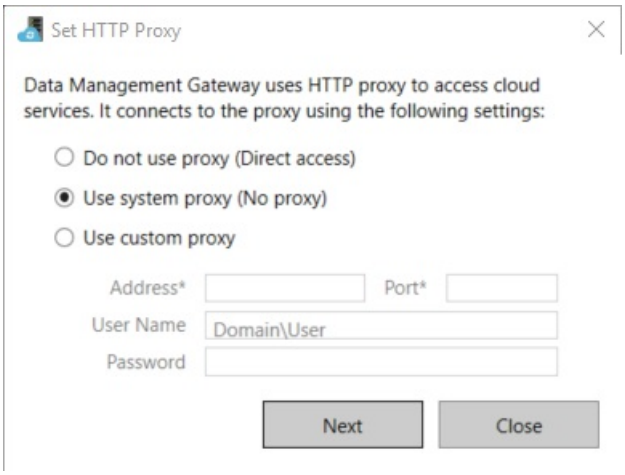
If you don't store credentials or secret values in an Azure Key Vault, the credentials or secret values will be stored in the machines where your self-hosted integration runtime locates. Each node will have a copy of credential with certain version. In order to make all nodes work together, the version number should be the same for all nodes.

Proxy server considerations

If your corporate network environment uses a proxy server to access the internet, configure the self-hosted integration runtime to use appropriate proxy settings. You can set the proxy during the initial registration phase.



When configured, the self-hosted integration runtime uses the proxy server to connect to the cloud service's source and destination (which use the HTTP or HTTPS protocol). This is why you select **Change link** during initial setup.



There are three configuration options:

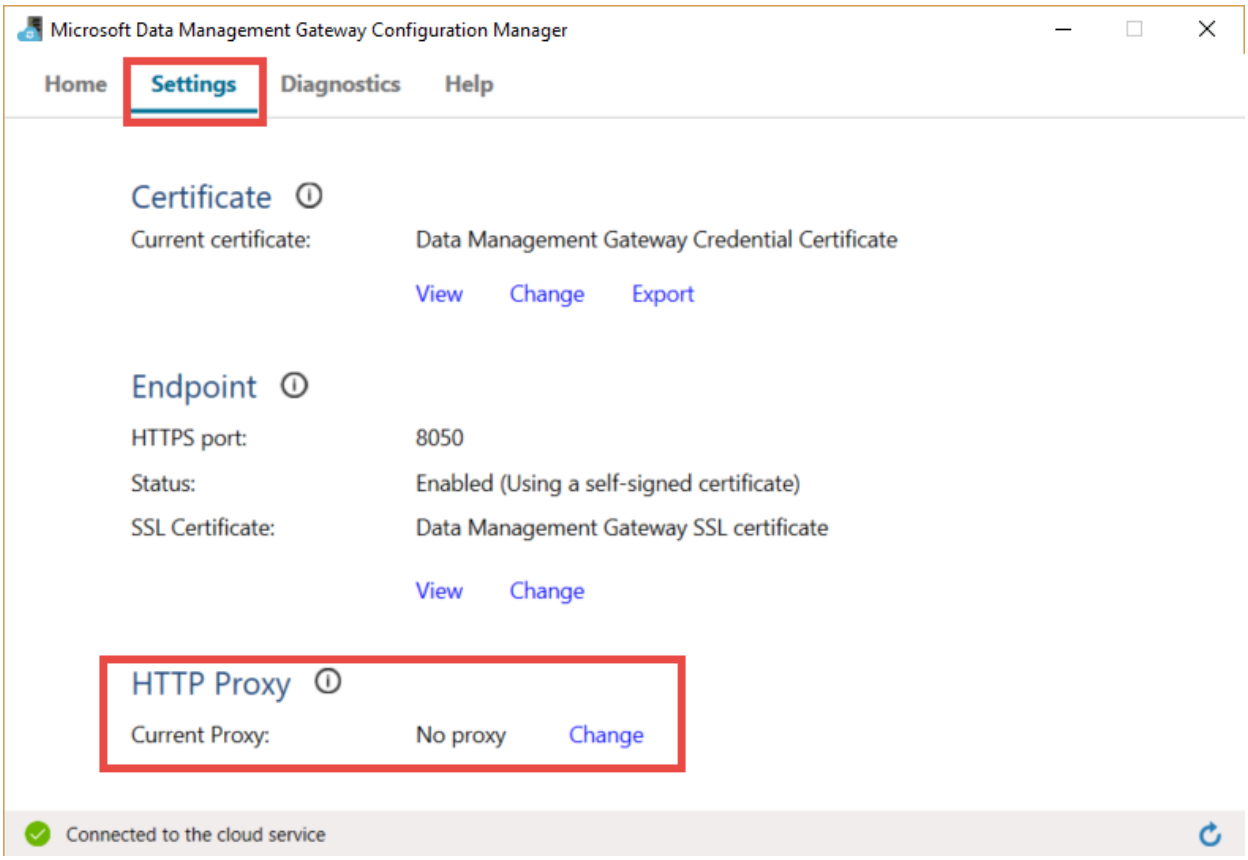
- **Do not use proxy**: The self-hosted integration runtime doesn't explicitly use any proxy to connect to cloud services.
- **Use system proxy**: The self-hosted integration runtime uses the proxy setting that is configured in diahost.exe.config and diawp.exe.config. If these files specify no proxy configuration, the self-hosted integration runtime connects to the cloud service directly without going through a proxy.
- **Use custom proxy**: Configure the HTTP proxy setting to use for the self-hosted integration runtime, instead of using configurations in diahost.exe.config and diawp.exe.config. **Address** and **Port** values are required. **User Name** and **Password** values are optional, depending on your proxy's authentication setting. All settings are encrypted with Windows DPAPI on the self-hosted integration runtime and stored locally on the machine.

The integration runtime host service restarts automatically after you save the updated proxy settings.

After you register the self-hosted integration runtime, if you want to view or update proxy settings, use Microsoft Integration Runtime Configuration Manager.

1. Open **Microsoft Integration Runtime Configuration Manager**.
2. Select the **Settings** tab.
3. Under **HTTP Proxy**, select the **Change** link to open the **Set HTTP Proxy** dialog box.
4. Select **Next**. You then see a warning that asks for your permission to save the proxy setting and restart the integration runtime host service.

You can use the configuration manager tool to view and update the HTTP proxy.



> **NOTE**
>
> If you set up a proxy server with NTLM authentication, the integration runtime host service runs under the domain account. If you later change the password for the domain account, remember to update the configuration settings for the service and restart the service. Because of this requirement, we suggest that you access the proxy server by using a dedicated domain account that doesn't require you to update the password frequently.

Configure proxy server settings

If you select the **Use system proxy** option for the HTTP proxy, the self-hosted integration runtime uses the proxy settings in diahost.exe.config and diawp.exe.config. When these files specify no proxy, the self-hosted integration runtime connects to the cloud service directly without going through a proxy. The following procedure provides instructions for updating the diahost.exe.config file:

1. In File Explorer, make a safe copy of C:\Program Files\Microsoft Integration Runtime\4.0\Shared\diahost.exe.config as a backup of the original file.
2. Open Notepad running as administrator.
3. In Notepad, open the text file C:\Program Files\Microsoft Integration Runtime\4.0\Shared\diahost.exe.config.
4. Find the default **system.net** tag as shown in the following code:

```
<system.net>
    <defaultProxy useDefaultCredentials="true" />
</system.net>
```

   You can then add proxy server details as shown in the following example:

```
<system.net>
    <defaultProxy enabled="true">
          <proxy bypassonlocal="true" proxyaddress="http://proxy.domain.org:8888/" />
    </defaultProxy>
</system.net>
```

The proxy tag allows additional properties to specify required settings like `scriptLocation`. See <proxy> Element (Network Settings) for syntax.

```
<proxy autoDetect="true|false|unspecified" bypassonlocal="true|false|unspecified"
proxyaddress="uriString" scriptLocation="uriString" usesystemdefault="true|false|unspecified "/>
```

5. Save the configuration file in its original location. Then restart the self-hosted integration runtime host service, which picks up the changes.

   To restart the service, use the services applet from Control Panel. Or from Integration Runtime Configuration Manager, select the **Stop Service** button, and then select **Start Service**.

   If the service doesn't start, you likely added incorrect XML tag syntax in the application configuration file that you edited.

> **IMPORTANT**
> Don't forget to update both diahost.exe.config and diawp.exe.config.

You also need to make sure that Microsoft Azure is in your company's allowlist. You can download the list of valid Azure IP addresses. IP Ranges for each cloud, broken down by region and by the tagged services in that cloud are now available on MS Download:

- Public: https://www.microsoft.com/download/details.aspx?id=56519
- US Gov: https://www.microsoft.com/download/details.aspx?id=57063
- Germany: https://www.microsoft.com/download/details.aspx?id=57064
- China: https://www.microsoft.com/download/details.aspx?id=57062

Possible symptoms for issues related to the firewall and proxy server

If you see error messages like the following ones, the likely reason is improper configuration of the firewall or proxy server. Such configuration prevents the self-hosted integration runtime from connecting to Data Factory or Synapse pipelines to authenticate itself. To ensure that your firewall and proxy server are properly configured, refer to the previous section.

- When you try to register the self-hosted integration runtime, you receive the following error message: "Failed to register this Integration Runtime node! Confirm that the Authentication key is valid and the integration service host service is running on this machine."
- When you open Integration Runtime Configuration Manager, you see a status of **Disconnected** or **Connecting**. When you view Windows event logs, under **Event Viewer** > **Application and Services Logs** > **Microsoft Integration Runtime**, you see error messages like this one:

```
Unable to connect to the remote server
A component of Integration Runtime has become unresponsive and restarts automatically. Component
name: Integration Runtime (Self-hosted).
```

Enable remote access from an intranet

If you use PowerShell to encrypt credentials from a networked machine other than where you installed the self-hosted integration runtime, you can enable the **Remote Access from Intranet** option. If you run PowerShell to encrypt credentials on the machine where you installed the self-hosted integration runtime, you can't enable **Remote Access from Intranet**.

Enable **Remote Access from Intranet** before you add another node for high availability and scalability.

When you run the self-hosted integration runtime setup version 3.3 or later, by default the self-hosted integration runtime installer disables **Remote Access from Intranet** on the self-hosted integration runtime machine.

When you use a firewall from a partner or others, you can manually open port 8060 or the user-configured port. If you have a firewall problem while setting up the self-hosted integration runtime, use the following command to install the self-hosted integration runtime without configuring the firewall:

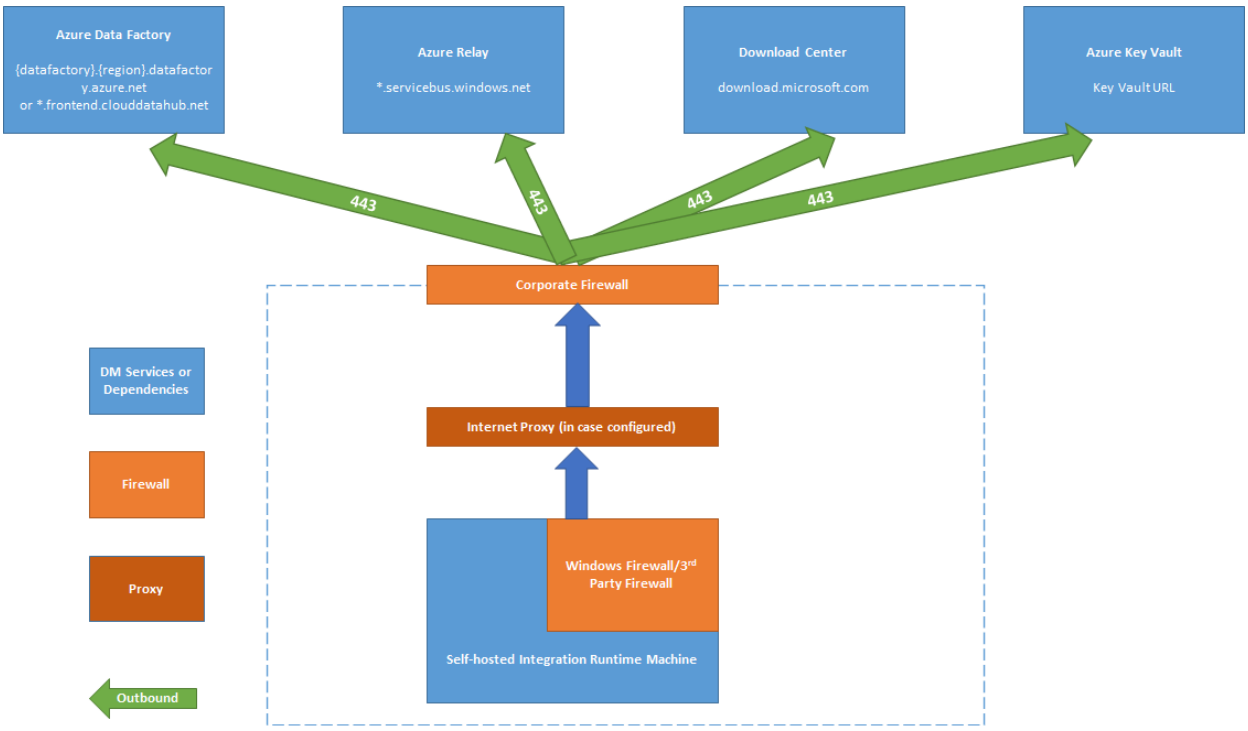
```
msiexec /q /i IntegrationRuntime.msi NOFIREWALL=1
```

If you choose not to open port 8060 on the self-hosted integration runtime machine, use mechanisms other than the Setting Credentials application to configure data-store credentials. For example, you can use the **New-AzDataFactoryV2LinkedServiceEncryptCredential** PowerShell cmdlet.

Ports and firewalls

There are two firewalls to consider:

- The *corporate firewall* that runs on the central router of the organization
- The *Windows firewall* that is configured as a daemon on the local machine where the self-hosted integration runtime is installed



At the corporate firewall level, you need to configure the following domains and outbound ports:

DOMAIN NAMES	OUTBOUND PORTS	DESCRIPTION
Public Cloud: `*.servicebus.windows.net` Azure Government: `*.servicebus.usgovcloudapi.net` China: `*.servicebus.chinacloudapi.cn`	443	Required by the self-hosted integration runtime for interactive authoring.
Public Cloud: `{datafactory}.{region}.datafactory.azure.net` or `*.frontend.clouddatahub.net` Azure Government: `{datafactory}.{region}.datafactory.azure.us` China: `{datafactory}.{region}.datafactory.azure.cn`	443	Required by the self-hosted integration runtime to connect to the Data Factory service. For new created Data Factory in public cloud, please find the FQDN from your Self-hosted Integration Runtime key which is in format {datafactory}.{region}.datafactory.azure.net. For old Data factory and Azure Synapse Analytics, if you don't see the FQDN in your Self-hosted Integration key, please use *.frontend.clouddatahub.net instead.
`download.microsoft.com`	443	Required by the self-hosted integration runtime for downloading the updates. If you have disabled auto-update, you can skip configuring this domain.
Key Vault URL	443	Required by Azure Key Vault if you store the credential in Key Vault.

At the Windows firewall level or machine level, these outbound ports are normally enabled. If they aren't, you can configure the domains and ports on a self-hosted integration runtime machine.

> **NOTE**
> As currently Azure Relay doesn't support service tag, you have to use service tag **AzureCloud** or **Internet** in NSG rules for the communication to Azure Relay. For the communication to Azure Data Factory and Synapse workspaces, you can use service tag **DataFactoryManagement** in the NSG rule setup.

Based on your source and sinks, you might need to allow additional domains and outbound ports in your corporate firewall or Windows firewall.

DOMAIN NAMES	OUTBOUND PORTS	DESCRIPTION
`*.core.windows.net`	443	Used by the self-hosted integration runtime to connect to the Azure storage account when you use the staged copy feature.
`*.database.windows.net`	1433	Required only when you copy from or to Azure SQL Database or Azure Synapse Analytics and optional otherwise. Use the staged-copy feature to copy data to SQL Database or Azure Synapse Analytics without opening port 1433.

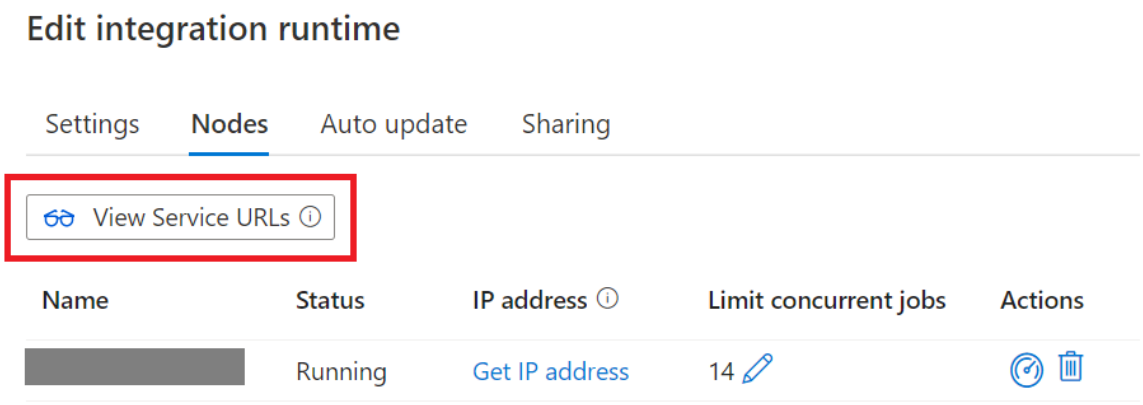
DOMAIN NAMES	OUTBOUND PORTS	DESCRIPTION
`*.azuredatalakestore.net` `login.microsoftonline.com/<tenant>/oauth2/token`	443	Required only when you copy from or to Azure Data Lake Store and optional otherwise.

For some cloud databases, such as Azure SQL Database and Azure Data Lake, you might need to allow IP addresses of self-hosted integration runtime machines on their firewall configuration.

Get URL of Azure Relay

One required domain and port that need to be put in the allowlist of your firewall is for the communication to Azure Relay. The self-hosted integration runtime uses it for interactive authoring such as test connection, browse folder list and table list, get schema, and preview data. If you don't want to allow **.servicebus.windows.net** and would like to have more specific URLs, then you can see all the FQDNs that are required by your self-hosted integration runtime from the service portal. Follow these steps:

1. Go to the service portal and select your self-hosted integration runtime.
2. In Edit page, select **Nodes**.
3. Select **View Service URLs** to get all FQDNs.



4. You can add these FQDNs in the allowlist of firewall rules.

> **NOTE**
> For the details related to Azure Relay connections protocol, see Azure Relay Hybrid Connections protocol.

Copy data from a source to a sink

Ensure that you properly enable firewall rules on the corporate firewall, the Windows firewall of the self-hosted integration runtime machine, and the data store itself. Enabling these rules lets the self-hosted integration runtime successfully connect to both source and sink. Enable rules for each data store that is involved in the copy operation.

For example, to copy from an on-premises data store to a SQL Database sink or an Azure Synapse Analytics sink, take the following steps:

1. Allow outbound TCP communication on port 1433 for both the Windows firewall and the corporate firewall.
2. Configure the firewall settings of the SQL Database to add the IP address of the self-hosted integration runtime machine to the list of allowed IP addresses.

> **NOTE**
> If your firewall doesn't allow outbound port 1433, the self-hosted integration runtime can't access the SQL database directly. In this case, you can use a staged copy to SQL Database and Azure Synapse Analytics. In this scenario, you require only HTTPS (port 443) for the data movement.

Credentials store

There are two ways to store the credentials when using self-hosted integration runtime:

1. Use Azure Key Vault. This is the recommended way to store your credentials in Azure. The self-hosted integration runtime can directly get the credentials from Azure Key Vault which can highly avoid some potential security issues or any credential in-sync problems between self-hosted integration runtime nodes.
2. Store credentials locally. The credentials will be push to the machine of your self-hosted integration runtime and be encrypted. When your self-hosted integration runtime is recovered from crash, you can either recover credential from the one you backup before or edit linked service and let the credential be pushed to self-hosted integration runtime again. Otherwise, the pipeline doesn't work due to the lack of credential when running via self-hosted integration runtime.

> **NOTE**
> If you prefer to store the credential locally, your need to put the domain for interactive authoring in the allowlist of your firewall and open the port. This channel is also for the self-hosted integration runtime to get the credentials. For the domain and port needed for interactive authoring, refer to Ports and firewalls

Installation best practices

You can install the self-hosted integration runtime by downloading a Managed Identity setup package from Microsoft Download Center. See the article Move data between on-premises and cloud for step-by-step instructions.

- Configure a power plan on the host machine for the self-hosted integration runtime so that the machine doesn't hibernate. If the host machine hibernates, the self-hosted integration runtime goes offline.
- Regularly back up the credentials associated with the self-hosted integration runtime.
- To automate self-hosted IR setup operations, refer to Set up an existing self hosted IR via PowerShell.

Next steps

For step-by-step instructions, see Tutorial: Copy on-premises data to cloud.

Create a trigger that runs a pipeline in response to a storage event

8/19/2022 • 10 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article describes the Storage Event Triggers that you can create in your Data Factory or Synapse pipelines.

Event-driven architecture (EDA) is a common data integration pattern that involves production, detection, consumption, and reaction to events. Data integration scenarios often require customers to trigger pipelines based on events happening in storage account, such as the arrival or deletion of a file in Azure Blob Storage account. Data Factory and Synapse pipelines natively integrate with Azure Event Grid, which lets you trigger pipelines on such events.

> **NOTE**
> The integration described in this article depends on Azure Event Grid. Make sure that your subscription is registered with the Event Grid resource provider. For more info, see Resource providers and types. You must be able to do the *Microsoft.EventGrid/eventSubscriptions/** action. This action is part of the EventGrid EventSubscription Contributor built-in role.

> **IMPORTANT**
> If you are using this feature in Azure Synapse Analytics, please ensure that your subscription is also registered with Data Factory resource provider, or otherwise you will get an error stating that *the creation of an "Event Subscription" failed*.

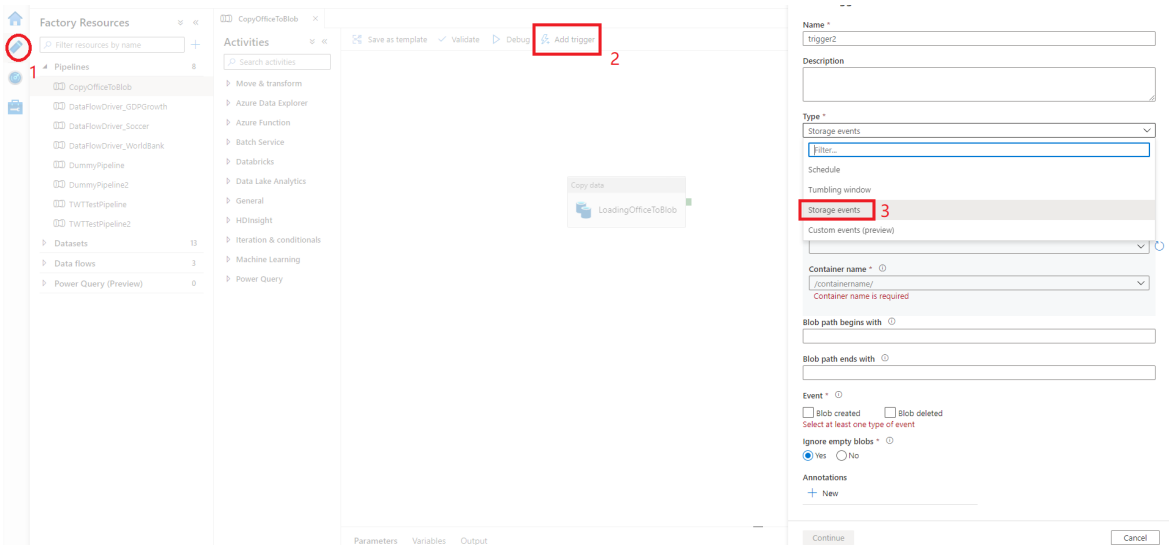
> **NOTE**
> If the blob storage account resides behind a private endpoint and blocks public network access, you need to configure network rules to allow communications from blob storage to Azure Event Grid. You can either grant storage access to trusted Azure services, such as Event Grid, following Storage documentation, or configure private endpoints for Event Grid that map to VNet address space, following Event Grid documentation

Create a trigger with UI

This section shows you how to create a storage event trigger within the Azure Data Factory and Synapse pipeline User Interface.

1. Switch to the **Edit** tab in Data Factory, or the **Integrate** tab in Azure Synapse.
2. Select **Trigger** on the menu, then select **New/Edit**.
3. On the **Add Triggers** page, select **Choose trigger...**, then select **+New**.
4. Select trigger type **Storage Event**

   - Azure Data Factory
   - Azure Synapse



5. Select your storage account from the Azure subscription dropdown or manually using its Storage account resource ID. Choose which container you wish the events to occur on. Container selection is required, but be mindful that selecting all containers can lead to a large number of events.

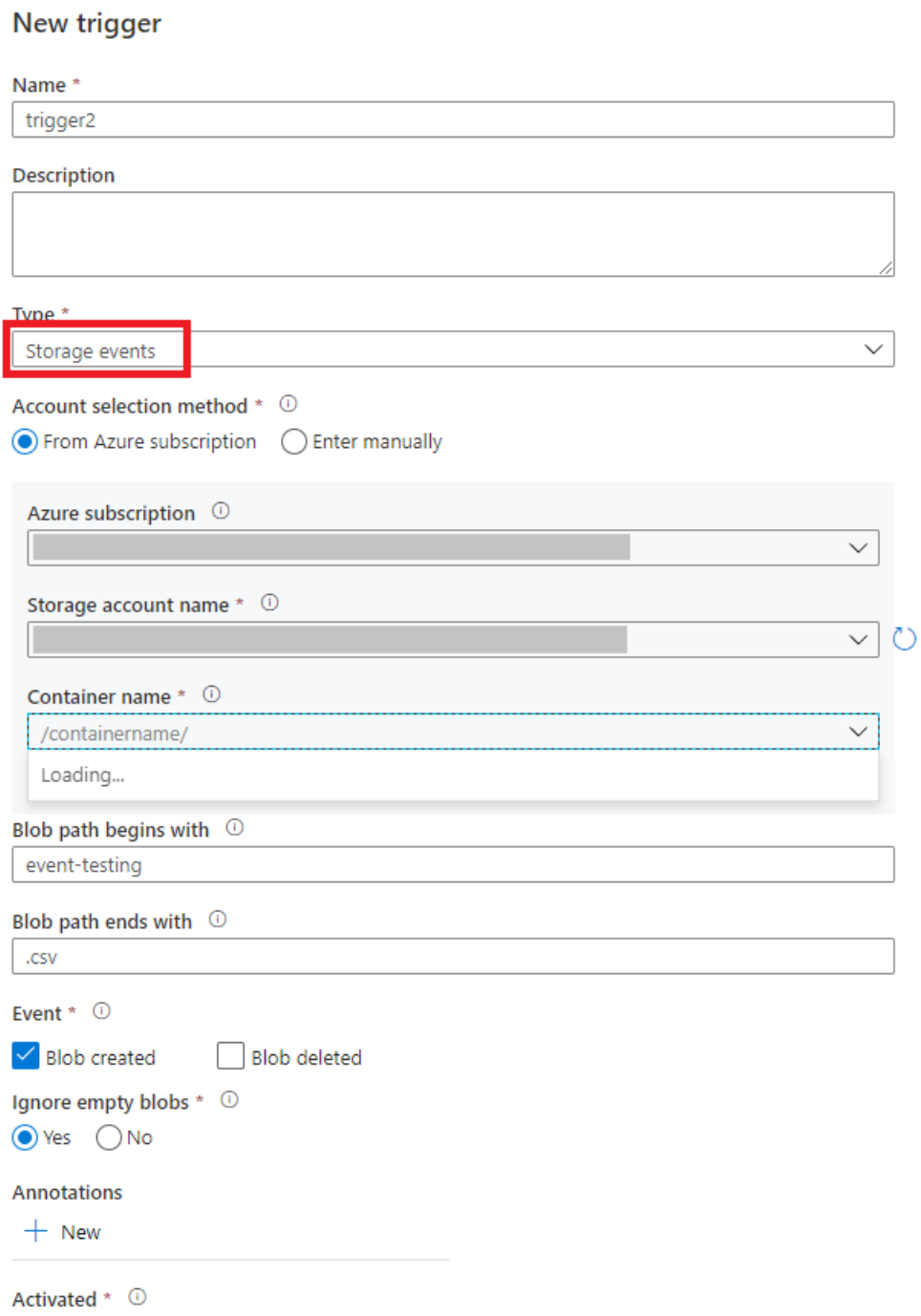
> **NOTE**
>
> The Storage Event Trigger currently supports only Azure Data Lake Storage Gen2 and General-purpose version 2 storage accounts. Due to an Azure Event Grid limitation, Azure Data Factory only supports a maximum of 500 storage event triggers per storage account. If you hit the limit, please contact support for recommendations and increasing the limit upon evaluation by Event Grid team.

> **NOTE**
>
> To create a new or modify an existing Storage Event Trigger, the Azure account used to log into the service and publish the storage event trigger must have appropriate role based access control (Azure RBAC) permission on the storage account. No additional permission is required: Service Principal for the Azure Data Factory and Azure Synapse does *not* need special permission to either the Storage account or Event Grid. For more information about access control, see Role based access control section.

6. The **Blob path begins with** and **Blob path ends with** properties allow you to specify the containers, folders, and blob names for which you want to receive events. Your storage event trigger requires at least one of these properties to be defined. You can use variety of patterns for both **Blob path begins with** and **Blob path ends with** properties, as shown in the examples later in this article.

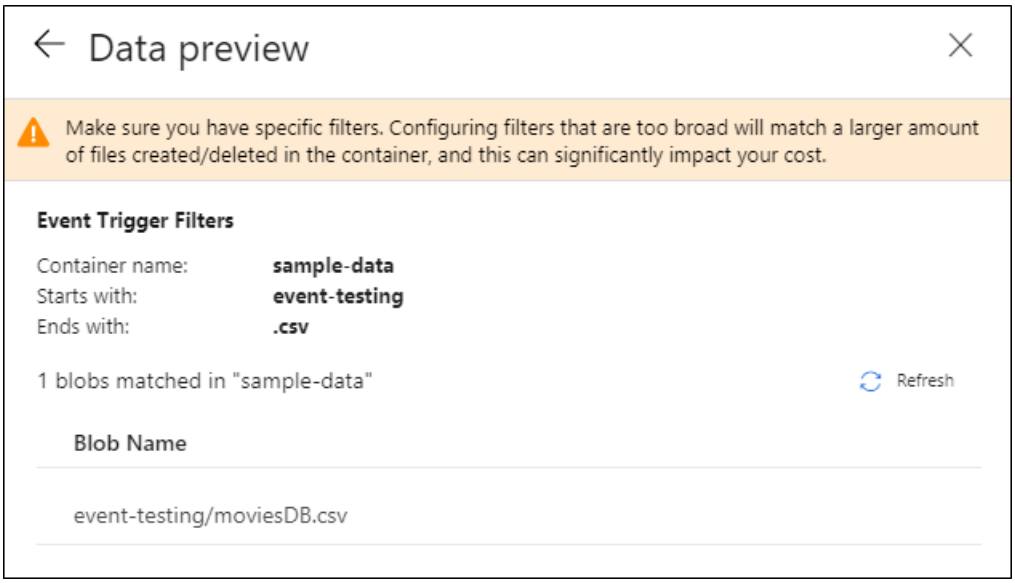
   - **Blob path begins with:** The blob path must start with a folder path. Valid values include `2018/` and `2018/april/shoes.csv`. This field can't be selected if a container isn't selected.
   - **Blob path ends with:** The blob path must end with a file name or extension. Valid values include `shoes.csv` and `.csv`. Container and folder names, when specified, they must be separated by a `/blobs/` segment. For example, a container named 'orders' can have a value of `/orders/blobs/2018/april/shoes.csv`. To specify a folder in any container, omit the leading '/' character. For example, `april/shoes.csv` will trigger an event on any file named `shoes.csv` in folder a called 'april' in any container.
   - Note that Blob path **begins with** and **ends with** are the only pattern matching allowed in Storage Event Trigger. Other types of wildcard matching aren't supported for the trigger type.

7. Select whether your trigger will respond to a **Blob created** event, **Blob deleted** event, or both. In your specified storage location, each event will trigger the Data Factory and Synapse pipelines associated with the trigger.



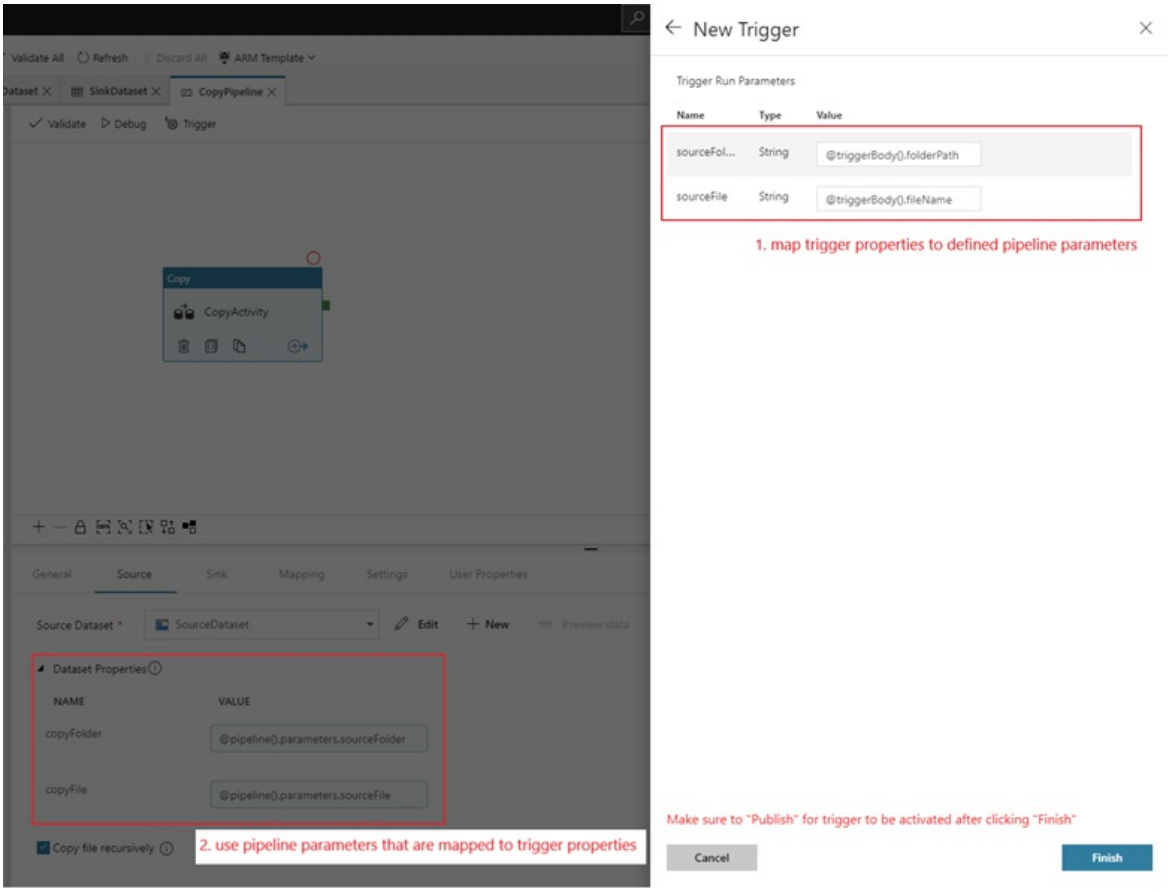
8. Select whether or not your trigger ignores blobs with zero bytes.

9. After you configure you trigger, click on **Next: Data preview**. This screen shows the existing blobs matched by your storage event trigger configuration. Make sure you've specific filters. Configuring filters that are too broad can match a large number of files created/deleted and may significantly impact your cost. Once your filter conditions have been verified, click **Finish**.



10. To attach a pipeline to this trigger, go to the pipeline canvas and click **Trigger** and select **New/Edit**. When the side nav appears, click on the **Choose trigger...** dropdown and select the trigger you created. Click **Next: Data preview** to confirm the configuration is correct and then **Next** to validate the Data preview is correct.

11. If your pipeline has parameters, you can specify them on the trigger runs parameter side nav. The storage event trigger captures the folder path and file name of the blob into the properties `@triggerBody().folderPath` and `@triggerBody().fileName`. To use the values of these properties in a pipeline, you must map the properties to pipeline parameters. After mapping the properties to parameters, you can access the values captured by the trigger through the `@pipeline().parameters.parameterName` expression throughout the pipeline. For detailed explanation, see Reference Trigger Metadata in Pipelines



In the preceding example, the trigger is configured to fire when a blob path ending in .csv is created in the folder *event-testing* in the container *sample-data*. The **folderPath** and **fileName** properties capture the

location of the new blob. For example, when MoviesDB.csv is added to the path sample-data/event-testing, `@triggerBody().folderPath` has a value of `sample-data/event-testing` and `@triggerBody().fileName` has a value of `moviesDB.csv`. These values are mapped, in the example, to the pipeline parameters `sourceFolder` and `sourceFile`, which can be used throughout the pipeline as `@pipeline().parameters.sourceFolder` and `@pipeline().parameters.sourceFile` respectively.

> **NOTE**
>
> If you are creating your pipeline and trigger in Azure Synapse Analytics, you must use `@trigger().outputs.body.fileName` and `@trigger().outputs.body.folderPath` as parameters. Those two properties capture blob information. Use those properties instead of using `@triggerBody().fileName` and `@triggerBody().folderPath`.

12. Click **Finish** once you are done.

JSON schema

The following table provides an overview of the schema elements that are related to storage event triggers:

JSON ELEMENT	DESCRIPTION	TYPE	ALLOWED VALUES	REQUIRED
scope	The Azure Resource Manager resource ID of the Storage Account.	String	Azure Resource Manager ID	Yes
events	The type of events that cause this trigger to fire.	Array	Microsoft.Storage.BlobCreated, Microsoft.Storage.BlobDeleted	Yes, any combination of these values.
blobPathBeginsWith	The blob path must begin with the pattern provided for the trigger to fire. For example, `/records/blobs/december/` only fires the trigger for blobs in the `december` folder under the `records` container.	String		Provide a value for at least one of these properties: `blobPathBeginsWith` or `blobPathEndsWith`.
blobPathEndsWith	The blob path must end with the pattern provided for the trigger to fire. For example, `december/boxes.csv` only fires the trigger for blobs named `boxes` in a `december` folder.	String		Provide a value for at least one of these properties: `blobPathBeginsWith` or `blobPathEndsWith`.

JSON ELEMENT	DESCRIPTION	TYPE	ALLOWED VALUES	REQUIRED
ignoreEmptyBlobs	Whether or not zero-byte blobs will trigger a pipeline run. By default, this is set to true.	Boolean	true or false	No

Examples of storage event triggers

This section provides examples of storage event trigger settings.

> **IMPORTANT**
> You have to include the `/blobs/` segment of the path, as shown in the following examples, whenever you specify container and folder, container and file, or container, folder, and file. For **blobPathBeginsWith**, the UI will automatically add `/blobs/` between the folder and container name in the trigger JSON.

PROPERTY	EXAMPLE	DESCRIPTION
Blob path begins with	`/containername/`	Receives events for any blob in the container.
Blob path begins with	`/containername/blobs/foldername/`	Receives events for any blobs in the `containername` container and `foldername` folder.
Blob path begins with	`/containername/blobs/foldername/subfoldername/`	You can also reference a subfolder.
Blob path begins with	`/containername/blobs/foldername/file.txt`	Receives events for a blob named `file.txt` in the `foldername` folder under the `containername` container.
Blob path ends with	`file.txt`	Receives events for a blob named `file.txt` in any path.
Blob path ends with	`/containername/blobs/file.txt`	Receives events for a blob named `file.txt` under container `containername`.
Blob path ends with	`foldername/file.txt`	Receives events for a blob named `file.txt` in `foldername` folder under any container.

Role-based access control

Azure Data Factory and Synapse pipelines use Azure role-based access control (Azure RBAC) to ensure that unauthorized access to listen to, subscribe to updates from, and trigger pipelines linked to blob events, are strictly prohibited.

- To successfully create a new or update an existing Storage Event Trigger, the Azure account signed into the service needs to have appropriate access to the relevant storage account. Otherwise, the operation will fail with *Access Denied*.

- Azure Data Factory and Azure Synapse need no special permission to your Event Grid, and you do *not* need to assign special RBAC permission to the Data Factory or Azure Synapse service principal for the operation.

Any of following RBAC settings works for storage event trigger:

- Owner role to the storage account
- Contributor role to the storage account
- *Microsoft.EventGrid/EventSubscriptions/Write* permission to storage account */subscriptions/####/resourceGroups/####/providers/Microsoft.Storage/storageAccounts/storageAccountName*

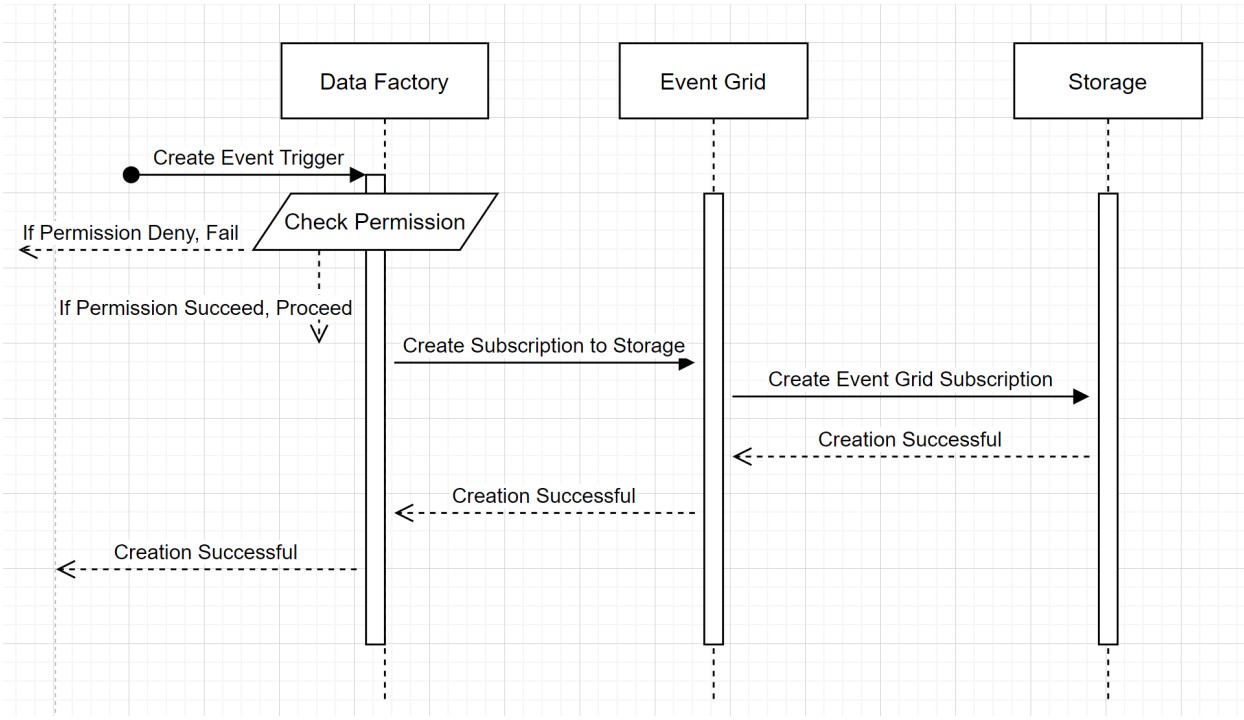
Specifically,

- When authoring in the data factory (in the development environment for instance), the Azure account signed in needs to have the above permission
- When publishing through CI/CD, the account used to publish the ARM template into the testing or production factory needs to have the above permission.

In order to understand how the service delivers the two promises, let's take back a step and take a peek behind the scenes. Here are the high-level work flows for integration between Azure Data Factory/Azure Synapse, Storage, and Event Grid.

Create a new Storage Event Trigger

This high-level work flow describes how Azure Data Factory interacts with Event Grid to create a Storage Event Trigger. For Azure Synapse the data flow is the same, with Synapse pipelines taking the role of the Data Factory in the diagram below.



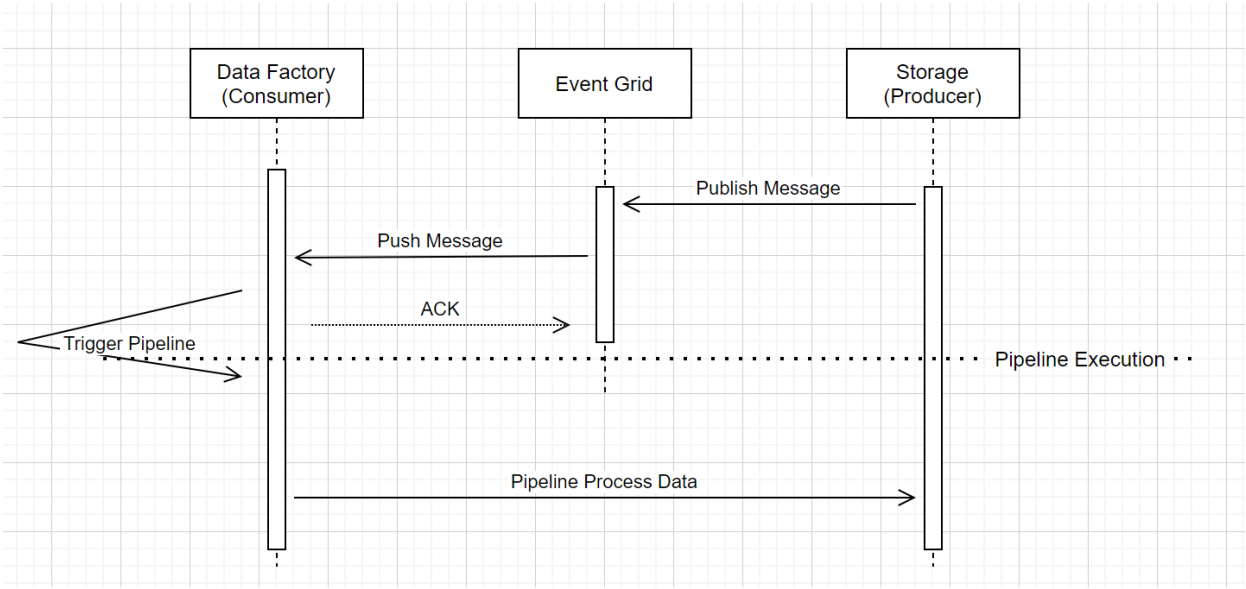
Two noticeable call outs from the work flows:

- Azure Data Factory and Azure Synapse make *no* direct contact with Storage account. Request to create a subscription is instead relayed and processed by Event Grid. Hence, the service needs no permission to Storage account for this step.

- Access control and permission checking happen within the service. Before the service sends a request to subscribe to storage event, it checks the permission for the user. More specifically, it checks whether the Azure account signed in and attempting to create the Storage Event trigger has appropriate access to the relevant storage account. If the permission check fails, trigger creation also fails.

Storage event trigger pipeline run

This high-level work flows describe how Storage event triggers pipeline run through Event Grid. For Azure Synapse the data flow is the same, with Synapse pipelines taking the role of the Data Factory in the diagram below.



There are three noticeable call outs in the workflow related to Event triggering pipelines within the service:

- Event Grid uses a Push model that relays the message as soon as possible when storage drops the message into the system. This is different from messaging system, such as Kafka where a Pull system is used.
- Event Trigger serves as an active listener to the incoming message and it properly triggers the associated pipeline.
- Storage Event Trigger itself makes no direct contact with Storage account
    - That said, if you have a Copy or other activity inside the pipeline to process the data in Storage account, the service will make direct contact with Storage, using the credentials stored in the Linked Service. Ensure that Linked Service is set up appropriately
    - However, if you make no reference to the Storage account in the pipeline, you do not need to grant permission to the service to access Storage account

Next steps

- For detailed information about triggers, see Pipeline execution and triggers.
- Learn how to reference trigger metadata in pipeline, see Reference Trigger Metadata in Pipeline Runs

Create a trigger that runs a pipeline on a schedule

8/19/2022 • 24 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides information about the schedule trigger and the steps to create, start, and monitor a schedule trigger. For other types of triggers, see Pipeline execution and triggers.

When creating a schedule trigger, you specify a schedule (start date, recurrence, end date etc.) for the trigger, and associate with a pipeline. Pipelines and triggers have a many-to-many relationship. Multiple triggers can kick off a single pipeline. A single trigger can kick off multiple pipelines.

The following sections provide steps to create a schedule trigger in different ways.

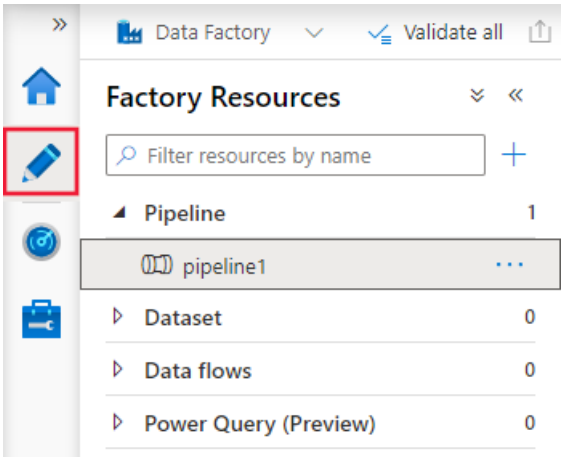
Azure Data Factory and Synapse portal experience

You can create a **schedule trigger** to schedule a pipeline to run periodically (hourly, daily, etc.).

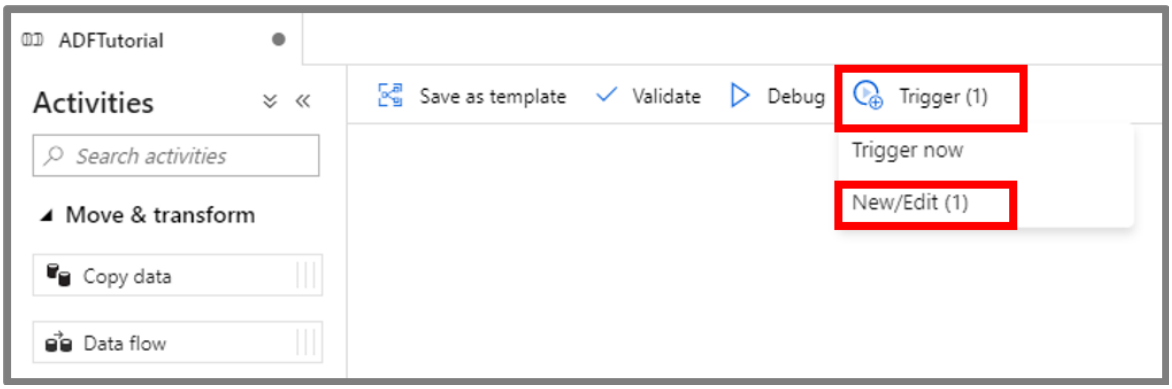
> **NOTE**
> For a complete walkthrough of creating a pipeline and a schedule trigger, which associates the trigger with the pipeline, and runs and monitors the pipeline, see Quickstart: create a data factory using Data Factory UI.

1. Switch to the **Edit** tab in Data Factory or the Integrate tab in Azure Synapse.

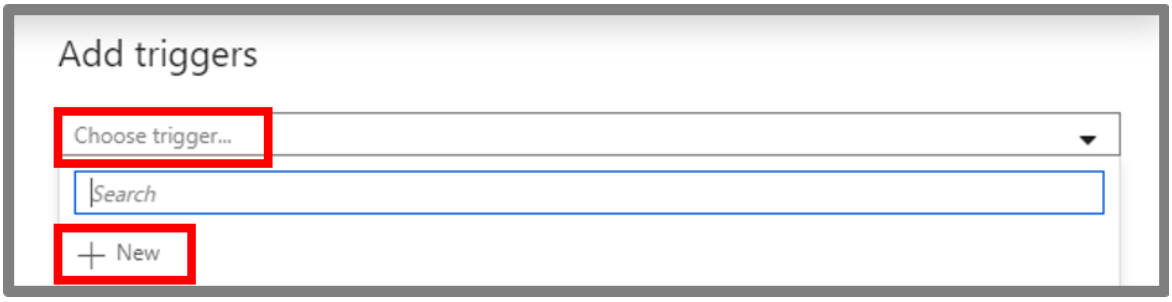
   - Azure Data Factory
   - Azure Synapse



2. Select **Trigger** on the menu, then select **New/Edit**.



3. On the **Add Triggers** page, select **Choose trigger...**, then select **+New**.



4. On the **New Trigger** page, do the following steps:

   a. Confirm that **Schedule** is selected for **Type**.

   b. Specify the start datetime of the trigger for **Start Date**. It's set to the current datetime in Coordinated Universal Time (UTC) by default.

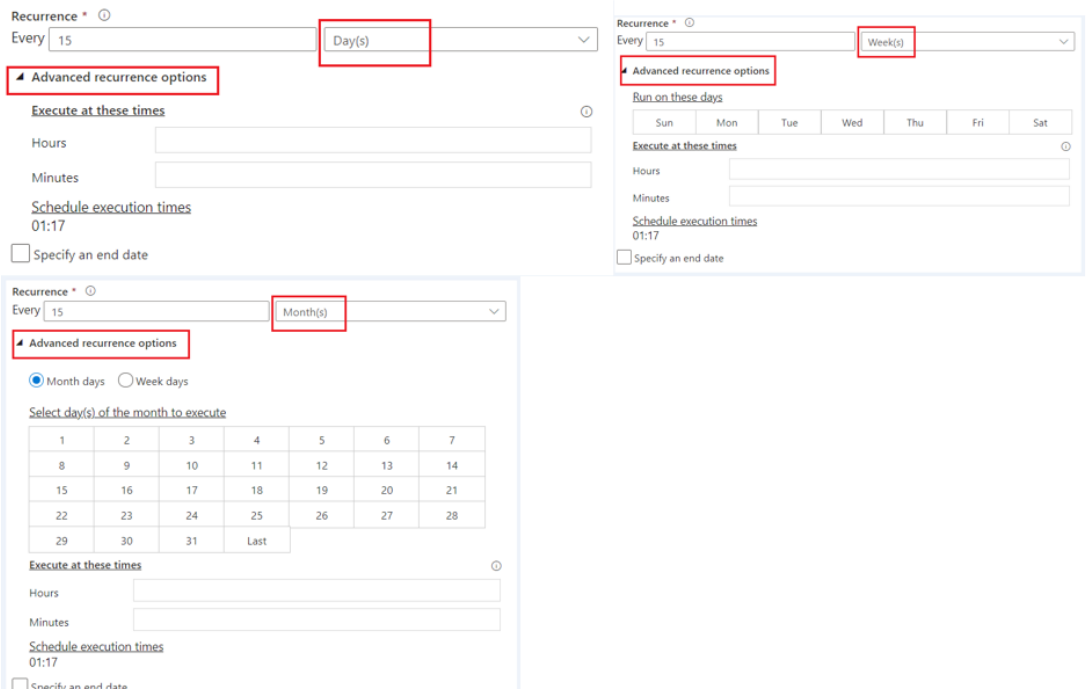
   c. Specify the time zone that the trigger will be created in. The time zone setting will apply to **Start Date**, **End Date**, and **Schedule Execution Times** in Advanced recurrence options. Changing Time Zone setting will not automatically change your start date. Make sure the Start Date is correct in the specified time zone. Please note that Scheduled Execution time of Trigger will be considered post the Start Date (Ensure Start Date is at least 1 minute lesser than the Execution time else it will trigger pipeline in next recurrence).

   > **NOTE**
   > For time zones that observe daylight saving, trigger time will auto-adjust for the twice a year change, if the recurrence is set to *Days* or above. To opt out of the daylight saving change, please select a time zone that does not observe daylight saving, for instance UTC

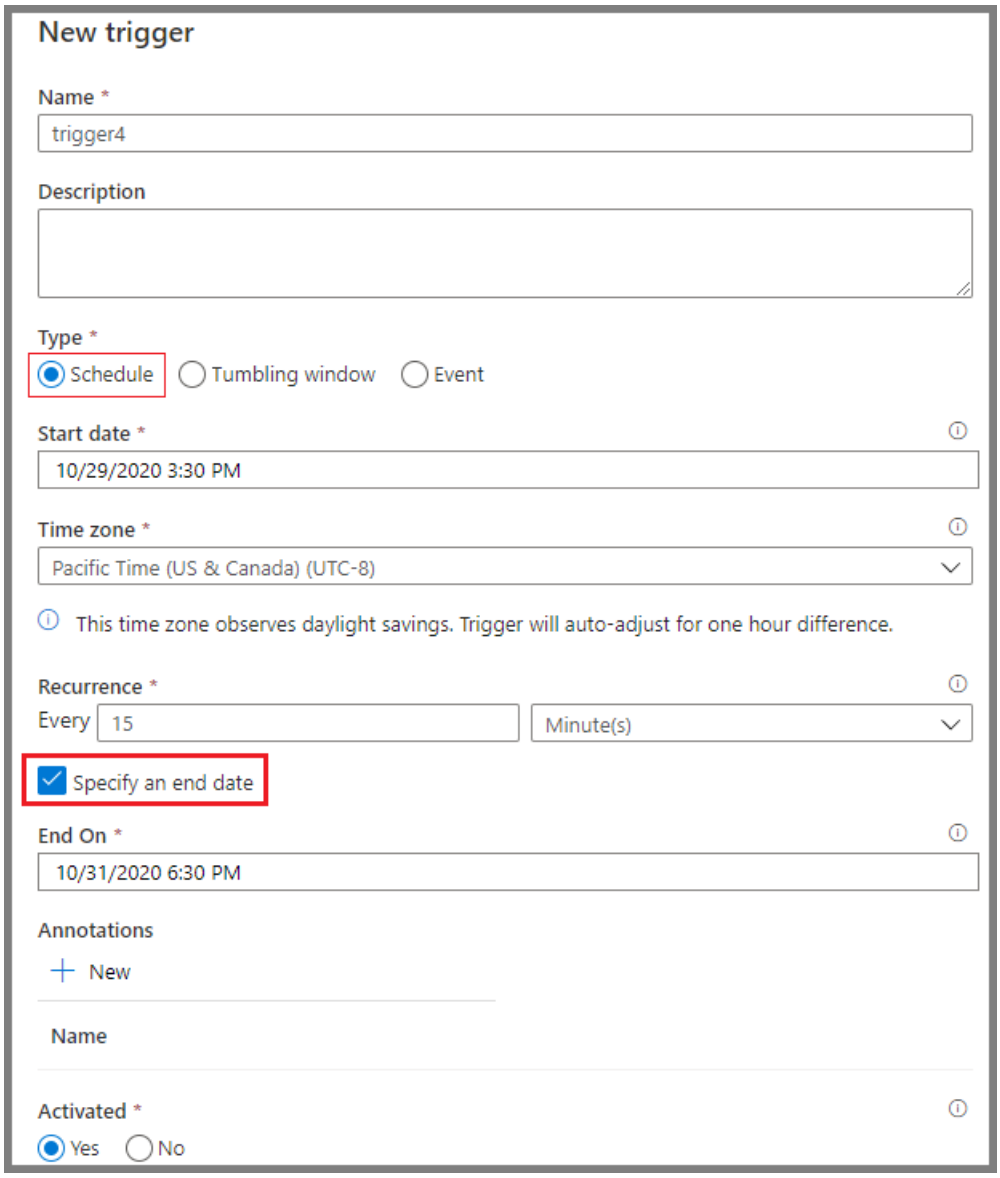
   > **IMPORTANT**
   > Daylight saving adjustment only happens for trigger with recurrence set to *Days* or above. If the trigger is set to *Hours* or *Minutes* frequency, it will continue to fire at regular intervals.

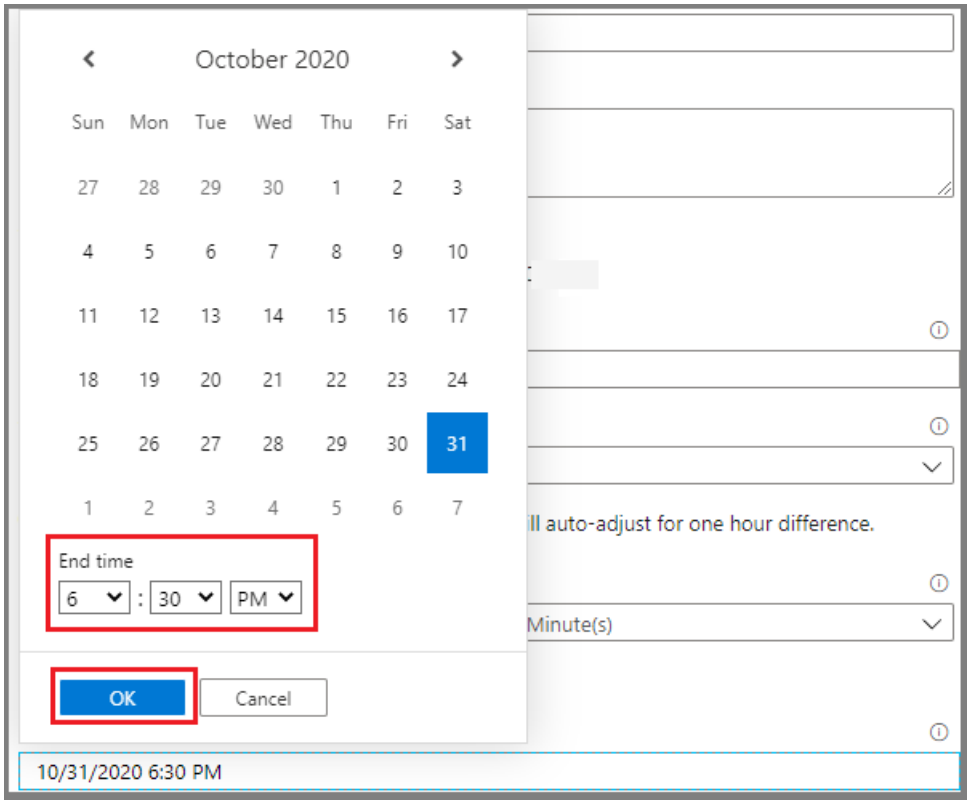
   d. Specify **Recurrence** for the trigger. Select one of the values from the drop-down list (Every minute, Hourly, Daily, Weekly, and Monthly). Enter the multiplier in the text box. For example, if you want the trigger to run once for every 15 minutes, you select **Every Minute**, and enter **15** in the text box.

   e. In the **Recurrence**, if you choose "Day(s), Week(s) or Month(s)" from the drop-down, you can find "Advanced recurrence options".



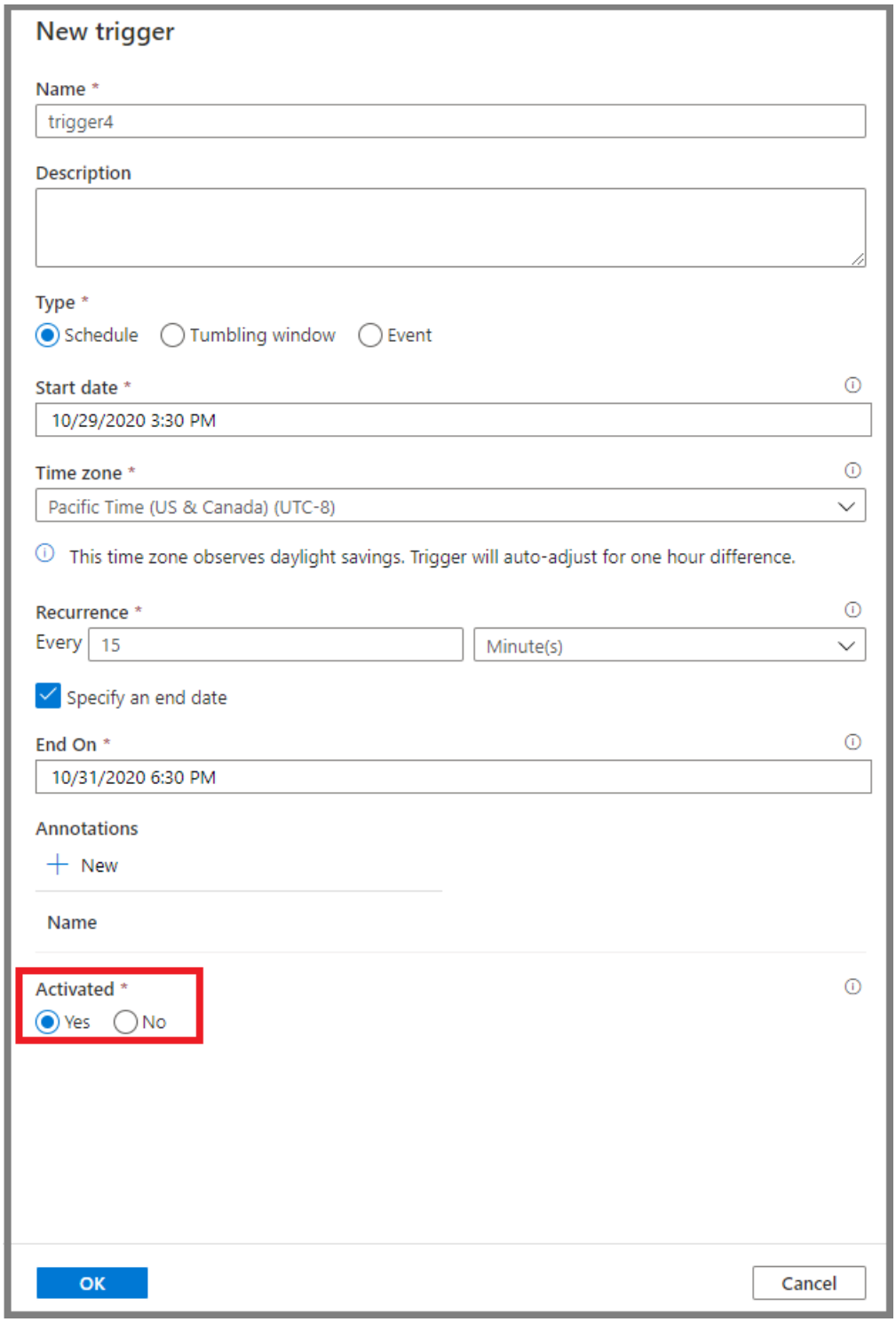
f. To specify an end date time, select **Specify an End Date**, and specify *Ends On*, then select **OK**. There is a cost associated with each pipeline run. If you are testing, you may want to ensure that the pipeline is triggered only a couple of times. However, ensure that there is enough time for the pipeline to run between the publish time and the end time. The trigger comes into effect only after you publish the solution, not when you save the trigger in the UI.



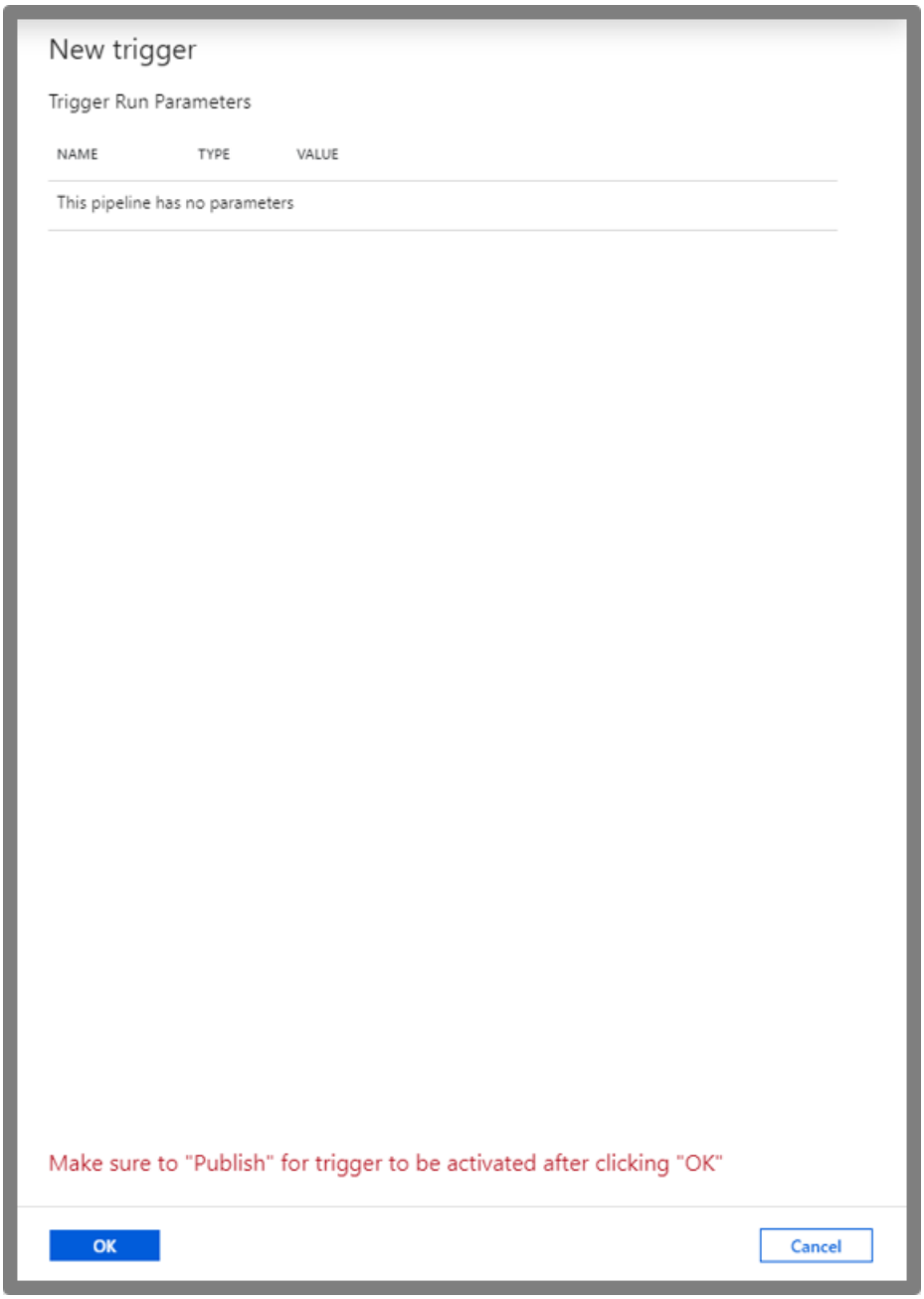


5. In the **New Trigger** window, select **Yes** in the **Activated** option, then select **OK**. You can use this

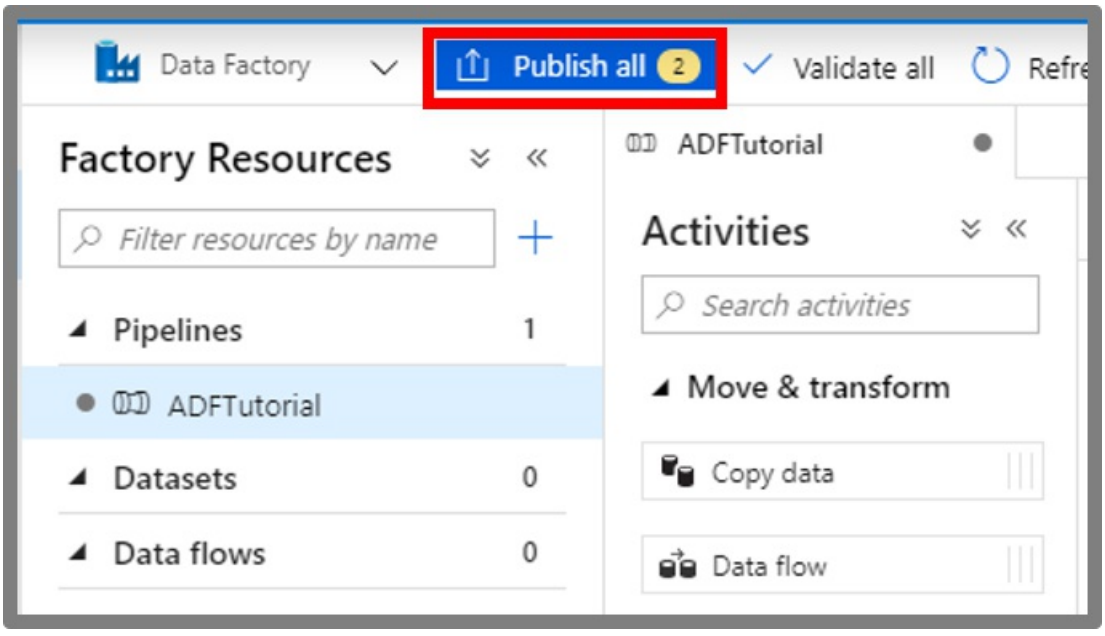
checkbox to deactivate the trigger later.



6. In the **New Trigger** window, review the warning message, then select **OK**.

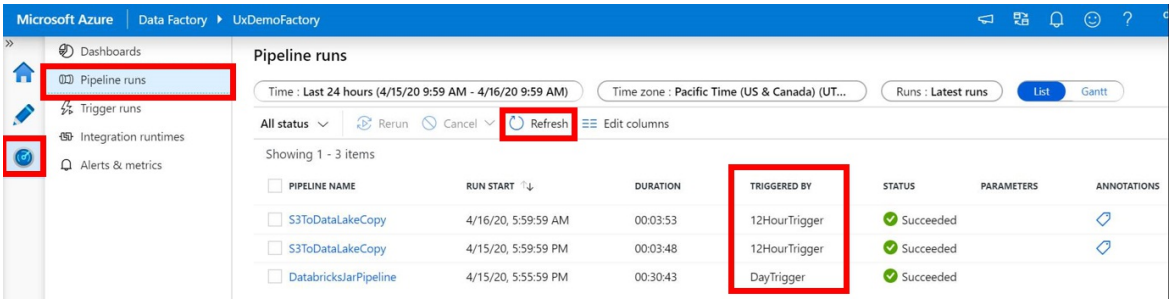


7. Select **Publish all** to publish the changes. Until you publish the changes, the trigger doesn't start triggering the pipeline runs.



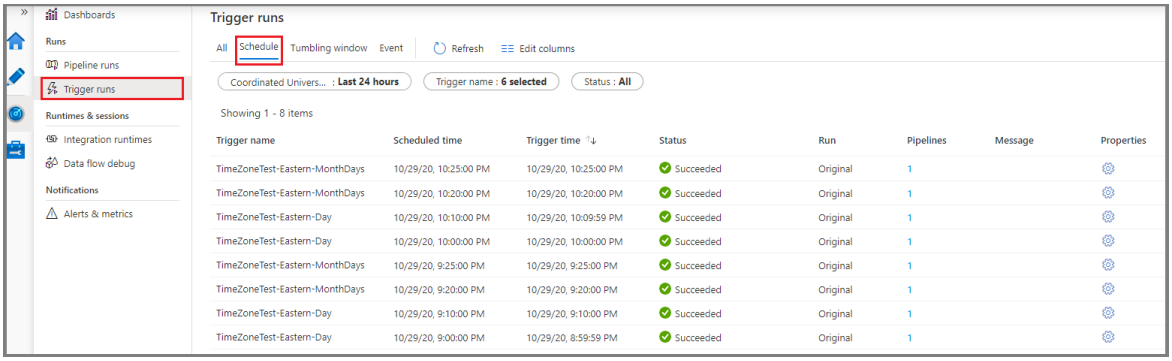
8. Switch to the **Pipeline runs** tab on the left, then select **Refresh** to refresh the list. You will see the pipeline runs triggered by the scheduled trigger. Notice the values in the **Triggered By** column. If you use the **Trigger Now** option, you will see the manual trigger run in the list.

   - Azure Data Factory
   - Azure Synapse



9. Switch to the **Trigger Runs \ Schedule** view.

   - Azure Data Factory
   - Azure Synapse



Azure PowerShell

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

This section shows you how to use Azure PowerShell to create, start, and monitor a schedule trigger. To see this sample working, first go through the Quickstart: Create a data factory by using Azure PowerShell. Then, add the following code to the main method, which creates and starts a schedule trigger that runs every 15 minutes. The trigger is associated with a pipeline named **Adfv2QuickStartPipeline** that you create as part of the Quickstart.

Prerequisites

- **Azure subscription**. If you don't have an Azure subscription, create a free account before you begin.
- **Azure PowerShell**. Follow the instructions in Install Azure PowerShell on Windows with PowerShellGet.

Sample Code

1. Create a JSON file named **MyTrigger.json** in the C:\ADFv2QuickStartPSH\ folder with the following content:

> **IMPORTANT**
>
> Before you save the JSON file, set the value of the **startTime** element to the current UTC time. Set the value of the **endTime** element to one hour past the current UTC time.

```
{
    "properties": {
        "name": "MyTrigger",
        "type": "ScheduleTrigger",
        "typeProperties": {
            "recurrence": {
                "frequency": "Minute",
                "interval": 15,
                "startTime": "2017-12-08T00:00:00Z",
                "endTime": "2017-12-08T01:00:00Z",
                "timeZone": "UTC"
            }
        },
        "pipelines": [{
                "pipelineReference": {
                    "type": "PipelineReference",
                    "referenceName": "Adfv2QuickStartPipeline"
                },
                "parameters": {
                    "inputPath": "adftutorial/input",
                    "outputPath": "adftutorial/output"
                }
            }
        ]
    }
}
```

In the JSON snippet:

- The **type** element of the trigger is set to "ScheduleTrigger".
- The **frequency** element is set to "Minute" and the **interval** element is set to 15. As such, the trigger runs the pipeline every 15 minutes between the start and end times.
- The **timeZone** element specifies the time zone that the trigger is created in. This setting affects both **startTime** and **endTime**.
- The **endTime** element is one hour after the value of the **startTime** element. As such, the trigger runs the pipeline 15 minutes, 30 minutes, and 45 minutes after the start time. Don't forget to update the start time to the current UTC time, and the end time to one hour past the start time.

  > **IMPORTANT**
  >
  > For UTC timezone, the startTime and endTime need to follow format 'yyyy-MM-ddTHH:mm:ssZ', while for other timezones, startTime and endTime follow 'yyyy-MM-ddTHH:mm:ss'.
  >
  > Per ISO 8601 standard, the *Z* suffix to timestamp mark the datetime to UTC timezone, and render timeZone field useless. While missing *Z* suffix for UTC time zone will result in an error upon trigger *activation*.

- The trigger is associated with the **Adfv2QuickStartPipeline** pipeline. To associate multiple pipelines with a trigger, add more **pipelineReference** sections.
- The pipeline in the Quickstart takes two **parameters** values: **inputPath** and **outputPath**. And you

pass values for these parameters from the trigger.

2. Create a trigger by using the Set-AzDataFactoryV2Trigger cmdlet:

```
Set-AzDataFactoryV2Trigger -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -
Name "MyTrigger" -DefinitionFile "C:\ADFv2QuickStartPSH\MyTrigger.json"
```

3. Confirm that the status of the trigger is **Stopped** by using the Get-AzDataFactoryV2Trigger cmdlet:

```
Get-AzDataFactoryV2Trigger -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -
Name "MyTrigger"
```

4. Start the trigger by using the Start-AzDataFactoryV2Trigger cmdlet:

```
Start-AzDataFactoryV2Trigger -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName
-Name "MyTrigger"
```

5. Confirm that the status of the trigger is **Started** by using the Get-AzDataFactoryV2Trigger cmdlet:

```
Get-AzDataFactoryV2Trigger -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -
Name "MyTrigger"
```

6. Get the trigger runs in Azure PowerShell by using the Get-AzDataFactoryV2TriggerRun cmdlet. To get the information about the trigger runs, execute the following command periodically. Update the **TriggerRunStartedAfter** and **TriggerRunStartedBefore** values to match the values in your trigger definition:

```
Get-AzDataFactoryV2TriggerRun -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName
-TriggerName "MyTrigger" -TriggerRunStartedAfter "2017-12-08T00:00:00" -TriggerRunStartedBefore
"2017-12-08T01:00:00"
```

> **NOTE**
> Trigger time of Schedule triggers are specified in UTC timestamp. *TriggerRunStartedAfter* and *TriggerRunStartedBefore* also expects UTC timestamp

To monitor the trigger runs and pipeline runs in the Azure portal, see Monitor pipeline runs.

Azure CLI

This section shows you how to use Azure CLI to create, start, and monitor a schedule trigger. To see this sample working, first go through the Quickstart: Create an Azure Data Factory using Azure CLI. Then, follow the steps below to create and start a schedule trigger that runs every 15 minutes. The trigger is associated with a pipeline named **Adfv2QuickStartPipeline** that you create as part of the Quickstart.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Azure Cloud Shell Quickstart - Bash.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows

or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.

- If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
- When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
- Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample Code

1. In your working directory, create a JSON file named **MyTrigger.json** with the trigger's properties. For this example use the following content:

> **IMPORTANT**
> Before you save the JSON file, set the value of the **startTime** element to the current UTC time. Set the value of the **endTime** element to one hour past the current UTC time.

```
{
    "name": "MyTrigger",
    "type": "ScheduleTrigger",
    "typeProperties": {
        "recurrence": {
            "frequency": "Minute",
            "interval": 15,
            "startTime": "2017-12-08T00:00:00Z",
            "endTime": "2017-12-08T01:00:00Z",
            "timeZone": "UTC"
        }
    },
    "pipelines": [{
            "pipelineReference": {
                "type": "PipelineReference",
                "referenceName": "Adfv2QuickStartPipeline"
            },
            "parameters": {
                "inputPath": "adftutorial/input",
                "outputPath": "adftutorial/output"
            }
        }
    ]
}
```

In the JSON snippet:

- The **type** element of the trigger is set to "ScheduleTrigger".
- The **frequency** element is set to "Minute" and the **interval** element is set to 15. As such, the trigger runs the pipeline every 15 minutes between the start and end times.
- The **timeZone** element specifies the time zone that the trigger is created in. This setting affects both **startTime** and **endTime**.
- The **endTime** element is one hour after the value of the **startTime** element. As such, the trigger runs the pipeline 15 minutes, 30 minutes, and 45 minutes after the start time. Don't forget to update the start time to the current UTC time, and the end time to one hour past the start time.

> **IMPORTANT**
>
> For UTC timezone, the startTime and endTime need to follow format 'yyyy-MM-ddTHH:mm:ssZ', while for other timezones, startTime and endTime follow 'yyyy-MM-ddTHH:mm:ss'.
>
> Per ISO 8601 standard, the *Z* suffix to timestamp mark the datetime to UTC timezone, and render timeZone field useless. While missing *Z* suffix for UTC time zone will result in an error upon trigger *activation*.

- The trigger is associated with the **Adfv2QuickStartPipeline** pipeline. To associate multiple pipelines with a trigger, add more **pipelineReference** sections.
- The pipeline in the Quickstart takes two **parameters** values: **inputPath** and **outputPath**. And you pass values for these parameters from the trigger.

2. Create a trigger by using the az datafactory trigger create command:

```
az datafactory trigger create --resource-group "ADFQuickStartRG" --factory-name "ADFTutorialFactory"
--name "MyTrigger" --properties @MyTrigger.json
```

3. Confirm that the status of the trigger is **Stopped** by using the az datafactory trigger show command:

```
az datafactory trigger show --resource-group "ADFQuickStartRG" --factory-name "ADFTutorialFactory" --
name "MyTrigger"
```

4. Start the trigger by using the az datafactory trigger start command:

```
az datafactory trigger start --resource-group "ADFQuickStartRG" --factory-name "ADFTutorialFactory" -
-name "MyTrigger"
```

5. Confirm that the status of the trigger is **Started** by using the az datafactory trigger show command:

```
az datafactory trigger show --resource-group "ADFQuickStartRG" --factory-name "ADFTutorialFactory" --
name "MyTrigger"
```

6. Get the trigger runs in Azure CLI by using the az datafactory trigger-run query-by-factory command. To get information about the trigger runs, execute the following command periodically. Update the **last-updated-after** and **last-updated-before** values to match the values in your trigger definition:

```
az datafactory trigger-run query-by-factory --resource-group "ADFQuickStartRG" --factory-name
"ADFTutorialFactory" --filters operand="TriggerName" operator="Equals" values="MyTrigger" --last-
updated-after "2017-12-08T00:00:00" --last-updated-before "2017-12-08T01:00:00"
```

> **NOTE**
>
> Trigger time of Schedule triggers are specified in UTC timestamp. *last-updated-after* and *last-updated-before* also expects UTC timestamp

To monitor the trigger runs and pipeline runs in the Azure portal, see Monitor pipeline runs.

.NET SDK

This section shows you how to use the .NET SDK to create, start, and monitor a trigger. To see this sample working, first go through the Quickstart: Create a data factory by using the .NET SDK. Then, add the following code to the main method, which creates and starts a schedule trigger that runs every 15 minutes. The trigger is associated with a pipeline named **Adfv2QuickStartPipeline** that you create as part of the Quickstart.

To create and start a schedule trigger that runs every 15 minutes, add the following code to the main method:

```
            // Create the trigger
            Console.WriteLine("Creating the trigger");

            // Set the start time to the current UTC time
            DateTime startTime = DateTime.UtcNow;

            // Specify values for the inputPath and outputPath parameters
            Dictionary<string, object> pipelineParameters = new Dictionary<string, object>();
            pipelineParameters.Add("inputPath", "adftutorial/input");
            pipelineParameters.Add("outputPath", "adftutorial/output");

            // Create a schedule trigger
            string triggerName = "MyTrigger";
            ScheduleTrigger myTrigger = new ScheduleTrigger()
            {
                Pipelines = new List<TriggerPipelineReference>()
                {
                    // Associate the Adfv2QuickStartPipeline pipeline with the trigger
                    new TriggerPipelineReference()
                    {
                        PipelineReference = new PipelineReference(pipelineName),
                        Parameters = pipelineParameters,
                    }
                },
                Recurrence = new ScheduleTriggerRecurrence()
                {
                    // Set the start time to the current UTC time and the end time to one hour after the
start time
                    StartTime = startTime,
                    TimeZone = "UTC",
                    EndTime = startTime.AddHours(1),
                    Frequency = RecurrenceFrequency.Minute,
                    Interval = 15,
                }
            };

            // Now, create the trigger by invoking the CreateOrUpdate method
            TriggerResource triggerResource = new TriggerResource()
            {
                Properties = myTrigger
            };
            client.Triggers.CreateOrUpdate(resourceGroup, dataFactoryName, triggerName, triggerResource);

            // Start the trigger
            Console.WriteLine("Starting the trigger");
            client.Triggers.Start(resourceGroup, dataFactoryName, triggerName);
```

To create triggers in a different time zone, other than UTC, following settings are required:

```
<<ClientInstance>>.SerializationSettings.DateFormatHandling =
Newtonsoft.Json.DateFormatHandling.IsoDateFormat;
<<ClientInstance>>.SerializationSettings.DateTimeZoneHandling =
Newtonsoft.Json.DateTimeZoneHandling.Unspecified;
<<ClientInstance>>.SerializationSettings.DateParseHandling = DateParseHandling.None;
<<ClientInstance>>.DeserializationSettings.DateParseHandling = DateParseHandling.None;
<<ClientInstance>>.DeserializationSettings.DateFormatHandling =
Newtonsoft.Json.DateFormatHandling.IsoDateFormat;
<<ClientInstance>>.DeserializationSettings.DateTimeZoneHandling =
Newtonsoft.Json.DateTimeZoneHandling.Unspecified;
```

To monitor a trigger run, add the following code before the last `Console.WriteLine` statement in the sample:

```
            // Check that the trigger runs every 15 minutes
            Console.WriteLine("Trigger runs. You see the output every 15 minutes");

            for (int i = 0; i < 3; i++)
            {
                System.Threading.Thread.Sleep(TimeSpan.FromMinutes(15));
                List<TriggerRun> triggerRuns = client.Triggers.ListRuns(resourceGroup, dataFactoryName,
triggerName, DateTime.UtcNow.AddMinutes(-15 * (i + 1)), DateTime.UtcNow.AddMinutes(2)).ToList();
                Console.WriteLine("{0} trigger runs found", triggerRuns.Count);

                foreach (TriggerRun run in triggerRuns)
                {
                    foreach (KeyValuePair<string, string> triggeredPipeline in run.TriggeredPipelines)
                    {
                        PipelineRun triggeredPipelineRun = client.PipelineRuns.Get(resourceGroup,
dataFactoryName, triggeredPipeline.Value);
                        Console.WriteLine("Pipeline run ID: {0}, Status: {1}", triggeredPipelineRun.RunId,
triggeredPipelineRun.Status);
                        List<ActivityRun> runs = client.ActivityRuns.ListByPipelineRun(resourceGroup,
dataFactoryName, triggeredPipelineRun.RunId, run.TriggerRunTimestamp.Value,
run.TriggerRunTimestamp.Value.AddMinutes(20)).ToList();
                    }
                }
            }
```

To monitor the trigger runs and pipeline runs in the Azure portal, see Monitor pipeline runs.

Python SDK

This section shows you how to use the Python SDK to create, start, and monitor a trigger. To see this sample working, first go through the Quickstart: Create a data factory by using the Python SDK. Then, add the following code block after the "monitor the pipeline run" code block in the Python script. This code creates a schedule trigger that runs every 15 minutes between the specified start and end times. Update the **start_time** variable to the current UTC time, and the **end_time** variable to one hour past the current UTC time.

```
    # Create a trigger
    tr_name = 'mytrigger'
    scheduler_recurrence = ScheduleTriggerRecurrence(frequency='Minute', interval='15',start_time='2017-12-
12T04:00:00Z', end_time='2017-12-12T05:00:00Z', time_zone='UTC')
    pipeline_parameters = {'inputPath':'adftutorial/input', 'outputPath':'adftutorial/output'}
    pipelines_to_run = []
    pipeline_reference = PipelineReference('copyPipeline')
    pipelines_to_run.append(TriggerPipelineReference(pipeline_reference, pipeline_parameters))
    tr_properties = TriggerResource(properties=ScheduleTrigger(description='My scheduler trigger', pipelines
= pipelines_to_run, recurrence=scheduler_recurrence))
    adf_client.triggers.create_or_update(rg_name, df_name, tr_name, tr_properties)

    # Start the trigger
    adf_client.triggers.start(rg_name, df_name, tr_name)
```

To monitor the trigger runs and pipeline runs in the Azure portal, see Monitor pipeline runs.

Azure Resource Manager template

You can use an Azure Resource Manager template to create a trigger. For step-by-step instructions, see Create an Azure data factory by using a Resource Manager template.

Pass the trigger start time to a pipeline

Azure Data Factory version 1 supports reading or writing partitioned data by using the system variables: **SliceStart**, **SliceEnd**, **WindowStart**, and **WindowEnd**. In the current version of Azure Data Factory and Synapse pipelines, you can achieve this behavior by using a pipeline parameter. The start time and scheduled time for the trigger are set as the value for the pipeline parameter. In the following example, the scheduled time for the trigger is passed as a value to the pipeline **scheduledRunTime** parameter:

```
"parameters": {
    "scheduledRunTime": "@trigger().scheduledTime"
}
```

JSON schema

The following JSON definition shows you how to create a schedule trigger with scheduling and recurrence:

```
{
  "properties": {
    "type": "ScheduleTrigger",
    "typeProperties": {
      "recurrence": {
        "frequency": <<Minute, Hour, Day, Week, Month>>,
        "interval": <<int>>,             // Optional, specifies how often to fire (default to 1)
        "startTime": <<datetime>>,
        "endTime": <<datetime - optional>>,
        "timeZone": "UTC"
        "schedule": {                    // Optional (advanced scheduling specifics)
          "hours": [<<0-23>>],
          "weekDays": [<<Monday-Sunday>>],
          "minutes": [<<0-59>>],
          "monthDays": [<<1-31>>],
          "monthlyOccurrences": [
               {
                    "day": <<Monday-Sunday>>,
                    "occurrence": <<1-5>>
               }
           ]
        }
      }
    },
   "pipelines": [
           {
               "pipelineReference": {
                   "type": "PipelineReference",
                   "referenceName": "<Name of your pipeline>"
               },
               "parameters": {
                   "<parameter 1 Name>": {
                       "type": "Expression",
                       "value": "<parameter 1 Value>"
                   },
                   "<parameter 2 Name>" : "<parameter 2 Value>"
               }
           }
      ]
  }
}
```

IMPORTANT

The **parameters** property is a mandatory property of the **pipelines** element. If your pipeline doesn't take any parameters, you must include an empty JSON definition for the **parameters** property.

Schema overview

The following table provides a high-level overview of the major schema elements that are related to recurrence and scheduling of a trigger:

JSON PROPERTY	DESCRIPTION
startTime	A Date-Time value. For simple schedules, the value of the **startTime** property applies to the first occurrence. For complex schedules, the trigger starts no sooner than the specified **startTime** value. For UTC time zone, format is `'yyyy-MM-ddTHH:mm:ssZ'`, for other time zone, format is `'yyyy-MM-ddTHH:mm:ss'`.

JSON PROPERTY	DESCRIPTION
endTime	The end date and time for the trigger. The trigger doesn't execute after the specified end date and time. The value for the property can't be in the past. This property is optional. For UTC time zone, format is `'yyyy-MM-ddTHH:mm:ssZ'`, for other time zone, format is `'yyyy-MM-ddTHH:mm:ss'`.
timeZone	The time zone the trigger is created in. This setting affects **startTime**, **endTime**, and **schedule**. See list of supported time zone
recurrence	A recurrence object that specifies the recurrence rules for the trigger. The recurrence object supports the **frequency**, **interval**, **endTime**, **count**, and **schedule** elements. When a recurrence object is defined, the **frequency** element is required. The other elements of the recurrence object are optional.
frequency	The unit of frequency at which the trigger recurs. The supported values include "minute," "hour," "day," "week," and "month."
interval	A positive integer that denotes the interval for the **frequency** value, which determines how often the trigger runs. For example, if the **interval** is 3 and the **frequency** is "week," the trigger recurs every 3 weeks.
schedule	The recurrence schedule for the trigger. A trigger with a specified **frequency** value alters its recurrence based on a recurrence schedule. The **schedule** property contains modifications for the recurrence that are based on minutes, hours, weekdays, month days, and week number.

IMPORTANT

For UTC timezone, the startTime and endTime need to follow format 'yyyy-MM-ddTHH:mm:ssZ', while for other timezones, startTime and endTime follow 'yyyy-MM-ddTHH:mm:ss'.

Per ISO 8601 standard, the *Z* suffix to timestamp mark the datetime to UTC timezone, and render timeZone field useless. While missing *Z* suffix for UTC time zone will result in an error upon trigger *activation*.

Schema defaults, limits, and examples

JSON PROPERTY	TYPE	REQUIRED	DEFAULT VALUE	VALID VALUES	EXAMPLE
startTime	String	Yes	None	ISO-8601 Date-Times	for UTC time zone `"startTime" : "2013-01-09T09:30:00-08:00Z"` for other time zone `"2013-01-09T09:30:00-08:00"`

JSON PROPERTY	TYPE	REQUIRED	DEFAULT VALUE	VALID VALUES	EXAMPLE
timeZone	String	Yes	None	Time Zone Values	`"UTC"`
recurrence	Object	Yes	None	Recurrence object	`"recurrence" : { "frequency" : "monthly", "interval" : 1 }`
interval	Number	No	1	1 to 1,000	`"interval":10`
endTime	String	Yes	None	A Date-Time value that represents a time in the future.	for UTC time zone `"endTime" : "2013-02-09T09:30:00-08:00Z"` for other time zone `"endTime" : "2013-02-09T09:30:00-08:00"`
schedule	Object	No	None	Schedule object	`"schedule" : { "minute" : [30], "hour" : [8,17] }`

Time zone option

Here are some of time zones supported for Schedule triggers:

TIME ZONE	UTC OFFSET (NON-DAYLIGHT SAVING)	TIMEZONE VALUE	OBSERVE DAYLIGHT SAVING	TIME STAMP FORMAT
Coordinated Universal Time	0	`UTC`	No	`'yyyy-MM-ddTHH:mm:ssZ'`
Pacific Time (PT)	-8	`Pacific Standard Time`	Yes	`'yyyy-MM-ddTHH:mm:ss'`
Central Time (CT)	-6	`Central Standard Time`	Yes	`'yyyy-MM-ddTHH:mm:ss'`
Eastern Time (ET)	-5	`Eastern Standard Time`	Yes	`'yyyy-MM-ddTHH:mm:ss'`
Greenwich Mean Time (GMT)	0	`GMT Standard Time`	Yes	`'yyyy-MM-ddTHH:mm:ss'`
Central European Standard Time	+1	`W. Europe Standard Time`	Yes	`'yyyy-MM-ddTHH:mm:ss'`
India Standard Time (IST)	+5:30	`India Standard Time`	No	`'yyyy-MM-ddTHH:mm:ss'`

TIME ZONE	UTC OFFSET (NON-DAYLIGHT SAVING)	TIMEZONE VALUE	OBSERVE DAYLIGHT SAVING	TIME STAMP FORMAT
China Standard Time	+8	`China Standard Time`	No	`'yyyy-MM-ddTHH:mm:ss'`

This list is incomplete. For complete list of time zone options, explore in the portal Trigger creation page

startTime property

The following table shows you how the **startTime** property controls a trigger run:

STARTTIME VALUE	RECURRENCE WITHOUT SCHEDULE	RECURRENCE WITH SCHEDULE
Start time in past	Calculates the first future execution time after the start time and runs at that time. Runs subsequent executions based on calculating from the last execution time. See the example that follows this table.	The trigger starts *no sooner than* the specified start time. The first occurrence is based on the schedule that's calculated from the start time. Runs subsequent executions based on the recurrence schedule.
Start time in future or at present	Runs once at the specified start time. Runs subsequent executions based on calculating from the last execution time.	The trigger starts *no sooner* than the specified start time. The first occurrence is based on the schedule that's calculated from the start time. Runs subsequent executions based on the recurrence schedule.

Let's see an example of what happens when the start time is in the past, with a recurrence, but no schedule. Assume that the current time is `2017-04-08 13:00`, the start time is `2017-04-07 14:00`, and the recurrence is every two days. (The **recurrence** value is defined by setting the **frequency** property to "day" and the **interval** property to 2.) Notice that the **startTime** value is in the past and occurs before the current time.

Under these conditions, the first execution is at `2017-04-09` at `14:00`. The Scheduler engine calculates execution occurrences from the start time. Any instances in the past are discarded. The engine uses the next instance that occurs in the future. In this scenario, the start time is `2017-04-07` at `2:00pm`, so the next instance is two days from that time, which is `2017-04-09` at `2:00pm`.

The first execution time is the same even if the **startTime** value is `2017-04-05 14:00` or `2017-04-01 14:00`. After the first execution, subsequent executions are calculated by using the schedule. Therefore, the subsequent executions are at `2017-04-11` at `2:00pm`, then `2017-04-13` at `2:00pm`, then `2017-04-15` at `2:00pm`, and so on.

Finally, when the hours or minutes aren't set in the schedule for a trigger, the hours or minutes of the first execution are used as the defaults.

schedule property

On one hand, the use of a schedule can limit the number of trigger executions. For example, if a trigger with a monthly frequency is scheduled to run only on day 31, the trigger runs only in those months that have a 31st day.

Whereas, a schedule can also expand the number of trigger executions. For example, a trigger with a monthly frequency that's scheduled to run on month days 1 and 2, runs on the 1st and 2nd days of the month, rather than once a month.

If multiple **schedule** elements are specified, the order of evaluation is from the largest to the smallest schedule

setting. The evaluation starts with week number, and then month day, weekday, hour, and finally, minute.

The following table describes the **schedule** elements in detail:

JSON ELEMENT	DESCRIPTION	VALID VALUES
minutes	Minutes of the hour at which the trigger runs.	• Integer • Array of integers
hours	Hours of the day at which the trigger runs.	• Integer • Array of integers
weekDays	Days of the week on which the trigger runs. The value can be specified with a weekly frequency only.	• Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday • Array of day values (maximum array size is 7) • Day values are not case-sensitive
monthlyOccurrences	Days of the month on which the trigger runs. The value can be specified with a monthly frequency only.	• Array of **monthlyOccurrence** objects: `{ "day": day, "occurrence": occurrence }`. • The **day** attribute is the day of the week on which the trigger runs. For example, a **monthlyOccurrences** property with a **day** value of `{Sunday}` means every Sunday of the month. The **day** attribute is required. • The **occurrence** attribute is the occurrence of the specified **day** during the month. For example, a **monthlyOccurrences** property with **day** and **occurrence** values of `{Sunday, -1}` means the last Sunday of the month. The **occurrence** attribute is optional.
monthDays	Day of the month on which the trigger runs. The value can be specified with a monthly frequency only.	• Any value <= -1 and >= -31 • Any value >= 1 and <= 31 • Array of values

Examples of trigger recurrence schedules

This section provides examples of recurrence schedules and focuses on the **schedule** object and its elements.

The examples assume that the **interval** value is 1, and that the **frequency** value is correct according to the schedule definition. For example, you can't have a **frequency** value of "day" and also have a "monthDays" modification in the **schedule** object. Restrictions such as these are mentioned in the table in the previous

section.

EXAMPLE	DESCRIPTION
`{"hours":[5]}`	Run at 5:00 AM every day.
`{"minutes":[15], "hours":[5]}`	Run at 5:15 AM every day.
`{"minutes":[15], "hours":[5,17]}`	Run at 5:15 AM and 5:15 PM every day.
`{"minutes":[15,45], "hours":[5,17]}`	Run at 5:15 AM, 5:45 AM, 5:15 PM, and 5:45 PM every day.
`{"minutes":[0,15,30,45]}`	Run every 15 minutes.
`{hours":[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]}`	Run every hour. This trigger runs every hour. The minutes are controlled by the **startTime** value, when a value is specified. If a value not specified, the minutes are controlled by the creation time. For example, if the start time or creation time (whichever applies) is 12:25 PM, the trigger runs at 00:25, 01:25, 02:25, ..., and 23:25. This schedule is equivalent to having a trigger with a **frequency** value of "hour," an **interval** value of 1, and no **schedule**. This schedule can be used with different **frequency** and **interval** values to create other triggers. For example, when the **frequency** value is "month," the schedule runs only once a month, rather than every day, when the **frequency** value is "day."
`{"minutes":[0]}`	Run every hour on the hour. This trigger runs every hour on the hour starting at 12:00 AM, 1:00 AM, 2:00 AM, and so on. This schedule is equivalent to a trigger with a **frequency** value of "hour" and a **startTime** value of zero minutes, or no **schedule** but a **frequency** value of "day." If the **frequency** value is "week" or "month," the schedule executes one day a week or one day a month only, respectively.
`{"minutes":[15]}`	Run at 15 minutes past every hour. This trigger runs every hour at 15 minutes past the hour starting at 00:15 AM, 1:15 AM, 2:15 AM, and so on, and ending at 11:15 PM.
`{"hours":[17], "weekDays":["saturday"]}`	Run at 5:00 PM on Saturdays every week.
`{"hours":[17], "weekDays":["monday", "wednesday", "friday"]}`	Run at 5:00 PM on Monday, Wednesday, and Friday every week.
`{"minutes":[15,45], "hours":[17], "weekDays": ["monday", "wednesday", "friday"]}`	Run at 5:15 PM and 5:45 PM on Monday, Wednesday, and Friday every week.
`{"minutes":[0,15,30,45], "weekDays":["monday", "tuesday", "wednesday", "thursday", "friday"]}`	Run every 15 minutes on weekdays.
`{"minutes":[0,15,30,45], "hours": [9, 10, 11, 12, 13, 14, 15, 16] "weekDays":["monday", "tuesday", "wednesday", "thursday", "friday"]}`	Run every 15 minutes on weekdays between 9:00 AM and 4:45 PM.

EXAMPLE	DESCRIPTION
`{"weekDays":["tuesday", "thursday"]}`	Run on Tuesdays and Thursdays at the specified start time.
`{"minutes":[0], "hours":[6], "monthDays":[28]}`	Run at 6:00 AM on the 28th day of every month (assuming a **frequency** value of "month").
`{"minutes":[0], "hours":[6], "monthDays":[-1]}`	Run at 6:00 AM on the last day of the month. To run a trigger on the last day of a month, use -1 instead of day 28, 29, 30, or 31.
`{"minutes":[0], "hours":[6], "monthDays":[1,-1]}`	Run at 6:00 AM on the first and last day of every month.
`{monthDays":[1,14]}`	Run on the first and 14th day of every month at the specified start time.
`{"minutes":[0], "hours":[5], "monthlyOccurrences": [{"day":"friday", "occurrence":1}]}`	Run on the first Friday of every month at 5:00 AM.
`{"monthlyOccurrences":[{"day":"friday", "occurrence":1}]}`	Run on the first Friday of every month at the specified start time.
`{"monthlyOccurrences":[{"day":"friday", "occurrence":-3}]}`	Run on the third Friday from the end of the month, every month, at the specified start time.
`{"minutes":[15], "hours":[5], "monthlyOccurrences": [{"day":"friday", "occurrence":1},{"day":"friday", "occurrence":-1}]}`	Run on the first and last Friday of every month at 5:15 AM.
`{"monthlyOccurrences":[{"day":"friday", "occurrence":1},{"day":"friday", "occurrence":-1}]}`	Run on the first and last Friday of every month at the specified start time.
`{"monthlyOccurrences":[{"day":"friday", "occurrence":5}]}`	Run on the fifth Friday of every month at the specified start time. When there's no fifth Friday in a month, the pipeline doesn't run, since it's scheduled to run only on fifth Fridays. To run the trigger on the last occurring Friday of the month, consider using -1 instead of 5 for the **occurrence** value.
`{"minutes":[0,15,30,45], "monthlyOccurrences": [{"day":"friday", "occurrence":-1}]}`	Run every 15 minutes on the last Friday of the month.
`{"minutes":[15,45], "hours":[5,17], "monthlyOccurrences":[{"day":"wednesday", "occurrence":3}]}`	Run at 5:15 AM, 5:45 AM, 5:15 PM, and 5:45 PM on the third Wednesday of every month.

Next steps

- For detailed information about triggers, see Pipeline execution and triggers.
- Learn how to reference trigger metadata in pipeline, see Reference Trigger Metadata in Pipeline Runs

Create a trigger that runs a pipeline on a tumbling window

8/19/2022 • 10 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

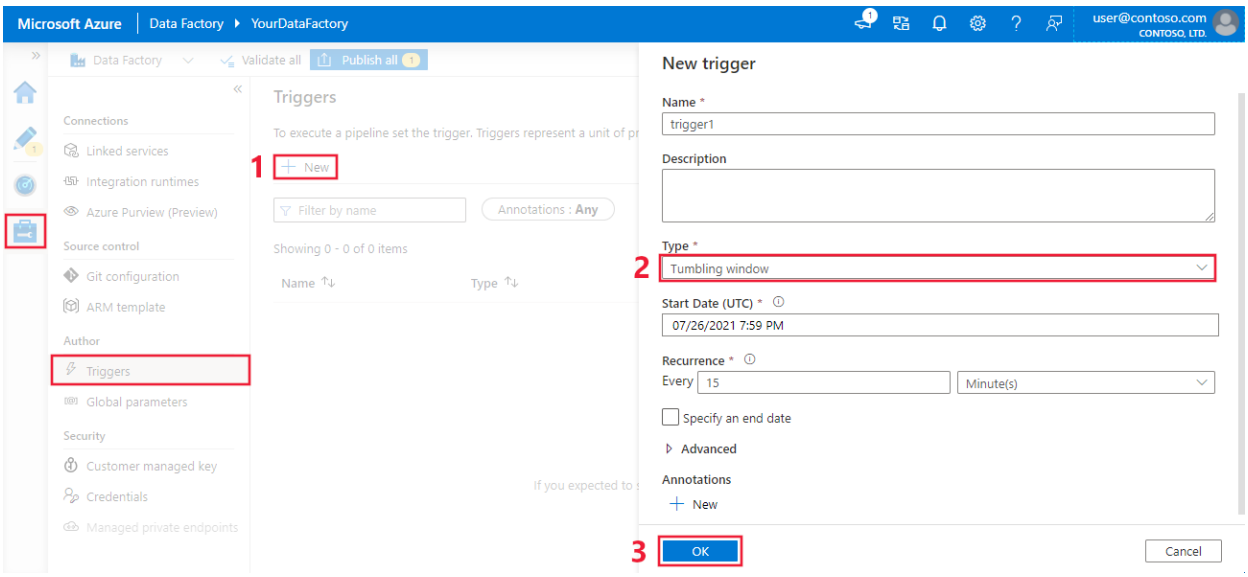
This article provides steps to create, start, and monitor a tumbling window trigger. For general information about triggers and the supported types, see Pipeline execution and triggers.

Tumbling window triggers are a type of trigger that fires at a periodic time interval from a specified start time, while retaining state. Tumbling windows are a series of fixed-sized, non-overlapping, and contiguous time intervals. A tumbling window trigger has a one-to-one relationship with a pipeline and can only reference a singular pipeline. Tumbling window trigger is a more heavy weight alternative for schedule trigger offering a suite of features for complex scenarios(dependency on other tumbling window triggers, rerunning a failed job and set user retry for pipelines). To further understand the difference between schedule trigger and tumbling window trigger, please visit here.

Azure Data Factory and Synapse portal experience

1. To create a tumbling window trigger in the Azure portal, select the **Triggers** tab, and then select **New**.
2. After the trigger configuration pane opens, select **Tumbling Window**, and then define your tumbling window trigger properties.
3. When you're done, select **Save**.

- Azure Data Factory
- Azure Synapse



Tumbling window trigger type properties

A tumbling window has the following trigger type properties:

```
{
    "name": "MyTriggerName",
    "properties": {
        "type": "TumblingWindowTrigger",
        "runtimeState": "<<Started/Stopped/Disabled - readonly>>",
        "typeProperties": {
            "frequency": <<Minute/Hour>>,
            "interval": <<int>>,
            "startTime": "<<datetime>>",
            "endTime": <<datetime – optional>>,
            "delay": <<timespan – optional>>,
            "maxConcurrency": <<int>> (required, max allowed: 50),
            "retryPolicy": {
                "count": <<int - optional, default: 0>>,
                "intervalInSeconds": <<int>>,
            },
            "dependsOn": [
                {
                    "type": "TumblingWindowTriggerDependencyReference",
                    "size": <<timespan – optional>>,
                    "offset": <<timespan – optional>>,
                    "referenceTrigger": {
                        "referenceName": "MyTumblingWindowDependency1",
                        "type": "TriggerReference"
                    }
                },
                {
                    "type": "SelfDependencyTumblingWindowTriggerReference",
                    "size": <<timespan – optional>>,
                    "offset": <<timespan>>
                }
            ]
        },
        "pipeline": {
            "pipelineReference": {
                "type": "PipelineReference",
                "referenceName": "MyPipelineName"
            },
            "parameters": {
                "parameter1": {
                    "type": "Expression",
                    "value": "@{concat('output',formatDateTime(trigger().outputs.windowStartTime,'-dd-MM-
yyyy-HH-mm-ss-ffff'))}"
                },
                "parameter2": {
                    "type": "Expression",
                    "value": "@{concat('output',formatDateTime(trigger().outputs.windowEndTime,'-dd-MM-yyyy-
HH-mm-ss-ffff'))}"
                },
                "parameter3": "https://mydemo.azurewebsites.net/api/demoapi"
            }
        }
    }
}
```

The following table provides a high-level overview of the major JSON elements that are related to recurrence and scheduling of a tumbling window trigger:

JSON ELEMENT	DESCRIPTION	TYPE	ALLOWED VALUES	REQUIRED

JSON ELEMENT	DESCRIPTION	TYPE	ALLOWED VALUES	REQUIRED
type	The type of the trigger. The type is the fixed value "TumblingWindowTrigger".	String	"TumblingWindowTrigger"	Yes
runtimeState	The current state of the trigger run time. **Note**: This element is <readOnly>.	String	"Started," "Stopped," "Disabled"	Yes
frequency	A string that represents the frequency unit (minutes or hours) at which the trigger recurs. If the **startTime** date values are more granular than the **frequency** value, the **startTime** dates are considered when the window boundaries are computed. For example, if the **frequency** value is hourly and the **startTime** value is 2017-09-01T10:10:10Z, the first window is (2017-09-01T10:10:10Z, 2017-09-01T11:10:10Z).	String	"minute," "hour"	Yes
interval	A positive integer that denotes the interval for the **frequency** value, which determines how often the trigger runs. For example, if the **interval** is 3 and the **frequency** is "hour," the trigger recurs every 3 hours. **Note**: The minimum window interval is 5 minutes.	Integer	A positive integer.	Yes
startTime	The first occurrence, which can be in the past. The first trigger interval is (**startTime**, **startTime** + **interval**).	DateTime	A DateTime value.	Yes

JSON ELEMENT	DESCRIPTION	TYPE	ALLOWED VALUES	REQUIRED
endTime	The last occurrence, which can be in the past.	DateTime	A DateTime value.	Yes
delay	The amount of time to delay the start of data processing for the window. The pipeline run is started after the expected execution time plus the amount of **delay**. The **delay** defines how long the trigger waits past the due time before triggering a new run. The **delay** doesn't alter the window **startTime**. For example, a **delay** value of 00:10:00 implies a delay of 10 minutes.	Timespan (hh:mm:ss)	A timespan value where the default is 00:00:00.	No
maxConcurrency	The number of simultaneous trigger runs that are fired for windows that are ready. For example, to back fill hourly runs for yesterday results in 24 windows. If **maxConcurrency** = 10, trigger events are fired only for the first 10 windows (00:00-01:00 - 09:00-10:00). After the first 10 triggered pipeline runs are complete, trigger runs are fired for the next 10 windows (10:00-11:00 - 19:00-20:00). Continuing with this example of **maxConcurrency** = 10, if there are 10 windows ready, there are 10 total pipeline runs. If there's only 1 window ready, there's only 1 pipeline run.	Integer	An integer between 1 and 50.	Yes
retryPolicy: Count	The number of retries before the pipeline run is marked as "Failed."	Integer	An integer, where the default is 0 (no retries).	No

JSON ELEMENT	DESCRIPTION	TYPE	ALLOWED VALUES	REQUIRED
retryPolicy: intervalInSeconds	The delay between retry attempts specified in seconds.	Integer	The number of seconds, where the default is 30.	No
dependsOn: type	The type of TumblingWindowTriggerReference. Required if a dependency is set.	String	"TumblingWindowTriggerDependencyReference", "SelfDependencyTumblingWindowTriggerReference"	No
dependsOn: size	The size of the dependency tumbling window.	Timespan (hh:mm:ss)	A positive timespan value where the default is the window size of the child trigger	No
dependsOn: offset	The offset of the dependency trigger.	Timespan (hh:mm:ss)	A timespan value that must be negative in a self-dependency. If no value specified, the window is the same as the trigger itself.	Self-Dependency: Yes Other: No

> **NOTE**
> After a tumbling window trigger is published, **interval** and **frequency** can't be edited.

WindowStart and WindowEnd system variables

You can use the **WindowStart** and **WindowEnd** system variables of the tumbling window trigger in your **pipeline** definition (that is, for part of a query). Pass the system variables as parameters to your pipeline in the **trigger** definition. The following example shows you how to pass these variables as parameters:

```
{
    "name": "MyTriggerName",
    "properties": {
        "type": "TumblingWindowTrigger",
            ...
        "pipeline": {
            "pipelineReference": {
                "type": "PipelineReference",
                "referenceName": "MyPipelineName"
            },
            "parameters": {
                "MyWindowStart": {
                    "type": "Expression",
                    "value": "@{concat('output',formatDateTime(trigger().outputs.windowStartTime,'-dd-MM-
yyyy-HH-mm-ss-ffff'))}"
                },
                "MyWindowEnd": {
                    "type": "Expression",
                    "value": "@{concat('output',formatDateTime(trigger().outputs.windowEndTime,'-dd-MM-yyyy-
HH-mm-ss-ffff'))}"
                }
            }
        }
    }
}
```

To use the **WindowStart** and **WindowEnd** system variable values in the pipeline definition, use your "MyWindowStart" and "MyWindowEnd" parameters, accordingly.

Execution order of windows in a backfill scenario

If the startTime of trigger is in the past, then based on this formula, M=(CurrentTime-TriggerStartTime)/TumblingWindowSize, the trigger will generate {M} backfill(past) runs in parallel, honoring trigger concurrency, before executing the future runs. The order of execution for windows is deterministic, from oldest to newest intervals. Currently, this behavior can't be modified.

Existing TriggerResource elements

The following points apply to update of existing **TriggerResource** elements:

- The value for the **frequency** element (or window size) of the trigger along with **interval** element cannot be changed once the trigger is created. This is required for proper functioning of triggerRun reruns and dependency evaluations
- If the value for the **endTime** element of the trigger changes (added or updated), the state of the windows that are already processed is *not* reset. The trigger honors the new **endTime** value. If the new **endTime** value is before the windows that are already executed, the trigger stops. Otherwise, the trigger stops when the new **endTime** value is encountered.

User assigned retries of pipelines

In case of pipeline failures, tumbling window trigger can retry the execution of the referenced pipeline automatically, using the same input parameters, without the user intervention. This can be specified using the property "retryPolicy" in the trigger definition.

Tumbling window trigger dependency

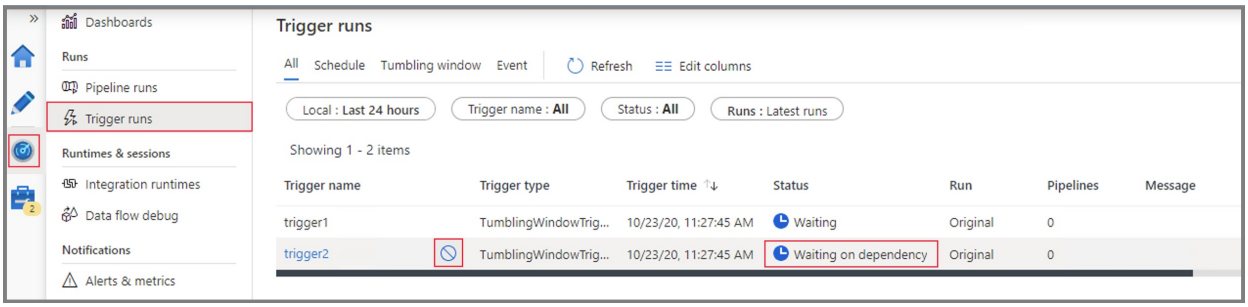
If you want to make sure that a tumbling window trigger is executed only after the successful execution of another tumbling window trigger in the data factory, create a tumbling window trigger dependency.

Cancel tumbling window run

You can cancel runs for a tumbling window trigger, if the specific window is in *Waiting*, *Waiting on Dependency*, or *Running* state

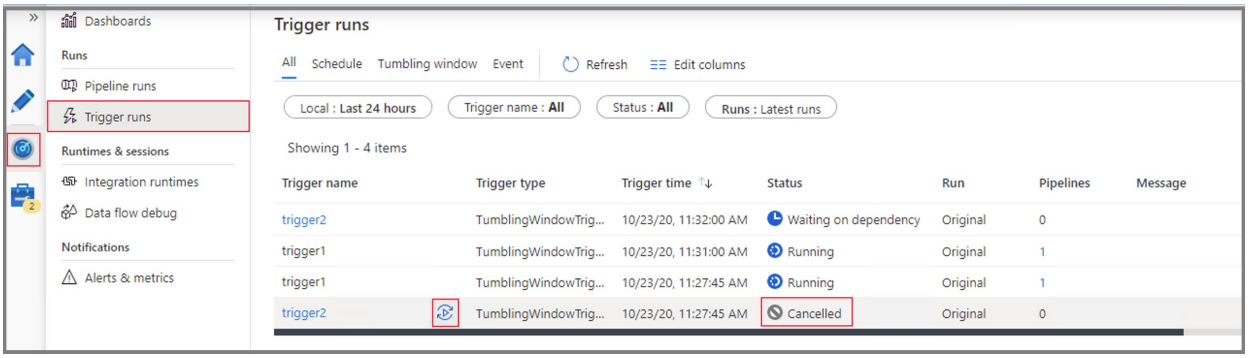
- If the window is in **Running** state, cancel the associated *Pipeline Run*, and the trigger run will be marked as *Canceled* afterwards
- If the window is in **Waiting** or **Waiting on Dependency** state, you can cancel the window from Monitoring:

- Azure Data Factory
- Azure Synapse



You can also rerun a canceled window. The rerun will take the *latest* published definitions of the trigger, and dependencies for the specified window will be *re-evaluated* upon rerun

- Azure Data Factory
- Azure Synapse



Sample for Azure PowerShell and Azure CLI

- Azure PowerShell
- Azure CLI

This section shows you how to use Azure PowerShell to create, start, and monitor a trigger.

> **NOTE**
> This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see Install Azure PowerShell. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Prerequisites

- **Azure subscription**. If you don't have an Azure subscription, create a free account before you begin.
- **Azure PowerShell**. Follow the instructions in Install Azure PowerShell on Windows with PowerShellGet.
- **Azure Data Factory**. Follow the instructions in Create an Azure Data Factory using PowerShell to create a data factory and a pipeline.

Sample Code

1. Create a JSON file named **MyTrigger.json** in the C:\ADFv2QuickStartPSH\ folder with the following content:

   > **IMPORTANT**
   >
   > Before you save the JSON file, set the value of the **startTime** element to the current UTC time. Set the value of the **endTime** element to one hour past the current UTC time.

   ```
   {
     "name": "PerfTWTrigger",
     "properties": {
       "type": "TumblingWindowTrigger",
       "typeProperties": {
         "frequency": "Minute",
         "interval": "15",
         "startTime": "2017-09-08T05:30:00Z",
         "endTime" : "2017-09-08T06:30:00Z",
         "delay": "00:00:01",
         "retryPolicy": {
           "count": 2,
           "intervalInSeconds": 30
         },
         "maxConcurrency": 50
       },
       "pipeline": {
         "pipelineReference": {
           "type": "PipelineReference",
           "referenceName": "DynamicsToBlobPerfPipeline"
         },
         "parameters": {
           "windowStart": "@trigger().outputs.windowStartTime",
           "windowEnd": "@trigger().outputs.windowEndTime"
         }
       },
       "runtimeState": "Started"
     }
   }
   ```

2. Create a trigger by using the Set-AzDataFactoryV2Trigger cmdlet:

   ```
   Set-AzDataFactoryV2Trigger -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -
   Name "MyTrigger" -DefinitionFile "C:\ADFv2QuickStartPSH\MyTrigger.json"
   ```

3. Confirm that the status of the trigger is **Stopped** by using the Get-AzDataFactoryV2Trigger cmdlet:

   ```
   Get-AzDataFactoryV2Trigger -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -
   Name "MyTrigger"
   ```

4. Start the trigger by using the Start-AzDataFactoryV2Trigger cmdlet:

   ```
   Start-AzDataFactoryV2Trigger -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName
   -Name "MyTrigger"
   ```

5. Confirm that the status of the trigger is **Started** by using the Get-AzDataFactoryV2Trigger cmdlet:

```
Get-AzDataFactoryV2Trigger -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -
Name "MyTrigger"
```

6. Get the trigger runs in Azure PowerShell by using the Get-AzDataFactoryV2TriggerRun cmdlet. To get information about the trigger runs, execute the following command periodically. Update the **TriggerRunStartedAfter** and **TriggerRunStartedBefore** values to match the values in your trigger definition:

```
Get-AzDataFactoryV2TriggerRun -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName
-TriggerName "MyTrigger" -TriggerRunStartedAfter "2017-12-08T00:00:00" -TriggerRunStartedBefore
"2017-12-08T01:00:00"
```

To monitor trigger runs and pipeline runs in the Azure portal, see Monitor pipeline runs.

Next steps

- For detailed information about triggers, see Pipeline execution and triggers.
- Create a tumbling window trigger dependency.
- Learn how to reference trigger metadata in pipeline, see Reference Trigger Metadata in Pipeline Runs

Create a tumbling window trigger dependency

8/19/2022 • 4 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

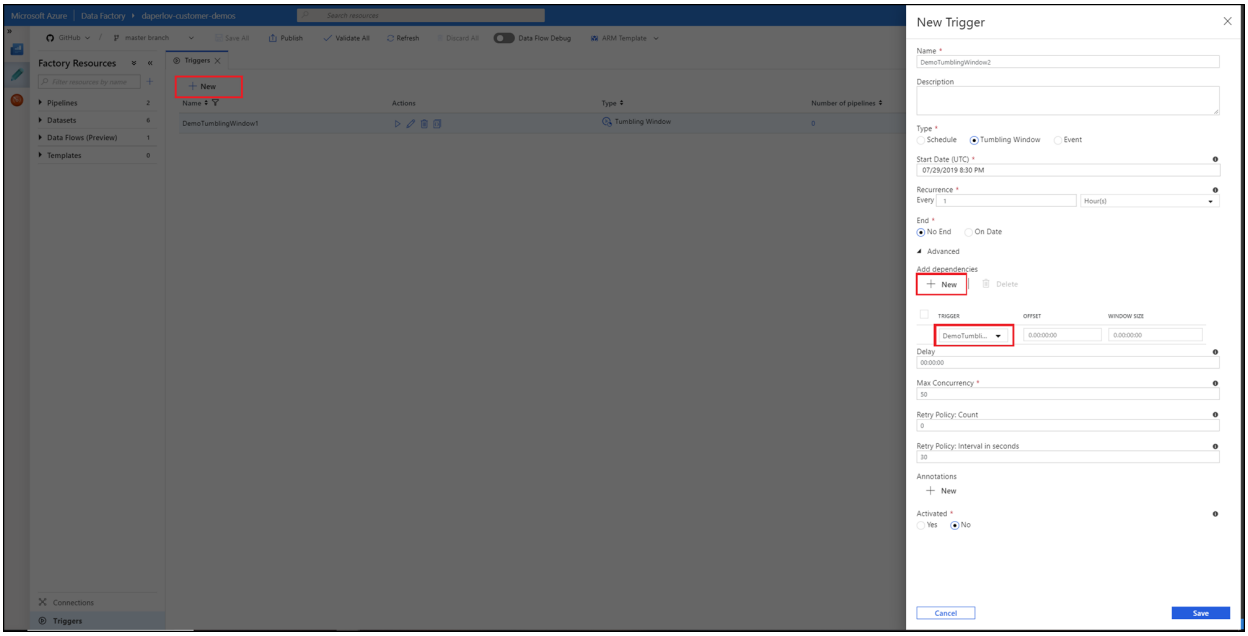
This article provides steps to create a dependency on a tumbling window trigger. For general information about Tumbling Window triggers, see How to create tumbling window trigger.

In order to build a dependency chain and make sure that a trigger is executed only after the successful execution of another trigger within the service, use this advanced feature to create a tumbling window dependency.

For a demonstration on how to create dependent pipelines using tumbling window trigger, watch the following video:

Create a dependency in the UI

To create dependency on a trigger, select **Trigger** > **Advanced** > **New**, and then choose the trigger to depend on with the appropriate offset and size. Select **Finish** and publish the changes for the dependencies to take effect.



Tumbling window dependency properties

A tumbling window trigger with a dependency has the following properties:

```
{
    "name": "MyTriggerName",
    "properties": {
        "type": "TumblingWindowTrigger",
        "runtimeState": <<Started/Stopped/Disabled - readonly>>,
        "typeProperties": {
            "frequency": <<Minute/Hour>>,
            "interval": <<int>>,
            "startTime": <<datetime>>,
            "endTime": <<datetime – optional>>,
            "delay": <<timespan – optional>>,
            "maxConcurrency": <<int>> (required, max allowed: 50),
            "retryPolicy": {
                "count": <<int - optional, default: 0>>,
                "intervalInSeconds": <<int>>,
            },
            "dependsOn": [
                {
                    "type": "TumblingWindowTriggerDependencyReference",
                    "size": <<timespan – optional>>,
                    "offset": <<timespan – optional>>,
                    "referenceTrigger": {
                        "referenceName": "MyTumblingWindowDependency1",
                        "type": "TriggerReference"
                    }
                },
                {
                    "type": "SelfDependencyTumblingWindowTriggerReference",
                    "size": <<timespan – optional>>,
                    "offset": <<timespan>>
                }
            ]
        }
    }
}
```

The following table provides the list of attributes needed to define a Tumbling Window dependency.

PROPERTY NAME	DESCRIPTION	TYPE	REQUIRED
type	All the existing tumbling window triggers are displayed in this drop down. Choose the trigger to take dependency on.	TumblingWindowTriggerDependencyReference or SelfDependencyTumblingWindowTriggerReference	Yes
offset	Offset of the dependency trigger. Provide a value in time span format and both negative and positive offsets are allowed. This property is mandatory if the trigger is depending on itself and in all other cases it is optional. Self-dependency should always be a negative offset. If no value specified, the window is the same as the trigger itself.	Timespan (hh:mm:ss)	Self-Dependency: Yes Other: No

PROPERTY NAME	DESCRIPTION	TYPE	REQUIRED
size	Size of the dependency tumbling window. Provide a positive timespan value. This property is optional.	Timespan (hh:mm:ss)	No

> **NOTE**
> A tumbling window trigger can depend on a maximum of five other triggers.

Tumbling window self-dependency properties

In scenarios where the trigger shouldn't proceed to the next window until the preceding window is successfully completed, build a self-dependency. A self-dependency trigger that's dependent on the success of earlier runs of itself within the preceding hour will have the properties indicated in the following code.

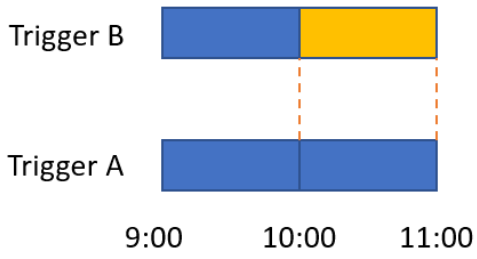
> **NOTE**
> If your triggered pipeline relies on the output of pipelines in previously triggered windows, we recommend using only tumbling window trigger self-dependency. To limit parallel trigger runs, set the maximimum trigger concurrency.

```
{
    "name": "DemoSelfDependency",
    "properties": {
        "runtimeState": "Started",
        "pipeline": {
            "pipelineReference": {
                "referenceName": "Demo",
                "type": "PipelineReference"
            }
        },
        "type": "TumblingWindowTrigger",
        "typeProperties": {
            "frequency": "Hour",
            "interval": 1,
            "startTime": "2018-10-04T00:00:00Z",
            "delay": "00:01:00",
            "maxConcurrency": 50,
            "retryPolicy": {
                "intervalInSeconds": 30
            },
            "dependsOn": [
                {
                    "type": "SelfDependencyTumblingWindowTriggerReference",
                    "size": "01:00:00",
                    "offset": "-01:00:00"
                }
            ]
        }
    }
}
```

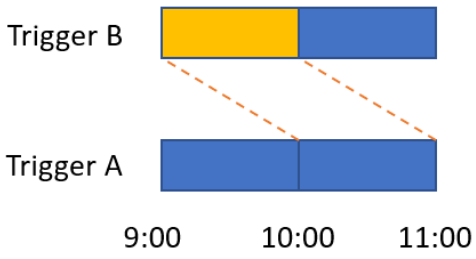
Usage scenarios and examples

Below are illustrations of scenarios and usage of tumbling window dependency properties.

Dependency offset

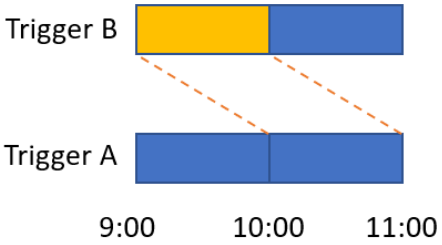


Dependency: A -> B
Offset: 0

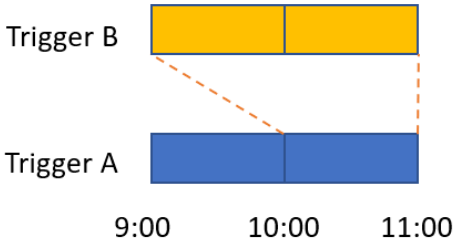


Dependency: A -> B
Offset: -1 hour

Dependency size

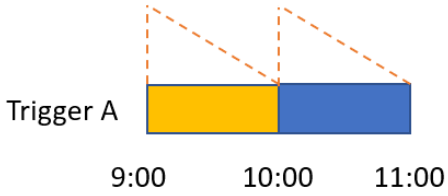


Dependency: A -> B
Offset: -1 hour
Size: unspecified

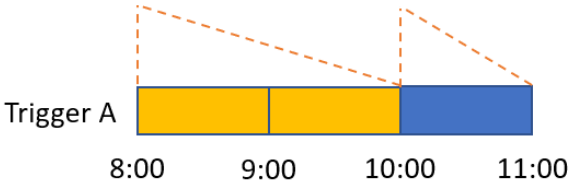


Dependency: A -> B
Offset: -1 hour
Size: 2 hours

Self-dependency



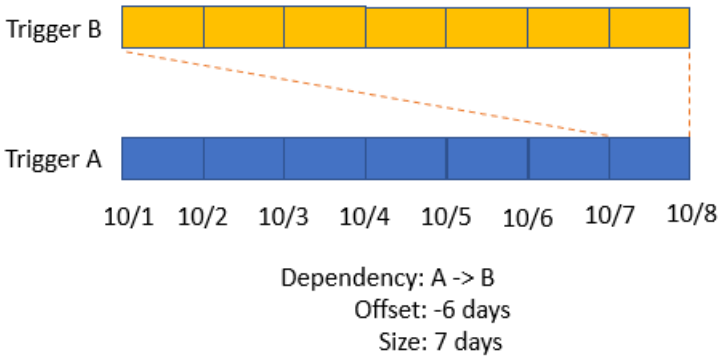
Dependency: A -> A
Offset: -1 hour
Size: unspecified



Dependency: A -> A
Offset: -2 hour
Size: 2 hours

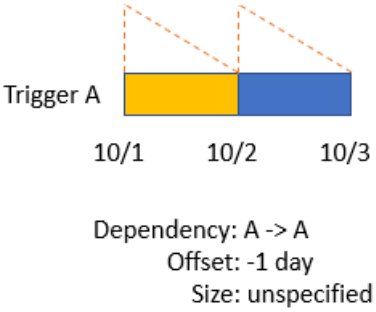
Dependency on another tumbling window trigger

A daily telemetry processing job depending on another daily job aggregating the last seven days output and generates seven day rolling window streams:



Dependency on itself

A daily job with no gaps in the output streams of the job:

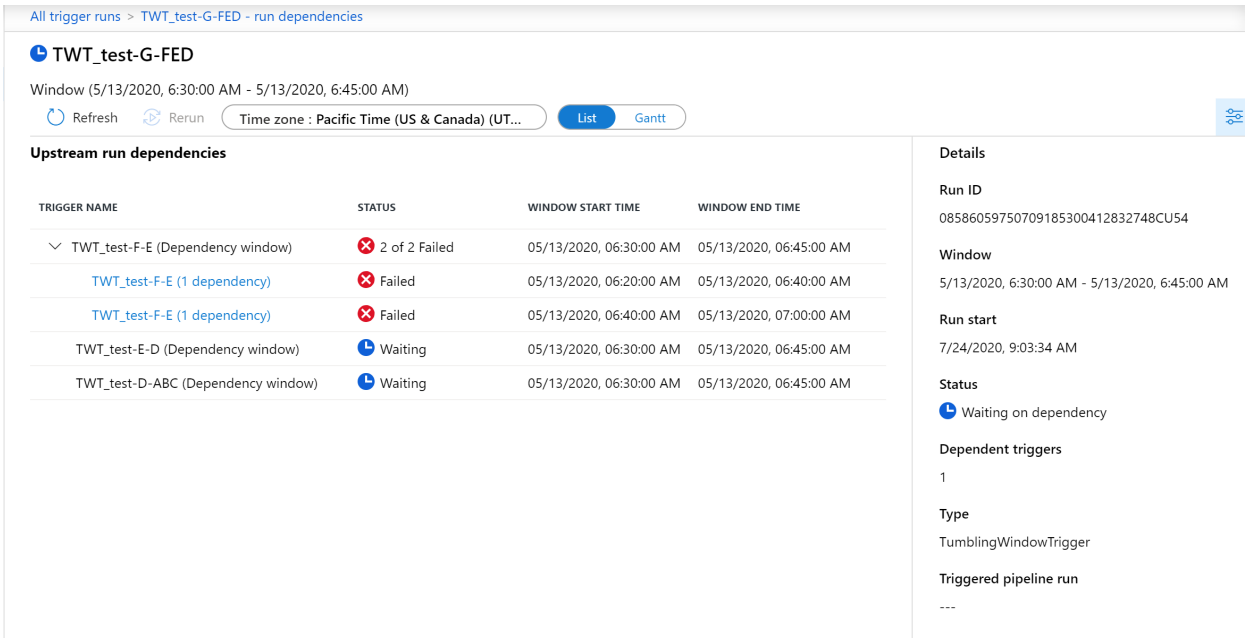


Monitor dependencies

You can monitor the dependency chain and the corresponding windows from the trigger run monitoring page. Navigate to **Monitoring** > **Trigger Runs**. If a Tumbling Window trigger has dependencies, Trigger Name will bear a hyperlink to dependency monitoring view.

TRIGGER NAME	TRIGGER TYPE	TRIGGER TIME	STATUS
TWT_test-D-ABC	TumblingWindowTrigger	7/24/20, 4:03:54 PM	Waiting on dependency
TWT_test-C	TumblingWindowTrigger	7/24/20, 4:03:54 PM	Waiting
TWT_test-D-ABC	TumblingWindowTrigger	7/24/20, 4:03:53 PM	Waiting on dependency
TWT_test-A	TumblingWindowTrigger	7/24/20, 4:03:53 PM	Waiting

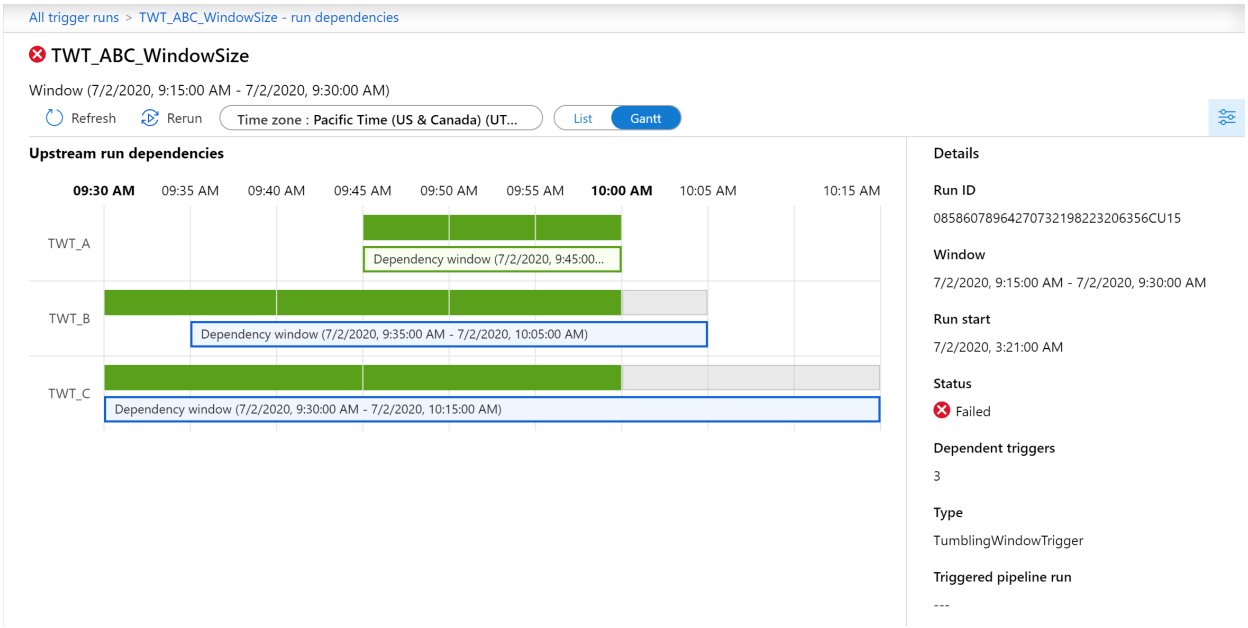
Click through the trigger name to view trigger dependencies. Right-hand panel shows detailed trigger run information, such as RunID, window time, status, and so on.



You can see the status of the dependencies, and windows for each dependent trigger. If one of the dependencies triggers fails, you must successfully rerun it in order for the dependent trigger to run.

A tumbling window trigger will wait on dependencies for *seven days* before timing out. After seven days, the trigger run will fail.

For a more visual to view the trigger dependency schedule, select the Gantt view.



Transparent boxes show the dependency windows for each down stream-dependent trigger, while solid colored boxes above show individual window runs. Here are some tips for interpreting the Gantt chart view:

- Transparent box renders blue when dependent windows are in pending or running state
- After all windows succeeds for a dependent trigger, the transparent box will turn green
- Transparent box renders red when some dependent window fails. Look for a solid red box to identify the failure window run

To rerun a window in Gantt chart view, select the solid color box for the window, and an action panel will pop up with details and rerun options

Next steps

- Review How to create a tumbling window trigger

Troubleshoot Azure Data Factory and Synapse pipelines

8/19/2022 • 38 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article explores common troubleshooting methods for external control activities in Azure Data Factory and Synapse pipelines.

Connector and copy activity

For connector issues such as an encounter error using the copy activity, refer to the Troubleshoot Connectors article.

Azure Databricks

Error code: 3200

- **Message**: Error 403.
- **Cause**: `The Databricks access token has expired.`
- **Recommendation**: By default, the Azure Databricks access token is valid for 90 days. Create a new token and update the linked service.

Error code: 3201

- **Message**: `Missing required field: settings.task.notebook_task.notebook_path.`
- **Cause**: `Bad authoring: Notebook path not specified correctly.`
- **Recommendation**: Specify the notebook path in the Databricks activity.

- **Message**: `Cluster... does not exist.`
- **Cause**: `Authoring error: Databricks cluster does not exist or has been deleted.`
- **Recommendation**: Verify that the Databricks cluster exists.

- **Message**: `Invalid Python file URI... Please visit Databricks user guide for supported URI schemes.`
- **Cause**: `Bad authoring.`
- **Recommendation**: Specify either absolute paths for workspace-addressing schemes, or `dbfs:/folder/subfolder/foo.py` for files stored in the Databricks File System (DFS).

- **Message**: `{0} LinkedService should have domain and accessToken as required properties.`
- **Cause**: `Bad authoring.`
- **Recommendation**: Verify the linked service definition.

- **Message**:
  `{0} LinkedService should specify either existing cluster ID or new cluster information for creation.`
- **Cause**: `Bad authoring.`
- **Recommendation**: Verify the linked service definition.

- **Message**:
  ```
  Node type Standard_D16S_v3 is not supported. Supported node types: Standard_DS3_v2, Standard_DS4_v2,
  Standard_DS5_v2, Standard_D8s_v3, Standard_D16s_v3, Standard_D32s_v3, Standard_D64s_v3,
  Standard_D3_v2, Standard_D8_v3, Standard_D16_v3, Standard_D32_v3, Standard_D64_v3, Standard_D12_v2,
  Standard_D13_v2, Standard_D14_v2, Standard_D15_v2, Standard_DS12_v2, Standard_DS13_v2,
  Standard_DS14_v2, Standard_DS15_v2, Standard_E8s_v3, Standard_E16s_v3, Standard_E32s_v3,
  Standard_E64s_v3, Standard_L4s, Standard_L8s, Standard_L16s, Standard_L32s, Standard_F4s,
  Standard_F8s, Standard_F16s, Standard_H16, Standard_F4s_v2, Standard_F8s_v2, Standard_F16s_v2,
  Standard_F32s_v2, Standard_F64s_v2, Standard_F72s_v2, Standard_NC12, Standard_NC24, Standard_NC6s_v3,
  Standard_NC12s_v3, Standard_NC24s_v3, Standard_L8s_v2, Standard_L16s_v2, Standard_L32s_v2,
  Standard_L64s_v2, Standard_L80s_v2.
  ```
- **Cause**: `Bad authoring.`
- **Recommendation**: Refer to the error message.

Error code: 3202

- **Message**:
  ```
  There were already 1000 jobs created in past 3600 seconds, exceeding rate limit: 1000 job creations
  per 3600 seconds.
  ```
- **Cause**: `Too many Databricks runs in an hour.`
- **Recommendation**: Check all pipelines that use this Databricks workspace for their job creation rate. If pipelines launched too many Databricks runs in aggregate, migrate some pipelines to a new workspace.

- **Message**:
  ```
  Could not parse request object: Expected 'key' and 'value' to be set for JSON map field
  base_parameters, got 'key: "..."' instead.
  ```
- **Cause**: `Authoring error: No value provided for the parameter.`
- **Recommendation**: Inspect the pipeline JSON and ensure all parameters in the baseParameters notebook specify a nonempty value.

- **Message**: `User:` SimpleUserContext{userId=..., name=user@company.com, orgId=...}
  `is not authorized to access cluster.`
- **Cause**: The user who generated the access token isn't allowed to access the Databricks cluster specified in the linked service.
- **Recommendation**: Ensure the user has the required permissions in the workspace.

- **Message**: `Job is not fully initialized yet. Please retry later.`
- **Cause**: The job has not initialized.
- **Recommendation**: Wait and try again later.

Error code: 3203

- **Message**:
  ```
  The cluster is in Terminated state, not available to receive jobs. Please fix the cluster or retry later.
  ```
- **Cause**: The cluster was terminated. For interactive clusters, this issue might be a race condition.
- **Recommendation**: To avoid this error, use job clusters.

Error code: 3204

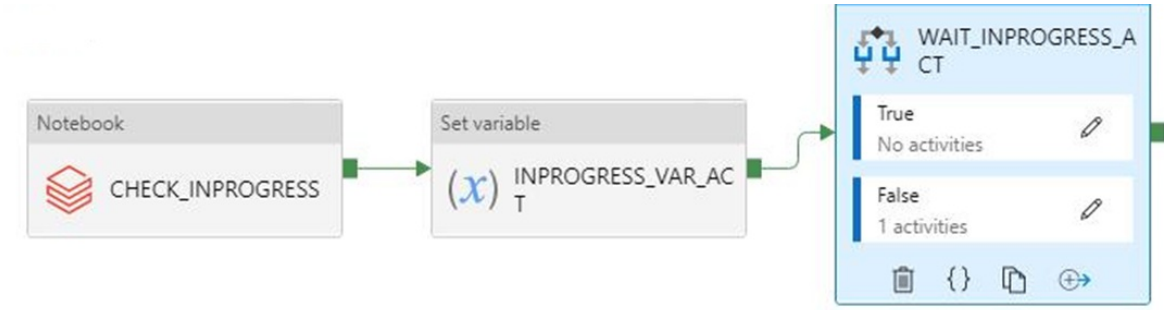
- **Message**: `Job execution failed.`
- **Cause**: Error messages indicate various issues, such as an unexpected cluster state or a specific activity. Often, no error message appears.
- **Recommendation**: N/A

Error code: 3208

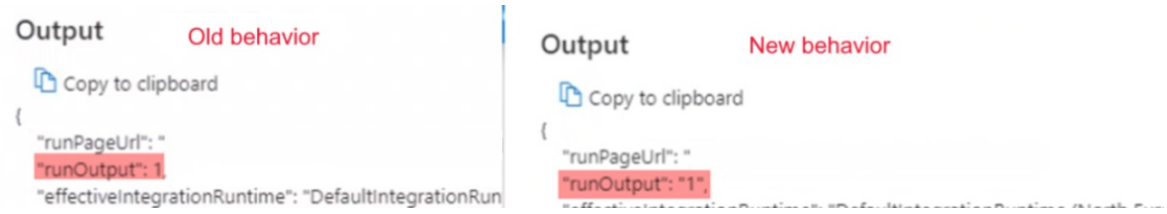
- **Message**: `An error occurred while sending the request.`
- **Cause**: The network connection to the Databricks service was interrupted.
- **Recommendation**: If you're using a self-hosted integration runtime, make sure that the network connection is reliable from the integration runtime nodes. If you're using Azure integration runtime, retry usually works.

The Boolean run output starts coming as string instead of expected int

- **Symptoms**: Your Boolean run output starts coming as string (for example, `"0"` or `"1"`) instead of expected int (for example, `0` or `1`).



You noticed this change on September 28, 2021 at around 9 AM IST when your pipeline relying on this output started failing. No change was made on the pipeline, and the Boolean output had been coming as expected before the failure.



- **Cause**: This issue is caused by a recent change, which is by design. After the change, if the result is a number that starts with zero, Azure Data Factory will convert the number to the octal value, which is a bug. This number is always 0 or 1, which never caused issues before the change. So to fix the octal conversion, the string output is passed from the Notebook run as is.
- **Recommendation**: Change the **if** condition to something like `if(value=="0")`.

Azure Data Lake Analytics

The following table applies to U-SQL.

Error code: 2709

- **Message**: `The access token is from the wrong tenant.`
- **Cause**: Incorrect Azure Active Directory (Azure AD) tenant.
- **Recommendation**: Incorrect Azure Active Directory (Azure AD) tenant.

- **Message**:
  ```
  We cannot accept your job at this moment. The maximum number of queued jobs for your account is 200.
  ```
- **Cause**: This error is caused by throttling on Data Lake Analytics.
- **Recommendation**: Reduce the number of submitted jobs to Data Lake Analytics. Either change triggers and concurrency settings on activities, or increase the limits on Data Lake Analytics.

- **Message**:
  ```
  This job was rejected because it requires 24 AUs. This account's administrator-defined policy prevents a job from using more than 5 AUs.
  ```
- **Cause**: This error is caused by throttling on Data Lake Analytics.
- **Recommendation**: Reduce the number of submitted jobs to Data Lake Analytics. Either change triggers and concurrency settings on activities, or increase the limits on Data Lake Analytics.

Error code: 2705

- **Message**:
  ```
  Forbidden. ACL verification failed. Either the resource does not exist or the user is not authorized to perform the requested operation.<br/> <br/> User is not able to access Data Lake Store. <br/> <br/> User is not authorized to use Data Lake Analytics.
  ```
- **Cause**: The service principal or certificate doesn't have access to the file in storage.
- **Recommendation**: Verify that the service principal or certificate that the user provides for Data Lake Analytics jobs has access to both the Data Lake Analytics account, and the default Data Lake Storage instance from the root folder.

Error code: 2711

- **Message**:
  ```
  Forbidden. ACL verification failed. Either the resource does not exist or the user is not authorized to perform the requested operation.<br/> <br/> User is not able to access Data Lake Store. <br/> <br/> User is not authorized to use Data Lake Analytics.
  ```
- **Cause**: The service principal or certificate doesn't have access to the file in storage.
- **Recommendation**: Verify that the service principal or certificate that the user provides for Data Lake Analytics jobs has access to both the Data Lake Analytics account, and the default Data Lake Storage instance from the root folder.

- **Message**: `Cannot find the 'Azure Data Lake Store' file or folder.`
- **Cause**: The path to the U-SQL file is wrong, or the linked service credentials don't have access.
- **Recommendation**: Verify the path and credentials provided in the linked service.

Error code: 2704

- **Message**:

```
Forbidden. ACL verification failed. Either the resource does not exist or the user is not authorized
to perform the requested operation.<br/> <br/> User is not able to access Data Lake Store. <br/> <br/>
User is not authorized to use Data Lake Analytics.
```

- **Cause**: The service principal or certificate doesn't have access to the file in storage.
- **Recommendation**: Verify that the service principal or certificate that the user provides for Data Lake Analytics jobs has access to both the Data Lake Analytics account, and the default Data Lake Storage instance from the root folder.

Error code: 2707

- **Message**:
  ```
  Cannot resolve the account of AzureDataLakeAnalytics. Please check 'AccountName' and
  'DataLakeAnalyticsUri'.
  ```
- **Cause**: The Data Lake Analytics account in the linked service is wrong.
- **Recommendation**: Verify that the right account is provided.

Error code: 2703

- **Message**: `Error Id: E_CQO_SYSTEM_INTERNAL_ERROR (or any error that starts with "Error Id:").`
- **Cause**: The error is from Data Lake Analytics.
- **Recommendation**: The job was submitted to Data Lake Analytics, and the script there, both failed. Investigate in Data Lake Analytics. In the portal, go to the Data Lake Analytics account and look for the job by using the Data Factory activity run ID (don't use the pipeline run ID). The job there provides more information about the error, and will help you troubleshoot.

  If the resolution isn't clear, contact the Data Lake Analytics support team and provide the job Universal Resource Locator (URL), which includes your account name and the job ID.

Azure functions

Error code: 3602

- **Message**: `Invalid HttpMethod: '%method;'.`
- **Cause**: The Httpmethod specified in the activity payload isn't supported by Azure Function Activity.
- **Recommendation**: The supported Httpmethods are: PUT, POST, GET, DELETE, OPTIONS, HEAD, and TRACE.

Error code: 3603

- **Message**: `Response Content is not a valid JObject.`
- **Cause**: The Azure function that was called didn't return a JSON Payload in the response. Azure Data Factory and Synapse pipeline Azure function activity only support JSON response content.
- **Recommendation**: Update the Azure function to return a valid JSON Payload such as a C# function may return `(ActionResult)new OkObjectResult("{\"Id\":\"123\"}");`

Error code: 3606

- **Message**: Azure function activity missing function key.
- **Cause**: The Azure function activity definition isn't complete.
- **Recommendation**: Check that the input Azure function activity JSON definition has a property named `functionKey`.

Error code: 3607

- **Message**: `Azure function activity missing function name.`
- **Cause**: The Azure function activity definition isn't complete.
- **Recommendation**: Check that the input Azure function activity JSON definition has a property named `functionName`.

Error code: 3608

- **Message**:
  ```
  Call to provided Azure function '%FunctionName;' failed with status-'%statusCode;' and message -
  '%message;'.
  ```
- **Cause**: The Azure function details in the activity definition may be incorrect.
- **Recommendation**: Fix the Azure function details and try again.

Error code: 3609

- **Message**: `Azure function activity missing functionAppUrl.`
- **Cause**: The Azure function activity definition isn't complete.
- **Recommendation**: Check that the input Azure Function activity JSON definition has a property named `functionAppUrl`.

Error code: 3610

- **Message**: `There was an error while calling endpoint.`
- **Cause**: The function URL may be incorrect.
- **Recommendation**: Verify that the value for `functionAppUrl` in the activity JSON is correct and try again.

Error code: 3611

- **Message**: `Azure function activity missing Method in JSON.`
- **Cause**: The Azure function activity definition isn't complete.
- **Recommendation**: Check that the input Azure function activity JSON definition has a property named `method`.

Error code: 3612

- **Message**: `Azure function activity missing LinkedService definition in JSON.`
- **Cause**: The Azure function activity definition isn't complete.
- **Recommendation**: Check that the input Azure function activity JSON definition has linked service details.

Azure Machine Learning

Error code: 4101

- **Message**:
  ```
  AzureMLExecutePipeline activity '%activityName;' has invalid value for property '%propertyName;'.
  ```
- **Cause**: Bad format or missing definition of property `%propertyName;`.
- **Recommendation**: Check if the activity `%activityName;` has the property `%propertyName;` defined with correct data.

Error code: 4110

- **Message**: `AzureMLExecutePipeline activity missing LinkedService definition in JSON.`
- **Cause**: The AzureMLExecutePipeline activity definition isn't complete.
- **Recommendation**: Check that the input AzureMLExecutePipeline activity JSON definition has correctly linked service details.

Error code: 4111

- **Message**:
  ```
  AzureMLExecutePipeline activity has wrong LinkedService type in JSON. Expected LinkedService type:
  '%expectedLinkedServiceType;', current LinkedService type: Expected LinkedService type:
  '%currentLinkedServiceType;'.
  ```
- **Cause**: Incorrect activity definition.
- **Recommendation**: Check that the input AzureMLExecutePipeline activity JSON definition has correctly linked service details.

Error code: 4112

- **Message**: `AzureMLService linked service has invalid value for property '%propertyName;'.`
- **Cause**: Bad format or missing definition of property '%propertyName;'.
- **Recommendation**: Check if the linked service has the property `%propertyName;` defined with correct data.

Error code: 4121

- **Message**:
  ```
  Request sent to Azure Machine Learning for operation '%operation;' failed with http status code
  '%statusCode;'. Error message from Azure Machine Learning: '%externalMessage;'.
  ```
- **Cause**: The Credential used to access Azure Machine Learning has expired.
- **Recommendation**: Verify that the credential is valid and retry.

Error code: 4122

- **Message**:
  ```
  Request sent to Azure Machine Learning for operation '%operation;' failed with http status code
  '%statusCode;'. Error message from Azure Machine Learning: '%externalMessage;'.
  ```
- **Cause**: The credential provided in Azure Machine Learning Linked Service is invalid, or doesn't have permission for the operation.
- **Recommendation**: Verify that the credential in Linked Service is valid, and has permission to access Azure Machine Learning.

Error code: 4123

- **Message**:
  ```
  Request sent to Azure Machine Learning for operation '%operation;' failed with http status code
  '%statusCode;'. Error message from Azure Machine Learning: '%externalMessage;'.
  ```
- **Cause**: The properties of the activity such as `pipelineParameters` are invalid for the Azure Machine Learning (ML) pipeline.
- **Recommendation**: Check that the value of activity properties matches the expected payload of the published Azure ML pipeline specified in Linked Service.

Error code: 4124

- **Message**:
  ```
  Request sent to Azure Machine Learning for operation '%operation;' failed with http status code
  '%statusCode;'. Error message from Azure Machine Learning: '%externalMessage;'.
  ```

- **Cause**: The published Azure ML pipeline endpoint doesn't exist.
- **Recommendation**: Verify that the published Azure Machine Learning pipeline endpoint specified in Linked Service exists in Azure Machine Learning.

Error code: 4125

- **Message**:
  ```
  Request sent to Azure Machine Learning for operation '%operation;' failed with http status code
  '%statusCode;'. Error message from Azure Machine Learning: '%externalMessage;'.
  ```
- **Cause**: There is a server error on Azure Machine Learning.
- **Recommendation**: Retry later. Contact the Azure Machine Learning team for help if the issue continues.

Error code: 4126

- **Message**:
  ```
  Azure ML pipeline run failed with status: '%amlPipelineRunStatus;'. Azure ML pipeline run Id:
  '%amlPipelineRunId;'. Please check in Azure Machine Learning for more error logs.
  ```
- **Cause**: The Azure ML pipeline run failed.
- **Recommendation**: Check Azure Machine Learning for more error logs, then fix the ML pipeline.

Azure Synapse Analytics

Error code: 3250

- **Message**: `There are not enough resources available in the workspace, details: '%errorMessage;'`
- **Cause**: Insufficient resources
- **Recommendation**: Try ending the running job(s) in the workspace, reducing the numbers of vCores requested, increasing the workspace quota or using another workspace.

Error code: 3251

- **Message**: `There are not enough resources available in the pool, details: '%errorMessage;'`
- **Cause**: Insufficient resources
- **Recommendation**: Try ending the running job(s) in the pool, reducing the numbers of vCores requested, increasing the pool maximum size or using another pool.

Error code: 3252

- **Message**: `There are not enough vcores available for your spark job, details: '%errorMessage;'`
- **Cause**: Insufficient vcores
- **Recommendation**: Try reducing the numbers of vCores requested or increasing your vCore quota. For more information, see Apache Spark core concepts.

Error code: 3253

- **Message**:
  ```
  There are substantial concurrent MappingDataflow executions which is causing failures due to
  throttling under the Integration Runtime used for ActivityId: '%activityId;'.
  ```
- **Cause**: Throttling threshold was reached.
- **Recommendation**: Retry the request after a wait period.

Error code: 3254

- **Message**: `AzureSynapseArtifacts linked service has invalid value for property '%propertyName;'.`

- **Cause**: Bad format or missing definition of property '%propertyName;'.
- **Recommendation**: Check if the linked service has property '%propertyName;' defined with correct data.

Common

Error code: 2103

- **Message**: `Please provide value for the required property '%propertyName;'.`
- **Cause**: The required value for the property has not been provided.
- **Recommendation**: Provide the value from the message and try again.

Error code: 2104

- **Message**: `The type of the property '%propertyName;' is incorrect.`
- **Cause**: The provided property type isn't correct.
- **Recommendation**: Fix the type of the property and try again.

Error code: 2105

- **Message**:
  ```
  An invalid json is provided for property '%propertyName;'. Encountered an error while trying to parse:
  '%message;'.
  ```
- **Cause**: The value for the property is invalid or isn't in the expected format.
- **Recommendation**: Refer to the documentation for the property and verify that the value provided includes the correct format and type.

Error code: 2106

- **Message**: `The storage connection string is invalid. %errorMessage;`
- **Cause**: The connection string for the storage is invalid or has incorrect format.
- **Recommendation**: Go to the Azure portal and find your storage, then copy-and-paste the connection string into your linked service and try again.

Error code: 2110

- **Message**:
  ```
  The linked service type '%linkedServiceType;' is not supported for '%executorType;' activities.
  ```
- **Cause**: The linked service specified in the activity is incorrect.
- **Recommendation**: Verify that the linked service type is one of the supported types for the activity. For example, the linked service type for HDI activities can be HDInsight or HDInsightOnDemand.

Error code: 2111

- **Message**:
  ```
  The type of the property '%propertyName;' is incorrect. The expected type is %expectedType;.
  ```
- **Cause**: The type of the provided property isn't correct.
- **Recommendation**: Fix the property type and try again.

Error code: 2112

- **Message**:
  ```
  The cloud type is unsupported or could not be determined for storage from the EndpointSuffix
  '%endpointSuffix;'.
  ```

- **Cause**: The cloud type is unsupported or couldn't be determined for storage from the EndpointSuffix.
- **Recommendation**: Use storage in another cloud and try again.

Custom

The following table applies to Azure Batch.

Error code: 2500

- **Message**: `Hit unexpected exception and execution failed.`
- **Cause**: `Can't launch command, or the program returned an error code.`
- **Recommendation**: Ensure that the executable file exists. If the program started, verify that *stdout.txt* and *stderr.txt* were uploaded to the storage account. It's a good practice to include logs in your code for debugging.

Error code: 2501

- **Message**: `Cannot access user batch account; please check batch account settings.`
- **Cause**: Incorrect Batch access key or pool name.
- **Recommendation**: Verify the pool name and the Batch access key in the linked service.

Error code: 2502

- **Message**: `Cannot access user storage account; please check storage account settings.`
- **Cause**: Incorrect storage account name or access key.
- **Recommendation**: Verify the storage account name and the access key in the linked service.

Error code: 2504

- **Message**: `Operation returned an invalid status code 'BadRequest'.`
- **Cause**: Too many files in the `folderPath` of the custom activity. The total size of `resourceFiles` can't be more than 32,768 characters.
- **Recommendation**: Remove unnecessary files, or Zip them and add an unzip command to extract them.

  For example, use

  ```
  powershell.exe -nologo -noprofile -command "& { Add-Type -A 'System.IO.Compression.FileSystem';
  [IO.Compression.ZipFile]::ExtractToDirectory($zipFile, $folder); }" ; $folder\yourProgram.exe
  ```

Error code: 2505

- **Message**: `Cannot create Shared Access Signature unless Account Key credentials are used.`
- **Cause**: Custom activities support only storage accounts that use an access key.
- **Recommendation**: Refer to the error description.

Error code: 2507

- **Message**: `The folder path does not exist or is empty: ...`
- **Cause**: No files are in the storage account at the specified path.
- **Recommendation**: The folder path must contain the executable files you want to run.

Error code: 2508

- **Message**: `There are duplicate files in the resource folder.`

- **Cause**: Multiple files of the same name are in different subfolders of folderPath.
- **Recommendation**: Custom activities flatten folder structure under folderPath. If you need to preserve the folder structure, zip the files and extract them in Azure Batch by using an unzip command.

  For example, use

  ```
  powershell.exe -nologo -noprofile -command "& { Add-Type -A 'System.IO.Compression.FileSystem';
  [IO.Compression.ZipFile]::ExtractToDirectory($zipFile, $folder); }" ; $folder\yourProgram.exe
  ```

Error code: 2509

- **Message**: `Batch url ... is invalid; it must be in Uri format.`
- **Cause**: Batch URLs must be similar to `https://mybatchaccount.eastus.batch.azure.com`
- **Recommendation**: Refer to the error description.

Error code: 2510

- **Message**: `An error occurred while sending the request.`
- **Cause**: The batch URL is invalid.
- **Recommendation**: Verify the batch URL.

HDInsight

Error code: 206

- **Message**: `The batch ID for Spark job is invalid. Please retry your job.`
- **Cause**: There was an internal problem with the service that caused this error.
- **Recommendation**: This issue could be transient. Retry your job after sometime.

Error code: 207

- **Message**:

  ```
  Could not determine the region from the provided storage account. Please try using another primary
  storage account for the on demand HDI.
  ```

- **Cause**: There was an internal error while trying to determine the region from the primary storage account.
- **Recommendation**: Try another storage.

Error code: 208

- **Message**:

  ```
  Service Principal or the MSI authenticator are not instantiated. Please consider providing a Service
  Principal in the HDI on demand linked service which has permissions to create an HDInsight cluster in
  the provided subscription and try again.
  ```

- **Cause**: There was an internal error while trying to read the Service Principal or instantiating the MSI authentication.
- **Recommendation**: Consider providing a service principal, which has permissions to create an HDInsight cluster in the provided subscription and try again. Verify that the Manage Identities are set up correctly.

Error code: 2300

- **Message**: `Failed to submit the job '%jobId;' to the cluster '%cluster;'. Error: %errorMessage;.`
- **Cause**: The error message contains a message similar to `The remote name could not be resolved.`. The provided cluster URI might be invalid.
- **Recommendation**: Verify that the cluster hasn't been deleted, and that the provided URI is correct. When

you open the URI in a browser, you should see the Ambari UI. If the cluster is in a virtual network, the URI should be the private URI. To open it, use a Virtual Machine (VM) that is part of the same virtual network.

For more information, see Directly connect to Apache Hadoop services.

- **Cause**: If the error message contains a message similar to `A task was canceled.`, the job submission timed out.
- **Recommendation**: The problem could be either general HDInsight connectivity or network connectivity. First confirm that the HDInsight Ambari UI is available from any browser. Then check that your credentials are still valid.

  If you're using a self-hosted integrated runtime (IR), perform this step from the VM or machine where the self-hosted IR is installed. Then try submitting the job again.

  For more information, read Ambari Web UI.

- **Cause**: When the error message contains a message similar to `User admin is locked out in Ambari` or `Unauthorized: Ambari user name or password is incorrect`, the credentials for HDInsight are incorrect or have expired.
- **Recommendation**: Correct the credentials and redeploy the linked service. First verify that the credentials work on HDInsight by opening the cluster URI on any browser and trying to sign in. If the credentials don't work, you can reset them from the Azure portal.

  For ESP cluster, reset the password through self service password reset.

- **Cause**: When the error message contains a message similar to `502 - Web server received an invalid response while acting as a gateway or proxy server`, this error is returned by HDInsight service.
- **Recommendation**: A 502 error often occurs when your Ambari Server process was shut down. You can restart the Ambari Services by rebooting the head node.

  1. Connect to one of your nodes on HDInsight using SSH.
  2. Identify your active head node host by running `ping headnodehost`.
  3. Connect to your active head node as Ambari Server sits on the active head node using SSH.
  4. Reboot the active head node.

     For more information, look through the Azure HDInsight troubleshooting documentation. For example:

     - Ambari UI 502 error
     - RpcTimeoutException for Apache Spark thrift server
     - Troubleshooting bad gateway errors in Application Gateway.

- **Cause**: When the error message contains a message similar to `Unable to service the submit job request as templeton service is busy with too many submit job requests` or `Queue root.joblauncher already has 500 applications, cannot accept submission of application`, too many jobs are being submitted to HDInsight at the same time.

- **Recommendation**: Limit the number of concurrent jobs submitted to HDInsight. Refer to activity concurrency if the jobs are being submitted by the same activity. Change the triggers so the concurrent pipeline runs are spread out over time.

  Refer to HDInsight documentation to adjust `templeton.parallellism.job.submit` as the error suggests.

Error code: 2301

- **Message**:
  ```
  Could not get the status of the application '%physicalJobId;' from the HDInsight service. Received the following error: %message;. Please refer to HDInsight troubleshooting documentation or contact their support for further assistance.
  ```
- **Cause**: HDInsight cluster or service has issues.
- **Recommendation**: This error occurs when the service doesn't receive a response from HDInsight cluster when attempting to request the status of the running job. This issue might be on the cluster itself, or HDInsight service might have an outage.

  Refer to HDInsight troubleshooting documentation, or contact Microsoft support for further assistance.

Error code: 2302

- **Message**:
  ```
  Hadoop job failed with exit code '%exitCode;'. See '%logPath;/stderr' for more details. Alternatively, open the Ambari UI on the HDI cluster and find the logs for the job '%jobId;'. Contact HDInsight team for further support.
  ```
- **Cause**: The job was submitted to the HDI cluster and failed there.
- **Recommendation**:

1. Check Ambari UI:
   a. Ensure that all services are still running.
   b. From Ambari UI, check the alert section in your dashboard.
      a. For more information on alerts and resolutions to alerts, see Managing and Monitoring a Cluster.
   c. Review your YARN memory. If your YARN memory is high, the processing of your jobs may be delayed. If you do not have enough resources to accommodate your Spark application/job, scale up the cluster to ensure the cluster has enough memory and cores.
2. Run a Sample test job.
   a. If you run the same job on HDInsight backend, check that it succeeded. For examples of sample runs, see Run the MapReduce examples included in HDInsight
3. If the job still failed on HDInsight, check the application logs and information, which to provide to Support:
   a. Check whether the job was submitted to YARN. If the job wasn't submitted to yarn, use `--master yarn`.
   b. If the application finished execution, collect the start time and end time of the YARN Application. If the application didn't complete the execution, collect Start time/Launch time.
   c. Check and collect application log with `yarn logs -applicationId <Insert_Your_Application_ID>`.
   d. Check and collect the yarn Resource Manager logs under the `/var/log/hadoop-yarn/yarn` directory.
   e. If these steps are not enough to resolve the issue, contact Azure HDInsight team for support and provide the above logs and timestamps.

Error code: 2303

- **Message**:
  ```
  Hadoop job failed with transient exit code '%exitCode;'. See '%logPath;/stderr' for more details. Alternatively, open the Ambari UI on the HDI cluster and find the logs for the job '%jobId;'. Try again or contact HDInsight team for further support.
  ```
- **Cause**: The job was submitted to the HDI cluster and failed there.

- **Recommendation**:

1. Check Ambari UI:
   a. Ensure that all services are still running.
   b. From Ambari UI, check the alert section in your dashboard.
      a. For more information on alerts and resolutions to alerts, see Managing and Monitoring a Cluster.
   c. Review your YARN memory. If your YARN memory is high, the processing of your jobs may be delayed. If you do not have enough resources to accommodate your Spark application/job, scale up the cluster to ensure the cluster has enough memory and cores.
2. Run a Sample test job.
   a. If you run the same job on HDInsight backend, check that it succeeded. For examples of sample runs, see Run the MapReduce examples included in HDInsight
3. If the job still failed on HDInsight, check the application logs and information, which to provide to Support:
   a. Check whether the job was submitted to YARN. If the job wasn't submitted to yarn, use `--master yarn`.
   b. If the application finished execution, collect the start time and end time of the YARN Application. If the application didn't complete the execution, collect Start time/Launch time.
   c. Check and collect application log with `yarn logs -applicationId <Insert_Your_Application_ID>`.
   d. Check and collect the yarn Resource Manager logs under the `/var/log/hadoop-yarn/yarn` directory.
   e. If these steps are not enough to resolve the issue, contact Azure HDInsight team for support and provide the above logs and timestamps.

Error code: 2304

- **Message**: `MSI authentication is not supported on storages for HDI activities.`
- **Cause**: The storage linked services used in the HDInsight (HDI) linked service or HDI activity, are configured with an MSI authentication that isn't supported.
- **Recommendation**: Provide full connection strings for storage accounts used in the HDI linked service or HDI activity.

Error code: 2305

- **Message**: `Failed to initialize the HDInsight client for the cluster '%cluster;'. Error: '%message;'`
- **Cause**: The connection information for the HDI cluster is incorrect, the provided user doesn't have permissions to perform the required action, or the HDInsight service has issues responding to requests from the service.
- **Recommendation**: Verify that the user information is correct, and that the Ambari UI for the HDI cluster can be opened in a browser from the VM where the IR is installed (for a self-hosted IR), or can be opened from any machine (for Azure IR).

Error code: 2306

- **Message**: `An invalid json is provided for script action '%scriptActionName;'. Error: '%message;'`
- **Cause**: The JSON provided for the script action is invalid.
- **Recommendation**: The error message should help to identify the issue. Fix the json configuration and try again.

  Check Azure HDInsight on-demand linked service for more information.

Error code: 2310

- **Message**: `Failed to submit Spark job. Error: '%message;'`

- **Cause**: The service tried to create a batch on a Spark cluster using Livy API (livy/batch), but received an error.
- **Recommendation**: Follow the error message to fix the issue. If there isn't enough information to get it resolved, contact the HDI team and provide them the batch ID and job ID, which can be found in the activity run Output in the service Monitoring page. To troubleshoot further, collect the full log of the batch job.

  For more information on how to collect the full log, see Get the full log of a batch job.

Error code: 2312

- **Message**:
  ```
  Spark job failed, batch id:%batchId;. Please follow the links in the activity run Output from the service Monitoring page to troubleshoot the run on HDInsight Spark cluster. Please contact HDInsight support team for further assistance.
  ```
- **Cause**: The job failed on the HDInsight Spark cluster.
- **Recommendation**: Follow the links in the activity run Output in the service Monitoring page to troubleshoot the run on HDInsight Spark cluster. Contact HDInsight support team for further assistance.

  For more information on how to collect the full log, see Get the full log of a batch job.

Error code: 2313

- **Message**:
  ```
  The batch with ID '%batchId;' was not found on Spark cluster. Open the Spark History UI and try to find it there. Contact HDInsight support for further assistance.
  ```
- **Cause**: The batch was deleted on the HDInsight Spark cluster.
- **Recommendation**: Troubleshoot batches on the HDInsight Spark cluster. Contact HDInsight support for further assistance.

  For more information on how to collect the full log, see Get the full log of a batch job, and share the full log with HDInsight support for further assistance.

Error code: 2328

- **Message**:
  ```
  Failed to create the on demand HDI cluster. Cluster or linked service name: '%clusterName;', error: '%message;'
  ```
- **Cause**: The error message should show the details of what went wrong.
- **Recommendation**: The error message should help to troubleshoot the issue.

Error code: 2329

- **Message**:
  ```
  Failed to delete the on demand HDI cluster. Cluster or linked service name: '%clusterName;', error: '%message;'
  ```
- **Cause**: The error message should show the details of what went wrong.
- **Recommendation**: The error message should help to troubleshoot the issue.

Error code: 2331

- **Message**: `The file path should not be null or empty.`
- **Cause**: The provided file path is empty.
- **Recommendation**: Provide a path for a file that exists.

Error code: 2340

- **Message**:
  ```
  HDInsightOnDemand linked service does not support execution via SelfHosted IR. Your IR name is '%IRName;'. Please select an Azure IR instead.
  ```
- **Cause**: The HDInsightOnDemand linked service doesn't support execution via SelfHosted IR.
- **Recommendation**: Select an Azure IR and try again.

Error code: 2341

- **Message**:
  ```
  HDInsight cluster URL '%clusterUrl;' is incorrect, it must be in URI format and the scheme must be 'https'.
  ```
- **Cause**: The provided URL isn't in correct format.
- **Recommendation**: Fix the cluster URL and try again.

Error code: 2342

- **Message**: `Failed to connect to HDInsight cluster: '%errorMessage;'.`
- **Cause**: Either the provided credentials are wrong for the cluster, or there was a network configuration or connection issue, or the IR is having problems connecting to the cluster.
- **Recommendation**:

  1. Verify that the credentials are correct by opening the HDInsight cluster's Ambari UI in a browser.
  2. If the cluster is in Virtual Network (VNet) and a self-hosted IR is being used, the HDI URL must be the private URL in VNets, and should have `-int` listed after the cluster name.

     For example, change `https://mycluster.azurehdinsight.net/` to `https://mycluster-int.azurehdinsight.net/`. Note the `-int` after `mycluster`, but before `.azurehdinsight.net`
  3. If the cluster is in VNet, the self-hosted IR is being used, and the private URL was used, and yet the connection still failed, then the VM where the IR is installed had problems connecting to the HDI.

     Connect to the VM where the IR is installed and open the Ambari UI in a browser. Use the private URL for the cluster. This connection should work from the browser. If it doesn't, contact HDInsight support team for further assistance.
  4. If self-hosted IR isn't being used, then the HDI cluster should be accessible publicly. Open the Ambari UI in a browser and check that it opens up. If there are any issues with the cluster or the services on it, contact HDInsight support team for assistance.

     The HDI cluster URL used in the linked service must be accessible for the IR (self-hosted or Azure) in order for the test connection to pass, and for runs to work. This state can be verified by opening the URL from a browser either from VM, or from any public machine.

Error code: 2343

- **Message**: `User name and password cannot be null or empty to connect to the HDInsight cluster.`
- **Cause**: Either the user name or the password is empty.
- **Recommendation**: Provide the correct credentials to connect to HDI and try again.

Error code: 2345

- **Message**: `Failed to read the content of the hive script. Error: '%message;'`
- **Cause**: The script file doesn't exist or the service couldn't connect to the location of the script.

- **Recommendation**: Verify that the script exists, and that the associated linked service has the proper credentials for a connection.

Error code: 2346

- **Message**: `Failed to create ODBC connection to the HDI cluster with error message '%message;'.`
- **Cause**: The service tried to establish an Open Database Connectivity (ODBC) connection to the HDI cluster, and it failed with an error.
- **Recommendation**:

  1. Confirm that you correctly set up your ODBC/Java Database Connectivity (JDBC) connection.
     a. For JDBC, if you're using the same virtual network, you can get this connection from: `Hive -> Summary -> HIVESERVER2 JDBC URL`
     b. To ensure that you have the correct JDBC set up, see Query Apache Hive through the JDBC driver in HDInsight.
     c. For Open Database (ODB), see Tutorial: Query Apache Hive with ODBC and PowerShell to ensure that you have the correct setup.
  2. Verify that Hiveserver2, Hive Metastore, and Hiveserver2 Interactive are active and working.
  3. Check the Ambari user interface (UI):
     a. Ensure that all services are still running.
     b. From the Ambari UI, check the alert section in your dashboard.
        a. For more information on alerts and resolutions to alerts, see Managing and Monitoring a Cluster .
  4. If these steps are not enough to resolve the issue, contact the Azure HDInsight team.

Error code: 2347

- **Message**: `Hive execution through ODBC failed with error message '%message;'.`
- **Cause**: The service submitted the hive script for execution to the HDI cluster via ODBC connection, and the script has failed on HDI.
- **Recommendation**:

  1. Confirm that you correctly set up your ODBC/Java Database Connectivity (JDBC) connection.
     a. For JDBC, if you're using the same virtual network, you can get this connection from: `Hive -> Summary -> HIVESERVER2 JDBC URL`
     b. To ensure that you have the correct JDBC set up, see Query Apache Hive through the JDBC driver in HDInsight.
     c. For Open Database (ODB), see Tutorial: Query Apache Hive with ODBC and PowerShell to ensure that you have the correct setup.
  2. Verify that Hiveserver2, Hive Metastore, and Hiveserver2 Interactive are active and working.
  3. Check the Ambari user interface (UI):
     a. Ensure that all services are still running.
     b. From the Ambari UI, check the alert section in your dashboard.
        a. For more information on alerts and resolutions to alerts, see Managing and Monitoring a Cluster .
  4. If these steps are not enough to resolve the issue, contact the Azure HDInsight team.

Error code: 2348

- **Message**:

```
The main storage has not been initialized. Please check the properties of the storage linked service
in the HDI linked service.
```

- **Cause**: The storage linked service properties are not set correctly.
- **Recommendation**: Only full connection strings are supported in the main storage linked service for HDI activities. Verify that you are not using MSI authorizations or applications.

Error code: 2350

- **Message**:
  ```
  Failed to prepare the files for the run '%jobId;'. HDI cluster: '%cluster;', Error: '%errorMessage;'
  ```
- **Cause**: The credentials provided to connect to the storage where the files should be located are incorrect, or the files do not exist there.
- **Recommendation**: This error occurs when the service prepares for HDI activities, and tries to copy files to the main storage before submitting the job to HDI. Check that files exist in the provided location, and that the storage connection is correct. As HDI activities do not support MSI authentication on storage accounts related to HDI activities, verify that those linked services have full keys or are using Azure Key Vault.

Error code: 2351

- **Message**: `Could not open the file '%filePath;' in container/fileSystem '%container;'.`
- **Cause**: The file doesn't exist at specified path.
- **Recommendation**: Check whether the file actually exists, and that the linked service with connection info pointing to this file has the correct credentials.

Error code: 2352

- **Message**:
  ```
  The file storage has not been initialized. Please check the properties of the file storage linked
  service in the HDI activity.
  ```
- **Cause**: The file storage linked service properties are not set correctly.
- **Recommendation**: Verify that the properties of the file storage linked service are properly configured.

Error code: 2353

- **Message**:
  ```
  The script storage has not been initialized. Please check the properties of the script storage linked
  service in the HDI activity.
  ```
- **Cause**: The script storage linked service properties are not set correctly.
- **Recommendation**: Verify that the properties of the script storage linked service are properly configured.

Error code: 2354

- **Message**:
  ```
  The storage linked service type '%linkedServiceType;' is not supported for '%executorType;' activities
  for property '%linkedServicePropertyName;'.
  ```
- **Cause**: The storage linked service type isn't supported by the activity.
- **Recommendation**: Verify that the selected linked service has one of the supported types for the activity. HDI activities support AzureBlobStorage and AzureBlobFSStorage linked services.

  For more information, read Compare storage options for use with Azure HDInsight clusters

Error code: 2355

- **Message**:
  ```
  The '%value' provided for commandEnvironment is incorrect. The expected value should be an array of
  strings where each string has the format CmdEnvVarName=CmdEnvVarValue.
  ```

- **Cause**: The provided value for `commandEnvironment` is incorrect.
- **Recommendation**: Verify that the provided value is similar to:

  ```
  \"variableName=variableValue\"
  ]
  ```

  Also verify that each variable appears in the list only once.

Error code: 2356

- **Message**: `The commandEnvironment already contains a variable named '%variableName;'.`
- **Cause**: The provided value for `commandEnvironment` is incorrect.
- **Recommendation**: Verify that the provided value is similar to:

  ```
  \"variableName=variableValue\"
  ]
  ```

  Also verify that each variable appears in the list only once.

Error code: 2357

- **Message**: `The certificate or password is wrong for ADLS Gen 1 storage.`
- **Cause**: The provided credentials are incorrect.
- **Recommendation**: Verify that the connection information in ADLS Gen 1 linked to the service, and verify that the test connection succeeds.

Error code: 2358

- **Message**:

  ```
  The value '%value;' for the required property 'TimeToLive' in the on demand HDInsight linked service
  '%linkedServiceName;' has invalid format. It should be a timespan between '00:05:00' and '24:00:00'.
  ```

- **Cause**: The provided value for the required property `TimeToLive` has an invalid format.
- **Recommendation**: Update the value to the suggested range and try again.

Error code: 2359

- **Message**:

  ```
  The value '%value;' for the property 'roles' is invalid. Expected types are 'zookeeper', 'headnode',
  and 'workernode'.
  ```

- **Cause**: The provided value for the property `roles` is invalid.
- **Recommendation**: Update the value to be one of the suggestions and try again.

Error code: 2360

- **Message**:

  ```
  The connection string in HCatalogLinkedService is invalid. Encountered an error while trying to parse:
  '%message;'.
  ```

- **Cause**: The provided connection string for the `HCatalogLinkedService` is invalid.
- **Recommendation**: Update the value to a correct Azure SQL connection string and try again.

Error code: 2361

- **Message**: `Failed to create on demand HDI cluster. Cluster name is '%clusterName;'.`

- **Cause**: The cluster creation failed, and the service did not get an error back from HDInsight service.
- **Recommendation**: Open the Azure portal and try to find the HDI resource with provided name, then check the provisioning status. Contact HDInsight support team for further assistance.

Error code: 2362

- **Message**:

```
Only Azure Blob storage accounts are supported as additional storages for HDInsight on demand linked service.
```

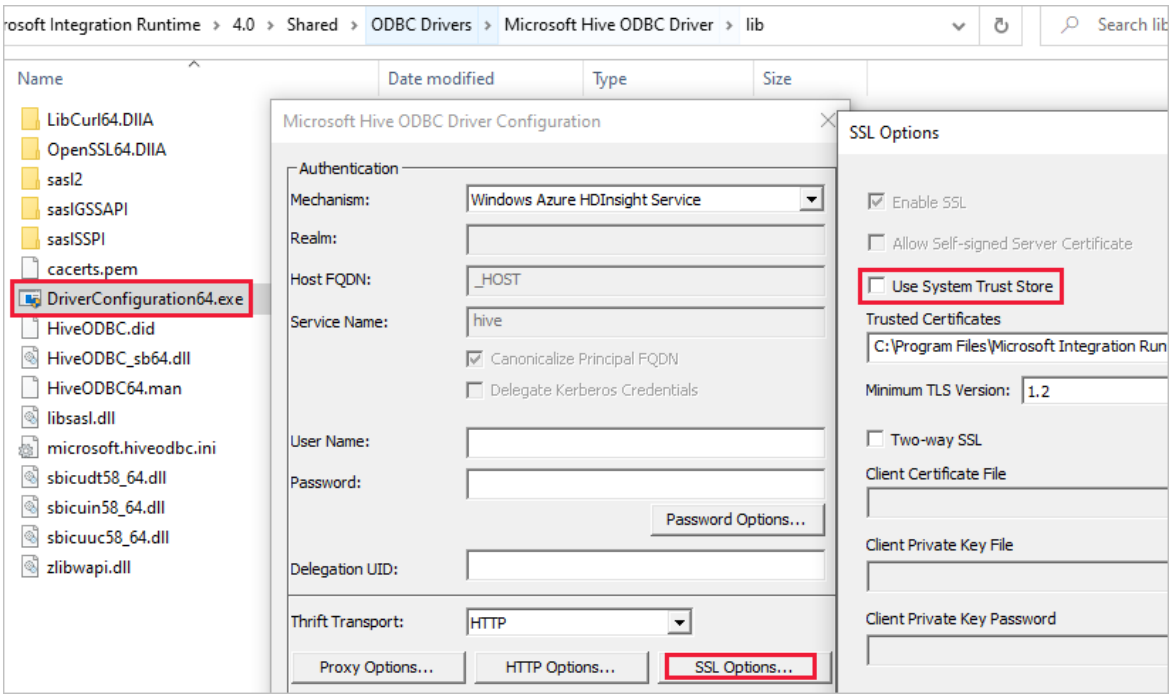
- **Cause**: The provided additional storage was not Azure Blob storage.
- **Recommendation**: Provide an Azure Blob storage account as an additional storage for HDInsight on-demand linked service.

SSL error when linked service using HDInsight ESP cluster

- **Message**:

```
Failed to connect to HDInsight cluster: 'ERROR [HY000] [Microsoft][DriverSupport] (1100) SSL certificate verification failed because the certificate is missing or incorrect.
```

- **Cause**: The issue is most likely related with System Trust Store.
- **Resolution**: You can navigate to the path **Microsoft Integration Runtime\4.0\Shared\ODBC Drivers\Microsoft Hive ODBC Driver\lib** and open DriverConfiguration64.exe to change the setting.



HDI activity stuck in preparing for cluster

If the HDI activity is stuck in preparing for cluster, follow the guidelines below:

1. Make sure the timeout is greater than what is described below and wait for the execution to complete or until it is timed out, and wait for Time To Live (TTL) time before submitting new jobs.

   The max default time that it takes to spin up a cluster is 2 hours, and if you have any init script, it will add up, up to another 2 hours.

2. Make sure the storage and HDI are provisioned in the same region.
3. Make sure that the service principal used for accessing the HDI cluster is valid.
4. If the issue still persists, as a workaround, delete the HDI linked service and re-create it with a new name.

Web Activity

Error code: 2128

- **Message**:
  ```
  No response from the endpoint. Possible causes: network connectivity, DNS failure, server certificate validation or timeout.
  ```
- **Cause**: This issue is due to either Network connectivity, a DNS failure, a server certificate validation, or a timeout.
- **Recommendation**: Validate that the endpoint you are trying to hit is responding to requests. You may use tools like **Fiddler/Postman/Netmon/Wireshark**.

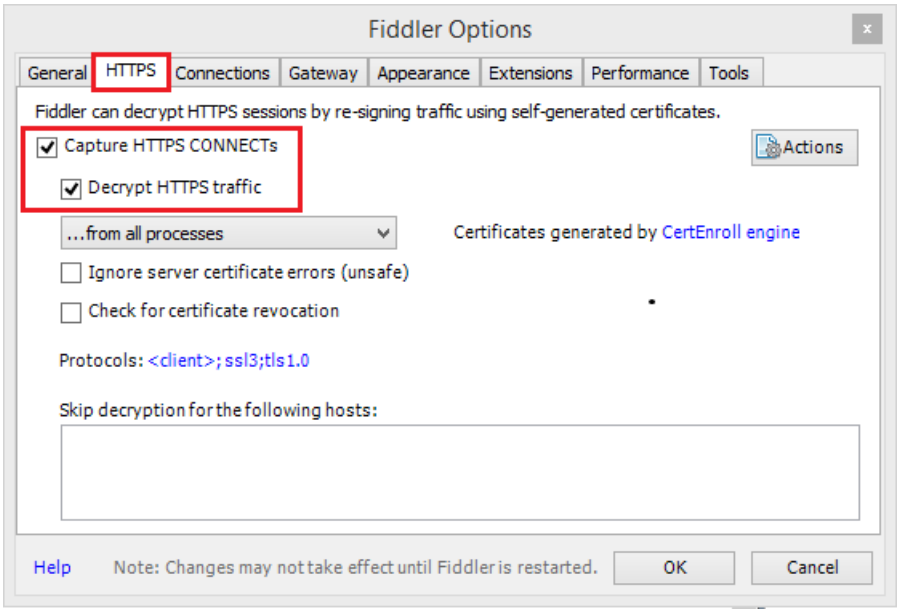
Error code: 2108

- **Message**: `Error calling the endpoint '%url;'. Response status code: '%code;'`
- **Cause**: The request failed due to an underlying issue such as network connectivity, a DNS failure, a server certificate validation, or a timeout.
- **Recommendation**: Use Fiddler/Postman/Netmon/Wireshark to validate the request.

More details

To use **Fiddler** to create an HTTP session of the monitored web application:

1. Download, install, and open Fiddler.
2. If your web application uses HTTPS, go to **Tools** > **Fiddler Options** > **HTTPS**.
   a. In the HTTPS tab, select both **Capture HTTPS CONNECTs** and **Decrypt HTTPS traffic**.



3. If your application uses TLS/SSL certificates, add the Fiddler certificate to your device.

   Go to: **Tools** > **Fiddler Options** > **HTTPS** > **Actions** > **Export Root Certificate to Desktop**.
4. Turn off capturing by going to **File** > **Capture Traffic**. Or press **F12**.
5. Clear your browser's cache so that all cached items are removed and must be downloaded again.
6. Create a request:
7. Select the **Composer** tab.
   a. Set the HTTP method and URL.

b. If needed, add headers and a request body.

c. Select **Execute**.

8. Turn on traffic capturing again, and complete the problematic transaction on your page.

9. Go to: **File** > **Save** > **All Sessions**.

For more information, see Getting started with Fiddler.

General

REST continuation token NULL error

Error message: {"token":null,"range":{"min":...}

Cause: When querying across multiple partitions/pages, backend service returns continuation token in JObject format with 3 properties: **token, min and max key ranges**, for instance, {"token":null,"range":{"min":"05C1E9AB0DAD76","max":"05C1E9CD673398"}}). Depending on source data, querying can result 0 indicating missing token though there is more data to fetch.

Recommendation: When the continuationToken is non-null, as the string {"token":null,"range":{"min":"05C1E9AB0DAD76","max":"05C1E9CD673398"}}, it is required to call queryActivityRuns API again with the continuation token from the previous response. You need to pass the full string for the query API again. The activities will be returned in the subsequent pages for the query result. You should ignore that there is empty array in this page, as long as the full continuationToken value != null, you need continue querying. For more details, please refer to REST api for pipeline run query.

Activity stuck issue

When you observe that the activity is running much longer than your normal runs with barely no progress, it may happen to be stuck. You can try canceling it and retry to see if it helps. If it's a copy activity, you can learn about the performance monitoring and troubleshooting from Troubleshoot copy activity performance; if it's a data flow, learn from Mapping data flows performance and tuning guide.

Payload is too large

Error message:

```
The payload including configurations on activity/dataSet/linked service is too large. Please check if you
have settings with very large value and try to reduce its size.
```

Cause: The payload for each activity run includes the activity configuration, the associated dataset(s), and linked service(s) configurations if any, and a small portion of system properties generated per activity type. The limit of such payload size is 896 KB as mentioned in the Azure limits documentation for Data Factory and Azure Synapse Analytics.

Recommendation: You hit this limit likely because you pass in one or more large parameter values from either upstream activity output or external, especially if you pass actual data across activities in control flow. Check if you can reduce the size of large parameter values, or tune your pipeline logic to avoid passing such values across activities and handle it inside the activity instead.

Unsupported compression causes files to be corrupted

Symptoms: You try to unzip a file that is stored in a blob container. A single copy activity in a pipeline has a source with the compression type set to "deflate64" (or any unsupported type). This activity runs successfully and produces the text file contained in the zip file. However, there is a problem with the text in the file, and this file appears corrupted. When this file is unzipped locally, it is fine.

Cause: Your zip file is compressed by the algorithm of "deflate64", while the internal zip library of Azure Data Factory only supports "deflate". If the zip file is compressed by the Windows system and the overall file size exceeds a certain number, Windows will use "deflate64" by default, which is not supported in Azure Data

Factory. On the other hand, if the file size is smaller or you use some third party zip tools that support specifying the compress algorithm, Windows will use "deflate" by default.

> **TIP**
>
> Actually, both Binary format in Azure Data Factory and Synapse Analytics and Delimited text format in Azure Data Factory and Azure Synapse Analytics clearly state that the "deflate64" format is not supported in Azure Data Factory.

Next steps

For more troubleshooting help, try these resources:

- Data Factory blog
- Data Factory feature requests
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory
- Azure videos
- Microsoft Q&A question page

Troubleshoot Azure Data Factory and Azure Synapse Analytics connectors

8/19/2022 • 6 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article describes how to troubleshoot connectors in Azure Data Factory and Azure Synapse Analytics.

Connector specific problems

You can refer to the troubleshooting pages for each connector to see problems specific to it with explanations of their causes and recommendations to resolve them.

- Azure Blob Storage
- Azure Cosmos DB (including SQL API connector)
- Azure Data Lake (Gen1 and Gen2)
- Azure Database for PostgreSQL
- Azure Files storage
- Azure Synapse Analytics, Azure SQL Database, and SQL Server
- DB2
- Delimited text format
- Dynamics 365, Dataverse (Common Data Service), and Dynamics CRM
- FTP, SFTP and HTTP
- Hive
- Oracle
- ORC format
- Parquet format
- REST
- SharePoint Online list
- XML format

General copy activity errors

The errors below are general to the copy activity and could occur with any connector.

Error code: JreNotFound

- **Message**:

  ```
  Java Runtime Environment cannot be found on the Self-hosted Integration Runtime machine. It is required for parsing or writing to Parquet/ORC files. Make sure Java Runtime Environment has been installed on the Self-hosted Integration Runtime machine.
  ```

- **Cause**: The self-hosted IR can't find Java Runtime. Java Runtime is required for reading particular sources.

- **Recommendation**: Check your integration runtime environment, see Use Self-hosted Integration Runtime.

Error code: WildcardPathSinkNotSupported

- **Message**: `Wildcard in path is not supported in sink dataset. Fix the path: '%setting;'.`

- **Cause**: The sink dataset doesn't support wildcard values.
- **Recommendation**: Check the sink dataset, and rewrite the path without using a wildcard value.

FIPS issue

- **Symptoms**: Copy activity fails on a FIPS-enabled self-hosted IR machine with the following error message:

  ```
  This implementation is not part of the Windows Platform FIPS validated cryptographic algorithms.
  ```

- **Cause**: This error might occur when you copy data with connectors such as Azure Blob, SFTP, and so on. Federal Information Processing Standards (FIPS) defines a certain set of cryptographic algorithms that are allowed to be used. When FIPS mode is enabled on the machine, some cryptographic classes that copy activity depends on are blocked in some scenarios.
- **Resolution**: Learn why we're not recommending "FIPS Mode" anymore, and evaluate whether you can disable FIPS on your self-hosted IR machine.

  Alternatively, if you only want to bypass FIPS and make the activity runs succeed, do the following:

  1. Open the folder where Self-hosted IR is installed. The path is usually *C:\Program Files\Microsoft Integration Runtime <IR version>\Shared*.
  2. Open the *diawp.exe.config* file and then, at the end of the `<runtime>` section, add `<enforceFIPSPolicy enabled="false"/>`, as shown here:

     ```
     <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
       <dependentAssembly>
         <assemblyIdentity name="BouncyCastle.Crypto" publicKeyToken="0e99375e54769942" culture="neutral" />
         <bindingRedirect oldVersion="0.0.0.0-1.8.5.0" newVersion="1.8.5.0" />
       </dependentAssembly>
     </assemblyBinding>
     <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
       <dependentAssembly>
         <assemblyIdentity name="YamlDotNet" publicKeyToken="ec19458f3c15af5e" culture="neutral" />
         <bindingRedirect oldVersion="0.0.0.0-6.0.0.0" newVersion="6.0.0.0" />
       </dependentAssembly>
     </assemblyBinding>
     <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
       <dependentAssembly>
         <assemblyIdentity name="Microsoft.Cloud.InstrumentationFramework.Metrics" publicKeyToken="31bf3856ad364e35" culture="neutral" />
         <bindingRedirect oldVersion="0.0.0.0-3.0.0.0" newVersion="3.0.0.0" />
       </dependentAssembly>
     </assemblyBinding>
     <enforceFIPSPolicy enabled="false"/>
   </runtime>
 </configuration>
     ```

  3. Save the file, and then restart the Self-hosted IR machine.

Error code: JniException

- **Message**: `An error occurred when invoking Java Native Interface.`
- **Cause**: If the error message contains "Cannot create JVM: JNI return code [-6][JNI call failed: Invalid arguments.]", the possible cause is that JVM can't be created because some illegal (global) arguments are set.
- **Recommendation**: Log in to the machine that hosts *each node* of your self-hosted integration runtime. Check to ensure that the system variable is set correctly, as follows: `_JAVA_OPTIONS "-Xms256m -Xmx16g" with memory bigger than 8G`. Restart all the integration runtime nodes, and then rerun the pipeline.

Error code: GetOAuth2AccessTokenErrorResponse

- **Message**:

  ```
  Failed to get access token from your token endpoint. Error returned from your authorization server: %errorResponse;.
  ```

- **Cause**: Your client ID or client secret is invalid, and the authentication failed in your authorization server.
- **Recommendation**: Correct all OAuth2 client credential flow settings of your authorization server.

Error code: FailedToGetOAuth2AccessToken

- **Message**: `Failed to get access token from your token endpoint. Error message: %errorMessage;.`
- **Cause**: OAuth2 client credential flow settings are invalid.
- **Recommendation**: Correct all OAuth2 client credential flow settings of your authorization server.

Error code: OAuth2AccessTokenTypeNotSupported

- **Message**:
  ```
  The toke type '%tokenType;' from your authorization server is not supported, supported types: '%tokenTypes;'.
  ```
- **Cause**: Your authorization server is not supported.
- **Recommendation**: Use an authorization server that can return tokens with supported token types.

Error code: OAuth2ClientIdColonNotAllowed

- **Message**:
  ```
  The character colon(:) is not allowed in clientId for OAuth2ClientCredential authentication.
  ```
- **Cause**: Your client ID includes the invalid character colon (`:`).
- **Recommendation**: Use a valid client ID.

Error code: ManagedIdentityCredentialObjectNotSupported

- **Message**:
  ```
  Managed identity credential is not supported in this version ('%version;') of Self Hosted Integration Runtime.
  ```
- **Recommendation**: Check the supported version and upgrade the integration runtime to a higher version.

Error code: QueryMissingFormatSettingsInDataset

- **Message**: `The format settings are missing in dataset %dataSetName;.`
- **Cause**: The dataset type is Binary, which is not supported.
- **Recommendation**: Use the DelimitedText, Json, Avro, Orc, or Parquet dataset instead.
- **Cause**: For the file storage, the format settings are missing in the dataset.
- **Recommendation**: Deselect the "Binary copy" in the dataset, and set correct format settings.

Error code: QueryUnsupportedCommandBehavior

- **Message**: `The command behavior "%behavior;" is not supported.`
- **Recommendation**: Don't add the command behavior as a parameter for preview or GetSchema API request URL.

Error code: DataConsistencyFailedToGetSourceFileMetadata

- **Message**: `Failed to retrieve source file ('%name;') metadata to validate data consistency.`
- **Cause**: There is a transient issue on the sink data store, or retrieving metadata from the sink data store is not allowed.

Error code: DataConsistencyFailedToGetSinkFileMetadata

- **Message**: `Failed to retrieve sink file ('%name;') metadata to validate data consistency.`
- **Cause**: There is a transient issue on the sink data store, or retrieving metadata from the sink data store is not allowed.

Error code: DataConsistencyValidationNotSupportedForNonDirectBinaryCopy

- **Message**: `Data consistency validation is not supported in current copy activity settings.`
- **Cause**: The data consistency validation is only supported in the direct binary copy scenario.
- **Recommendation**: Remove the 'validateDataConsistency' property in the copy activity payload.

Error code: DataConsistencyValidationNotSupportedForLowVersionSelfHostedIntegrationRuntime

- **Message**:
  ```
  'validateDataConsistency' is not supported in this version ('%version;') of Self Hosted Integration Runtime.
  ```
- **Recommendation**: Check the supported integration runtime version and upgrade it to a higher version, or remove the 'validateDataConsistency' property from copy activities.

Error code: SkipMissingFileNotSupportedForNonDirectBinaryCopy

- **Message**:
  ```
  Skip missing file is not supported in current copy activity settings, it's only supported with direct binary copy with folder.
  ```
- **Recommendation**: Remove 'fileMissing' of the skipErrorFile setting in the copy activity payload.

Error code: SkipInconsistencyDataNotSupportedForNonDirectBinaryCopy

- **Message**:
  ```
  Skip inconsistency is not supported in current copy activity settings, it's only supported with direct binary copy when validateDataConsistency is true.
  ```
- **Recommendation**: Remove 'dataInconsistency' of the skipErrorFile setting in the copy activity payload.

Error code: SkipForbiddenFileNotSupportedForNonDirectBinaryCopy

- **Message**:
  ```
  Skip forbidden file is not supported in current copy activity settings, it's only supported with direct binary copy with folder.
  ```
- **Recommendation**: Remove 'fileForbidden' of the skipErrorFile setting in the copy activity payload.

Error code: SkipForbiddenFileNotSupportedForThisConnector

- **Message**: `Skip forbidden file is not supported for this connector: ('%connectorName;').`
- **Recommendation**: Remove 'fileForbidden' of the skipErrorFile setting in the copy activity payload.

Error code: SkipInvalidFileNameNotSupportedForNonDirectBinaryCopy

- **Message**:
  ```
  Skip invalid file name is not supported in current copy activity settings, it's only supported with direct binary copy with folder.
  ```
- **Recommendation**: Remove 'invalidFileName' of the skipErrorFile setting in the copy activity payload.

Error code: SkipInvalidFileNameNotSupportedForSource

- **Message**: `Skip invalid file name is not supported for '%connectorName;' source.`
- **Recommendation**: Remove 'invalidFileName' of the skipErrorFile setting in the copy activity payload.

Error code: SkipInvalidFileNameNotSupportedForSink

- **Message**: `Skip invalid file name is not supported for '%connectorName;' sink.`
- **Recommendation**: Remove 'invalidFileName' of the skipErrorFile setting in the copy activity payload.

Error code: SkipAllErrorFileNotSupportedForNonBinaryCopy

- **Message**:

```
Skip all error file is not supported in current copy activity settings, it's only supported with
binary copy with folder.
```

- **Recommendation**: Remove 'allErrorFile' in the skipErrorFile setting in the copy activity payload.

Error code: DeleteFilesAfterCompletionNotSupportedForNonDirectBinaryCopy

- **Message**:
  ```
  'deleteFilesAfterCompletion' is not support in current copy activity settings, it's only supported
  with direct binary copy.
  ```
- **Recommendation**: Remove the 'deleteFilesAfterCompletion' setting or use direct binary copy.

Error code: DeleteFilesAfterCompletionNotSupportedForThisConnector

- **Message**: `'deleteFilesAfterCompletion' is not supported for this connector: ('%connectorName;').`
- **Recommendation**: Remove the 'deleteFilesAfterCompletion' setting in the copy activity payload.

Error code: FailedToDownloadCustomPlugins

- **Message**: `Failed to download custom plugins.`
- **Cause**: Invalid download links or transient connectivity issues.
- **Recommendation**: Retry if the message shows that it's a transient issue. If the problem persists, contact the support team.

General connector errors

Error code: UserErrorOdbcInvalidQueryString

- **Message**: `The following ODBC Query is not valid: '%'.`
- **Cause**: You provide a wrong or invalid query to fetch the data/schemas.
- **Recommendation**: Verify your query is valid and can return data/schemas. Use Script activity if you want to execute non-query scripts and your data store is supported. Alternatively, consider to use stored procedure that returns a dummy result to execute your non-query scripts.

Next steps

For more troubleshooting help, try these resources:

- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Azure Blob Storage connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Azure Blob Storage connector in Azure Data Factory and Azure Synapse.

Error code: AzureBlobOperationFailed

- **Message**: "Blob operation Failed. ContainerName: %containerName;, path: %path;."
- **Cause**: A problem with the Blob Storage operation.
- **Recommendation**: To check the error details, see Blob Storage error codes. For further help, contact the Blob Storage team.

Invalid property during copy activity

- **Message**:

```
Copy activity \<Activity Name> has an invalid "source" property. The source type is not compatible with the dataset \<Dataset Name> and its linked service \<Linked Service Name>. Please verify your input against.
```

- **Cause**: The type defined in the dataset is inconsistent with the source or sink type that's defined in the copy activity.
- **Resolution**: Edit the dataset or pipeline JSON definition to make the types consistent, and then rerun the deployment.

Error code: FIPSModeIsNotSupport

- **Message**:

```
Fail to read data form Azure Blob Storage for Azure Blob connector needs MD5 algorithm which can't co-work with FIPS mode. Please change diawp.exe.config in self-hosted integration runtime install directory to disable FIPS policy following https://docs.microsoft.com/dotnet/framework/configure-apps/file-schema/runtime/enforcefipspolicy-element.
```

- **Cause**: Then FIPS policy is enabled on the VM where the self-hosted integration runtime was installed.
- **Recommendation**: Disable the FIPS mode on the VM where the self-hosted integration runtime was installed. Windows doesn't recommend the FIPS mode.

Error code: AzureBlobInvalidBlockSize

- **Message**: `Block size should between %minSize; MB and 100 MB.`
- **Cause**: The block size is over the blob limitation.

Error code: AzureStorageOperationFailedConcurrentWrite

- **Message**:

```
Error occurred when trying to upload a file. It's possible because you have multiple concurrent copy activities runs writing to the same file '%name;'. Check your ADF configuration.
```

- **Cause**: You have multiple concurrent copy activity runs or applications writing to the same file.

Error code: AzureAppendBlobConcurrentWriteConflict

- **Message**:

```
Detected concurrent write to the same append blob file, it's possible because you have multiple concurrent copy activities runs or applications writing to the same file '%name;'. Please check your ADF configuration and retry.
```

- **Cause**: Multiple concurrent writing requests occur, which causes conflicts on file content.

Error code: AzureBlobFailedToCreateContainer

- **Message**:

```
Unable to create Azure Blob container. Endpoint: '%endpoint;', Container Name: '%containerName;'.
```

- **Cause**: This error happens when copying data with Azure Blob Storage account public access.
- **Recommendation**: For more information about connection errors in the public endpoint, see Connection error in public endpoint.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Azure Cosmos DB connector in Azure Data Factory and Azure Synapse

8/19/2022 • 3 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Azure Cosmos DB and Azure Cosmos DB (SQL API) connectors in Azure Data Factory and Azure Synapse.

Error message: Request size is too large

- **Symptoms**: When you copy data into Azure Cosmos DB with a default write batch size, you receive the following error: `Request size is too large.`
- **Cause**: Azure Cosmos DB limits the size of a single request to 2 MB. The formula is *request size = single document size * write batch size*. If your document size is large, the default behavior will result in a request size that's too large. You can tune the write batch size.
- **Resolution**: In the copy activity sink, reduce the *write batch size* value (the default value is 10000).

Error message: Unique index constraint violation

- **Symptoms**: When you copy data into Azure Cosmos DB, you receive the following error:

```
Message=Partition range id 0 | Failed to import mini-batch. Exception was Message: {"Errors":
["Encountered exception while executing function. Exception = Error: {\"Errors\":[\"Unique index
constraint violation.\"]}...
```

- **Cause**: There are two possible causes:
  - Cause 1: If you use **Insert** as the write behavior, this error means that your source data has rows or objects with same ID.
  - Cause 2: If you use **Upsert** as the write behavior and you set another unique key to the container, this error means that your source data has rows or objects with different IDs but the same value for the defined unique key.
- **Resolution**:
  - For cause 1, set **Upsert** as the write behavior.
  - For cause 2, make sure that each document has a different value for the defined unique key.

Error message: Request rate is large

- **Symptoms**: When you copy data into Azure Cosmos DB, you receive the following error:

```
Type=Microsoft.Azure.Documents.DocumentClientException, Message=Message: {"Errors":["Request rate is
large"]}
```

- **Cause**: The number of used request units (RUs) is greater than the available RUs configured in Azure Cosmos DB. To learn how Azure Cosmos DB calculates RUs, see Request units in Azure Cosmos DB.
- **Resolution**: Try either of the following two solutions:
  - Increase the *container RUs* number to a greater value in Azure Cosmos DB. This solution will improve the copy activity performance, but it will incur more cost in Azure Cosmos DB.

- Decrease *writeBatchSize* to a lesser value, such as 1000, and decrease *parallelCopies* to a lesser value, such as 1. This solution will reduce copy run performance, but it won't incur more cost in Azure Cosmos DB.

Columns missing in column mapping

- **Symptoms**: When you import a schema for Azure Cosmos DB for column mapping, some columns are missing.
- **Cause**: Azure Data Factory and Synapse pipelines infer the schema from the first 10 Azure Cosmos DB documents. If some document columns or properties don't contain values, the schema isn't detected and consequently isn't displayed.
- **Resolution**: You can tune the query as shown in the following code to force the column values to be displayed in the result set with empty values. Assume that the *impossible* column is missing in the first 10 documents). Alternatively, you can manually add the column for mapping.

```
select c.company, c.category, c.comments, (c.impossible??'') as impossible from c
```

Error message: The GuidRepresentation for the reader is CSharpLegacy

- **Symptoms**: When you copy data from Azure Cosmos DB MongoAPI or MongoDB with the universally unique identifier (UUID) field, you receive the following error:

```
Failed to read data via MongoDB client.,
Source=Microsoft.DataTransfer.Runtime.MongoDbV2Connector,Type=System.FormatException, Message=The
GuidRepresentation for the reader is CSharpLegacy which requires the binary sub type to be UuidLegacy
not UuidStandard.,Source=MongoDB.Bson,’“,
```

- **Cause**: There are two ways to represent the UUID in Binary JSON (BSON): UuidStardard and UuidLegacy. By default, UuidLegacy is used to read data. You will receive an error if your UUID data in MongoDB is UuidStandard.
- **Resolution**: In the MongoDB connection string, add the *uuidRepresentation=standard* option. For more information, see MongoDB connection string.

Error code: CosmosDbSqlApiOperationFailed

- **Message**: `CosmosDbSqlApi operation Failed. ErrorMessage: %msg;.`
- **Cause**: A problem with the CosmosDbSqlApi operation. This applies to the Cosmos DB (SQL API) connector specifically.
- **Recommendation**: To check the error details, see Azure Cosmos DB help document. For further help, contact the Azure Cosmos DB team.

Error code: CosmosDbSqlApiPartitionKeyExceedStorage

- **Message**:

```
The size of data each logical partition can store is limited, current partitioning design and workload
failed to store more than the allowed amount of data for a given partition key value.
```

- **Cause**: The data size of each logical partition is limited, and the partition key reached the maximum size of your logical partition.
- **Recommendation**: Check your Azure Cosmos DB partition design. For more information, see Logical

partitions.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Azure Data Explorer connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Azure Data Explorer connector in Azure Data Factory and Azure Synapse.

Error code: KustoMappingReferenceHasWrongKind

- **Message**: `Mapping reference should be of kind 'Csv'. Mapping reference: '%reference;'. Kind '%kind;'.`
- **Cause**: The ingestion mapping reference is not CSV type.
- **Recommendation**: Create a CSV ingestion mapping reference.

Error code: KustoWriteFailed

- **Message**: `Write to Kusto failed with following error: '%message;'.`
- **Cause**: Wrong configuration or transient errors when the sink reads data from the source.
- **Recommendation**: For transient failures, set retries for the activity. For permanent failures, check your configuration and contact support.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Azure Data Lake Storage connectors in Azure Data Factory and Azure Synapse

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Azure Data Lake Storage Gen1 and Gen2 connectors in Azure Data Factory and Azure Synapse.

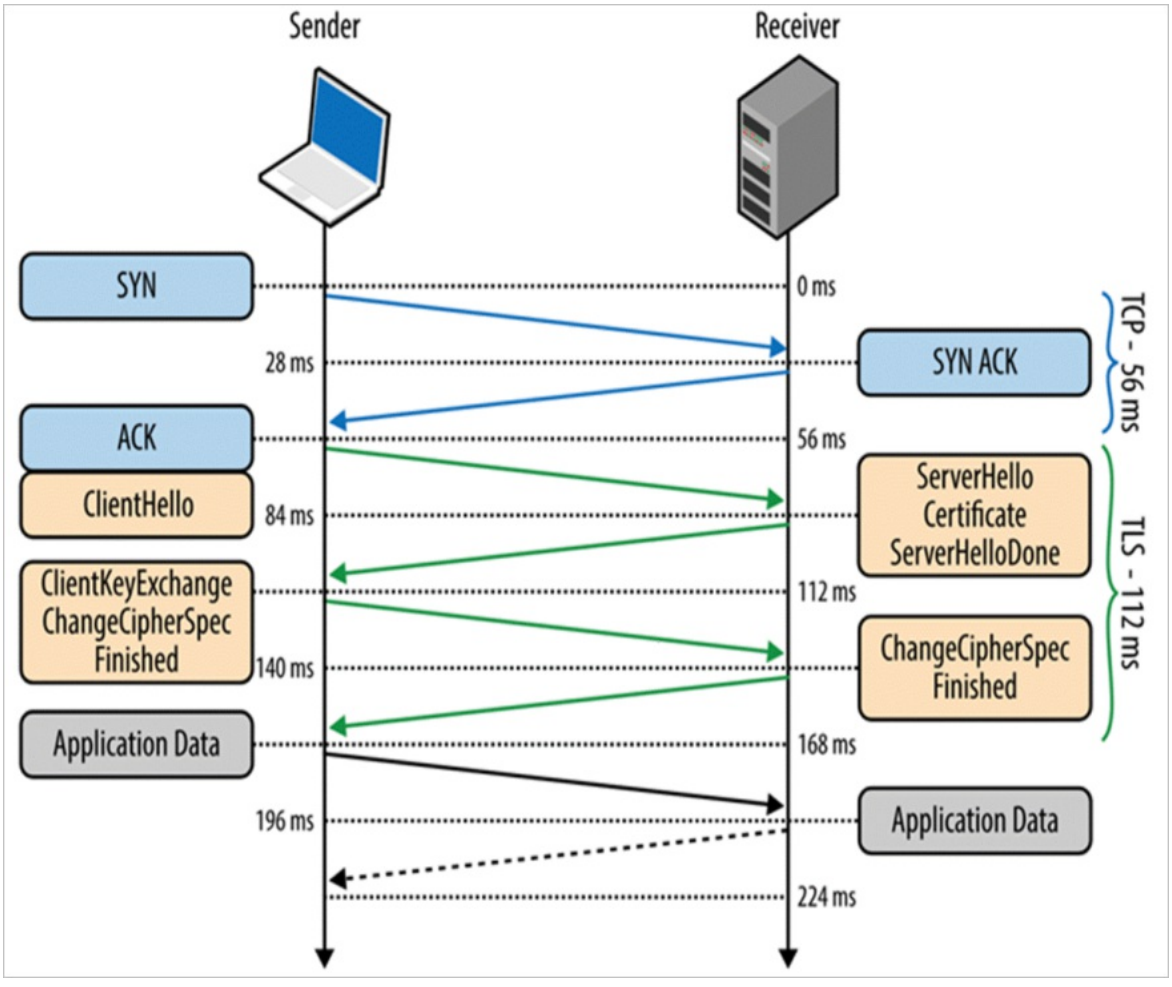
Azure Data Lake Storage Gen1

Error message: The underlying connection was closed: Could not establish trust relationship for the SSL/TLS secure channel.

- **Symptoms**: Copy activity fails with the following error:

```
Message: ErrorCode = UserErrorFailedFileOperation, Error Message = The underlying connection was
closed: Could not establish trust relationship for the SSL/TLS secure channel.
```

- **Cause**: The certificate validation failed during the TLS handshake.
- **Resolution**: As a workaround, use the staged copy to skip the Transport Layer Security (TLS) validation for Azure Data Lake Storage Gen1. You need to reproduce this issue and gather the network monitor (netmon) trace, and then engage your network team to check the local network configuration.



Error message: The remote server returned an error: (403) Forbidden

- **Symptoms**: Copy activity fail with the following error:

```
Message: The remote server returned an error: (403) Forbidden. Response details: {"RemoteException":
{"exception":"AccessControlException""message":"CREATE failed with error 0x83090aa2 (Forbidden. ACL
verification failed. Either the resource does not exist or the user is not authorized to perform the
requested operation.)....
```

- **Cause**: One possible cause is that the service principal or managed identity you use doesn't have permission to access certain folders or files.
- **Resolution**: Grant appropriate permissions to all the folders and subfolders you need to copy. For more information, see Copy data to or from Azure Data Lake Storage Gen1.

Error message: Failed to get access token by using service principal. ADAL Error: service_unavailable

- **Symptoms**: Copy activity fails with the following error:

```
Failed to get access token by using service principal. ADAL Error: service_unavailable, The remote
server returned an error: (503) Server Unavailable.
```

- **Cause**: When the Service Token Server (STS) that's owned by Azure Active Directory is not available, that means it's too busy to handle requests, and it returns HTTP error 503.
- **Resolution**: Rerun the copy activity after several minutes.

Azure Data Lake Storage Gen2

Error code: ADLSGen2OperationFailed

- **Message**: `ADLS Gen2 operation failed for: %adlsGen2Message;.%exceptionData;.`
- **Causes and recommendations**: Different causes may lead to this error. Check below list for possible cause analysis and related recommendation.

CAUSE ANALYSIS	RECOMMENDATION
If Azure Data Lake Storage Gen2 throws error indicating some operation failed.	Check the detailed error message thrown by Azure Data Lake Storage Gen2. If the error is a transient failure, retry the operation. For further help, contact Azure Storage support, and provide the request ID in error message.
If the error message contains the string "Forbidden", the service principal or managed identity you use might not have sufficient permission to access Azure Data Lake Storage Gen2.	To troubleshoot this error, see Copy and transform data in Azure Data Lake Storage Gen2.
If the error message contains the string "InternalServerError", the error is returned by Azure Data Lake Storage Gen2.	The error might be caused by a transient failure. If so, retry the operation. If the issue persists, contact Azure Storage support and provide the request ID from the error message.
If the error message is `Unable to read data from the transport connection: An existing connection was forcibly closed by the remote host`, your integration runtime has a network issue in connecting to Azure Data Lake Storage Gen2.	In the firewall rule setting of Azure Data Lake Storage Gen2, make sure Azure Data Factory IP addresses are in the allowed list. For more information, see Configure Azure Storage firewalls and virtual networks.

CAUSE ANALYSIS	RECOMMENDATION
If the error message is `This endpoint does not support BlobStorageEvents or SoftDelete`, you are using an Azure Data Lake Storage Gen2 linked service to connect to an Azure Blob Storage account that enables Blob storage events or soft delete.	Try the following options: 1. If you still want to use an Azure Data Lake Storage Gen2 linked service, upgrade your Azure Blob Storage to Azure Data Lake Storage Gen2. For more information, see Upgrade Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities. 2. Switch your linked service to Azure Blob Storage. 3. Disable Blob storage events or soft delete in your Azure Blob Storage account.

Request to Azure Data Lake Storage Gen2 account caused a timeout error

- **Message**:
  - Error Code = `UserErrorFailedBlobFSOperation`
  - Error Message = `BlobFS operation failed for: A task was canceled.`
- **Cause**: The issue is caused by the Azure Data Lake Storage Gen2 sink timeout error, which usually occurs on the Self-hosted Integration Runtime (IR) machine.
- **Recommendation**:
  - Place your Self-hosted IR machine and target Azure Data Lake Storage Gen2 account in the same region, if possible. This can help avoid a random timeout error and produce better performance.
  - Check whether there's a special network setting, such as ExpressRoute, and ensure that the network has enough bandwidth. We suggest that you lower the Self-hosted IR concurrent jobs setting when the overall bandwidth is low. Doing so can help avoid network resource competition across multiple concurrent jobs.
  - If the file size is moderate or small, use a smaller block size for nonbinary copy to mitigate such a timeout error. For more information, see Blob Storage Put Block.

    To specify the custom block size, edit the property in your JSON file editor as shown here:

    ```
    "sink": {
        "type": "DelimitedTextSink",
        "storeSettings": {
            "type": "AzureBlobFSWriteSettings",
            "blockSizeInMB": 8
        }
    }
    ```

The copy activity is not able to pick files from Azure Data Lake Storage Gen2

- **Symptoms**: The copy activity is not able to pick files from Azure Data Lake Storage Gen2 when the file name is "Asset_Metadata". The issue only occurs in the Parquet type dataset. Other types of datasets with the same file name work correctly.
- **Cause**: For the backward compatibility, `_metadata` is treated as a reserved substring in the file name.
- **Recommendation**: Change the file name to avoid the reserved list for Parquet below:
  1. The file name contains `_metadata`.
  2. The file name starts with `.` (dot).

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Azure Database for PostgreSQL connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Azure Database for PostgreSQL connector in Azure Data Factory and Azure Synapse.

Error code: AzurePostgreSqlNpgsqlDataTypeNotSupported

- **Message**:

  ```
  The data type of the chosen Partition Column, '%partitionColumn;', is '%dataType;' and this data type is not supported for partitioning.
  ```

- **Recommendation**: Pick a partition column with int, bigint, smallint, serial, bigserial, smallserial, timestamp with or without time zone, time without time zone or date data type.

Error code: AzurePostgreSqlNpgsqlPartitionColumnNameNotProvided

- **Message**: `Partition column name must be specified.`
- **Cause**: No partition column name is provided, and it couldn't be decided automatically.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Azure Files connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Azure Files connector in Azure Data Factory and Azure Synapse.

Error code: AzureFileOperationFailed

- **Message**: `Azure File operation Failed. Path: %path;. ErrorMessage: %msg;.`
- **Cause**: A problem with the Azure Files storage operation.
- **Recommendation**: To check the error details, see Azure Files help. For further help, contact the Azure Files team.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Azure Synapse Analytics, Azure SQL Database, and SQL Server connectors in Azure Data Factory and Azure Synapse

8/19/2022 • 11 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Azure Synapse Analytics, Azure SQL Database, and SQL Server connectors in Azure Data Factory and Azure Synapse.

Error code: SqlFailedToConnect

- **Message**:

```
Cannot connect to SQL Database: '%server;', Database: '%database;', User: '%user;'. Check the linked service configuration is correct, and make sure the SQL Database firewall allows the integration runtime to access.
```

- **Causes and recommendations**: Different causes may lead to this error. Check below list for possible cause analysis and related recommendation.

CAUSE ANALYSIS	RECOMMENDATION
For Azure SQL, if the error message contains the string "SqlErrorNumber=47073", it means that public network access is denied in the connectivity setting.	On the Azure SQL firewall, set the **Deny public network access** option to *No*. For more information, see Azure SQL connectivity settings.
For Azure SQL, if the error message contains an SQL error code such as "SqlErrorNumber=[errorcode]", see the Azure SQL troubleshooting guide.	For a recommendation, see Troubleshoot connectivity issues and other errors with Azure SQL Database and Azure SQL Managed Instance.
Check to see whether port 1433 is in the firewall allowlist.	For more information, see Ports used by SQL Server.
If the error message contains the string "SqlException", SQL Database the error indicates that some specific operation failed.	For more information, search by SQL error code in Database engine errors. For further help, contact Azure SQL support.
If this is a transient issue (for example, an instable network connection), add retry in the activity policy to mitigate.	For more information, see Pipelines and activities.
If the error message contains the string "Client with IP address '...' is not allowed to access the server", and you're trying to connect to Azure SQL Database, the error is usually caused by an Azure SQL Database firewall issue.	In the Azure SQL Server firewall configuration, enable the **Allow Azure services and resources to access this server** option. For more information, see Azure SQL Database and Azure Synapse IP firewall rules.

Error code: SqlOperationFailed

- **Message**: `A database operation failed. Please search error to get more details.`
- **Causes and recommendations**: Different causes may lead to this error. Check below list for possible

cause analysis and related recommendation.

CAUSE ANALYSIS	RECOMMENDATION
If the error message contains the string "SqlException", SQL Database throws an error indicating some specific operation failed.	If the SQL error is not clear, try to alter the database to the latest compatibility level '150'. It can throw the latest version SQL errors. For more information, see the documentation. For more information about troubleshooting SQL issues, search by SQL error code in Database engine errors. For further help, contact Azure SQL support.
If the error message contains the string "PdwManagedToNativeInteropException", it's usually caused by a mismatch between the source and sink column sizes.	Check the size of both the source and sink columns. For further help, contact Azure SQL support.
If the error message contains the string "InvalidOperationException", it's usually caused by invalid input data.	To identify which row has encountered the problem, enable the fault tolerance feature on the copy activity, which can redirect problematic rows to the storage for further investigation. For more information, see Fault tolerance of copy activity.
If the error message contains "Execution Timeout Expired", it's usually caused by query timeout.	Configure **Query timeout** in the source and **Write batch timeout** in the sink to increase timeout.

Error code: SqlUnauthorizedAccess

- **Message**: `Cannot connect to '%connectorName;'. Detail Message: '%message;'`
- **Cause**: The credentials are incorrect or the login account can't access the SQL database.
- **Recommendation**: Check to ensure that the login account has sufficient permissions to access the SQL database.

Error code: SqlOpenConnectionTimeout

- **Message**: `Open connection to database timeout after '%timeoutValue;' seconds.`
- **Cause**: The problem could be a SQL database transient failure.
- **Recommendation**: Retry the operation to update the linked service connection string with a larger connection timeout value.

Error code: SqlAutoCreateTableTypeMapFailed

- **Message**:
  ```
  Type '%dataType;' in source side cannot be mapped to a type that supported by sink side(column name:'%columnName;') in autocreate table.
  ```
- **Cause**: The autocreation table can't meet the source requirement.
- **Recommendation**: Update the column type in *mappings*, or manually create the sink table in the target server.

Error code: SqlDataTypeNotSupported

- **Message**: `A database operation failed. Check the SQL errors.`

- **Cause**: If the issue occurs in the SQL source and the error is related to SqlDateTime overflow, the data value exceeds the logic type range (1/1/1753 12:00:00 AM - 12/31/9999 11:59:59 PM).
- **Recommendation**: Cast the type to the string in the source SQL query or, in the copy activity column mapping, change the column type to *String*.
- **Cause**: If the issue occurs on the SQL sink and the error is related to SqlDateTime overflow, the data value exceeds the allowed range in the sink table.
- **Recommendation**: Update the corresponding column type to the *datetime2* type in the sink table.

Error code: SqlInvalidDbStoredProcedure

- **Message**:
  ```
  The specified Stored Procedure is not valid. It could be caused by that the stored procedure doesn't return any data. Invalid Stored Procedure script: '%scriptName;'.
  ```
- **Causes and recommendations**: Different causes may lead to this error. Check below list for possible cause analysis and related recommendation.

CAUSE ANALYSIS	RECOMMENDATION
The specified stored procedure is invalid.	Validate the stored procedure by using SQL Tools. Make sure that the stored procedure can return data.
The Lookup activity requires the stored procedure to return some value, but the stored procedure code doesn't return any value.	Use the Stored Procedure Activity if the stored procedure is expected to return no data.

Error code: SqlInvalidDbQueryString

- **Message**:
  ```
  The specified SQL Query is not valid. It could be caused by that the query doesn't return any data. Invalid query: '%query;'
  ```
- **Cause**: The specified SQL query is invalid. The cause might be that the query doesn't return any data.
- **Recommendation**: Validate the SQL query by using SQL Tools. Make sure that the query can return data.

Error code: SqlInvalidColumnName

- **Message**:
  ```
  Column '%column;' does not exist in the table '%tableName;', ServerName: '%serverName;', DatabaseName: '%dbName;'.
  ```
- **Cause**: The column can't be found because the configuration might be incorrect.
- **Recommendation**: Verify the column in the query, *structure* in the dataset, and *mappings* in the activity.

Error code: SqlBatchWriteTimeout

- **Message**: `Timeouts in SQL write operation.`
- **Cause**: The problem could be caused by a SQL database transient failure.
- **Recommendation**: Retry the operation. If the problem persists, contact Azure SQL support.

Error code: SqlBatchWriteTransactionFailed

- **Message**: `SQL transaction commits failed.`
- **Cause**: If exception details constantly indicate a transaction timeout, the network latency between the integration runtime and the database is greater than the default threshold of 30 seconds.
- **Recommendation**: Update the SQL-linked service connection string with a *connection timeout* value that's equal to or greater than 120 and rerun the activity.
- **Cause**: If the exception details intermittently indicate that the SQL connection is broken, it might be a transient network failure or a SQL database side issue.
- **Recommendation**: Retry the activity and review the SQL database side metrics.

Error code: SqlBulkCopyInvalidColumnLength

- **Message**: `SQL Bulk Copy failed due to receive an invalid column length from the bcp client.`
- **Cause**: SQL Bulk Copy failed because it received an invalid column length from the bulk copy program utility (bcp) client.
- **Recommendation**: To identify which row has encountered the problem, enable the fault tolerance feature on the copy activity. This can redirect problematic rows to the storage for further investigation. For more information, see Fault tolerance of copy activity.

Error code: SqlConnectionIsClosed

- **Message**: `The connection is closed by SQL Database.`
- **Cause**: The SQL connection is closed by the SQL database when a high concurrent run and the server terminate the connection.
- **Recommendation**: Retry the connection. If the problem persists, contact Azure SQL support.

Error code: SqlServerInvalidLinkedServiceCredentialMissing

- **Message**: `The SQL Server linked service is invalid with its credential being missing.`
- **Cause**: The linked service was not configured properly.
- **Recommendation**: Validate and fix the SQL server linked service.

Error code: SqlParallelFailedToDetectPartitionColumn

- **Message**: `Failed to detect the partition column with command '%command;', %message;.`
- **Cause**: There is no primary key or unique key in the table.
- **Recommendation**: Check the table to make sure that a primary key or a unique index is created.

Error code: SqlParallelFailedToDetectPhysicalPartitions

- **Message**: `Failed to detect the physical partitions with command '%command;', %message;.`
- **Cause**: No physical partitions are created for the table. Check your database.
- **Recommendation**: Reference Create Partitioned Tables and Indexes to solve this issue.

Error code: SqlParallelFailedToGetPartitionRangeSynapse

- **Message**: `Failed to get the partitions for azure synapse with command '%command;', %message;.`
- **Cause**: No physical partitions are created for the table. Check your database.
- **Recommendation**: Reference Partitioning tables in dedicated SQL pool to solve this issue.

Error message: Conversion failed when converting from a character string to uniqueidentifier

- **Symptoms**: When you copy data from a tabular data source (such as SQL Server) into Azure Synapse Analytics using staged copy and PolyBase, you receive the following error:

```
ErrorCode=FailedDbOperation,Type=Microsoft.DataTransfer.Common.Shared.HybridDeliveryException,
Message=Error happened when loading data into Azure Synapse Analytics.,
Source=Microsoft.DataTransfer.ClientLibrary,Type=System.Data.SqlClient.SqlException,
Message=Conversion failed when converting from a character string to uniqueidentifier...
```

- **Cause**: Azure Synapse Analytics PolyBase can't convert an empty string to a GUID.
- **Resolution**: In the copy activity sink, under PolyBase settings, set the **use type default** option to *false*.

Error message: Expected data type: DECIMAL(x,x), Offending value

- **Symptoms**: When you copy data from a tabular data source (such as SQL Server) into Azure Synapse Analytics by using staged copy and PolyBase, you receive the following error:

```
ErrorCode=FailedDbOperation,Type=Microsoft.DataTransfer.Common.Shared.HybridDeliveryException,
Message=Error happened when loading data into Azure Synapse Analytics.,
Source=Microsoft.DataTransfer.ClientLibrary,Type=System.Data.SqlClient.SqlException, Message=Query
aborted-- the maximum reject threshold (0 rows) was reached while reading from an external source: 1
rows rejected out of total 415 rows processed. (/file_name.txt) Column ordinal: 18, Expected data
type: DECIMAL(x,x), Offending value:..
```

- **Cause**: Azure Synapse Analytics PolyBase can't insert an empty string (null value) into a decimal column.
- **Resolution**: In the copy activity sink, under PolyBase settings, set the **use type default** option to false.

Error message: Java exception message: HdfsBridge::CreateRecordReader

- **Symptoms**: You copy data into Azure Synapse Analytics by using PolyBase and receive the following error:

```
Message=110802;An internal DMS error occurred that caused this operation to fail. Details: Exception:
Microsoft.SqlServer.DataWarehouse.DataMovement.Common.ExternalAccess.HdfsAccessException, Message:
Java exception raised on call to HdfsBridge_CreateRecordReader. Java exception
message:HdfsBridge::CreateRecordReader - Unexpected error encountered creating the record reader.:
Error [HdfsBridge::CreateRecordReader - Unexpected error encountered creating the record reader.]
occurred while accessing external file.....
```

- **Cause**: The cause might be that the schema (total column width) is too large (larger than 1 MB). Check the schema of the target Azure Synapse Analytics table by adding the size of all columns:
  - Int = 4 bytes
  - Bigint = 8 bytes
  - Varchar(n), char(n), binary(n), varbinary(n) = n bytes
  - Nvarchar(n), nchar(n) = n*2 bytes
  - Date = 6 bytes
  - Datetime/(2), smalldatetime = 16 bytes

    - Datetimeoffset = 20 bytes
    - Decimal = 19 bytes
    - Float = 8 bytes
    - Money = 8 bytes
    - Smallmoney = 4 bytes
    - Real = 4 bytes
    - Smallint = 2 bytes
    - Time = 12 bytes
    - Tinyint = 1 byte
- **Resolution**:
    - Reduce column width to less than 1 MB.
    - Or use a bulk insert approach by disabling PolyBase.

Error message: The condition specified using HTTP conditional header(s) is not met

- **Symptoms**: You use SQL query to pull data from Azure Synapse Analytics and receive the following error:

  ```
  ...StorageException: The condition specified using HTTP conditional header(s) is not met...
  ```

- **Cause**: Azure Synapse Analytics encountered an issue while querying the external table in Azure Storage.
- **Resolution**: Run the same query in SQL Server Management Studio (SSMS) and check to see whether you get the same result. If you do, open a support ticket to Azure Synapse Analytics and provide your Azure Synapse Analytics server and database name.

Performance tier is low and leads to copy failure

- **Symptoms**: You copy data into Azure SQL Database and receive the following error:

  ```
  Database operation failed. Error message from database execution : ExecuteNonQuery requires an open and available Connection. The connection's current state is closed.
  ```

- **Cause**: Azure SQL Database s1 has hit input/output (I/O) limits.
- **Resolution**: Upgrade the Azure SQL Database performance tier to fix the issue.

SQL table can't be found

- **Symptoms**: You copy data from hybrid into an on-premises SQL Server table and receive the following error: `Cannot find the object "dbo.Contoso" because it does not exist or you do not have permissions.`
- **Cause**: The current SQL account doesn't have sufficient permissions to execute requests issued by .NET SqlBulkCopy.WriteToServer.
- **Resolution**: Switch to a more privileged SQL account.

Error message: String or binary data is truncated

- **Symptoms**: An error occurs when you copy data into an on-premises Azure SQL Server table.
- **Cause**: The Cx SQL table schema definition has one or more columns with less length than expected.
- **Resolution**: To resolve the issue, try the following:

1. To troubleshoot which rows have the issue, apply SQL sink fault tolerance, especially "redirectIncompatibleRowSettings."

   > **NOTE**
   > Fault tolerance might require additional execution time, which could lead to higher costs.

2. Double-check the redirected data against the SQL table schema column length to see which columns need to be updated.
3. Update the table schema accordingly.

Error code: FailedDbOperation

- **Message**: `User does not have permission to perform this action.`
- **Recommendation**: Make sure the user configured in the Azure Synapse Analytics connector must have 'CONTROL' permission on the target database while using PolyBase to load data. For more detailed information, refer to this document.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Azure Table Storage connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Azure Table Storage connector in Azure Data Factory and Azure Synapse.

Error code: AzureTableDuplicateColumnsFromSource

- **Message**:

  ```
  Duplicate columns with same name '%name;' are detected from source. This is NOT supported by Azure Table Storage sink.
  ```

- **Cause**: Duplicated source columns might occur for one of the following reasons:
  - You're using the database as a source and applied table joins.
  - You have unstructured CSV files with duplicated column names in the header row.
- **Recommendation**: Double-check and fix the source columns, as necessary.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the DB2 connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Azure Database for PostgreSQL connector in Azure Data Factory and Azure Synapse.

Error code: DB2DriverRunFailed

- **Message**: `Error thrown from driver. Sql code: '%code;'`
- **Cause**: If the error message contains the string "SQLSTATE=51002 SQLCODE=-805", follow the "Tip" in Copy data from DB2.
- **Recommendation**: Try to set "NULLID" in the `packageCollection` property.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the delimited text format connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the delimited text formatL connector in Azure Data Factory and Azure Synapse.

Error code: DelimitedTextColumnNameNotAllowNull

- **Message**:

  ```
  The name of column index %index; is empty. Make sure column name is properly specified in the header row.
  ```

- **Cause**: When 'firstRowAsHeader' is set in the activity, the first row is used as the column name. This error means that the first row contains an empty value (for example, 'ColumnA, ColumnB').

- **Recommendation**: Check the first row, and fix the value if it is empty.

Error code: DelimitedTextMoreColumnsThanDefined

- **Message**:

  ```
  Error found when processing '%function;' source '%name;' with row number %rowCount;: found more columns than expected column count: %expectedColumnCount;.
  ```

- **Causes and recommendations**: Different causes may lead to this error. Check below list for possible cause analysis and related recommendation.

CAUSE ANALYSIS	RECOMMENDATION
The problematic row's column count is larger than the first row's column count. It might be caused by a data issue or incorrect column delimiter or quote char settings.	Get the row count from the error message, check the row's column, and fix the data.
If the expected column count is "1" in an error message, you might have specified wrong compression or format settings, which caused the files to be parsed incorrectly.	Check the format settings to make sure they match your source files.
If your source is a folder, the files under the specified folder might have a different schema.	Make sure that the files in the specified folder have an identical schema.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page

- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Dynamics 365, Dataverse (Common Data Service), and Dynamics CRM connectors in Azure Data Factory and Azure Synapse

8/19/2022 • 7 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Dynamics 365, Dataverse (Common Data Service), and Dynamics CRM connectors in Azure Data Factory and Azure Synapse.

Error code: DynamicsCreateServiceClientError

- **Message**: `This is a transient issue on Dynamics server side. Try to rerun the pipeline.`
- **Cause**: The problem is a transient issue on the Dynamics server side.
- **Recommendation**: Rerun the pipeline. If it fails again, try to reduce the parallelism. If the problem persists, contact Dynamics support.

Missing columns when you import a schema or preview data

- **Symptoms**: Some columns are missing when you import a schema or preview data. Error message: `The valid structure information (column name and type) are required for Dynamics source.`
- **Cause**: This issue is by design, because Data Factory and Synapse pipelines are unable to show columns that contain no values in the first 10 records. Make sure that the columns you've added are in the correct format.
- **Recommendation**: Manually add the columns in the mapping tab. For more details, refer to Explicit mapping.

Error code: DynamicsMissingTargetForMultiTargetLookupField

- **Message**: `Cannot find the target column for multi-target lookup field: '%fieldName;'.`
- **Cause**: The target column doesn't exist in the source or in the column mapping.
- **Recommendation**:

  1. Make sure that the source contains the target column.
  2. Add the target column in the column mapping. Ensure that the sink column is in the format *{fieldName}@EntityReference*.

Error code: DynamicsInvalidTargetForMultiTargetLookupField

- **Message**:

```
The provided target: '%targetName;' is not a valid target of field: '%fieldName;'. Valid targets are: '%validTargetNames;'
```

- **Cause**: A wrong entity name is provided as target entity of a multi-target lookup field.

- **Recommendation**: Provide a valid entity name for the multi-target lookup field.

Error code: DynamicsInvalidTypeForMultiTargetLookupField

- **Message**: `The provided target type is not a valid string. Field: '%fieldName;'.`
- **Cause**: The value in the target column is not a string.
- **Recommendation**: Provide a valid string in the multi-target lookup target column.

Error code: DynamicsFailedToRequetServer

- **Message**:
  ```
  The Dynamics server or the network is experiencing issues. Check network connectivity or check
  Dynamics server log for more details.
  ```
- **Cause**: The Dynamics server is instable or inaccessible, or the network is experiencing issues.
- **Recommendation**: For more details, check network connectivity or check the Dynamics server log. For further help, contact Dynamics support.

Error code: DynamicsFailedToConnect

- **Message**: `Failed to connect to Dynamics: %message;`
- **Causes and recommendations**: Different causes may lead to this error. Check below list for possible cause analysis and related recommendation.

CAUSE ANALYSIS	RECOMMENDATION

You are seeing
`ERROR REQUESTING ORGS FROM THE DISCOVERY SERVERFCB 'EnableRegionalDisco' is disabled.`
or otherwise
`Unable to Login to Dynamics CRM, message:ERROR REQUESTING Token FROM THE Authentication context - USER intervention required but not permitted by prompt behavior AADSTS50079: Due to a configuration change made by your administrator, or because you moved to a new location, you must enroll in multi-factor authentication to access '00000007-0000-0000-c000-000000000000'`
If your use case meets **all** of the following three conditions:
• You are connecting to Dynamics 365, Common Data Service, or Dynamics CRM.
• You are using Office365 Authentication.
• Your tenant and user is configured in Azure Active Directory for conditional access and/or Multi-Factor Authentication is required (see this link to Dataverse doc).
Under these circumstances, the connection used to succeed before 6/8/2021. Starting 6/9/2021 connection will start to fail because of the deprecation of regional Discovery Service (see this link).
	If your tenant and user is configured in Azure Active Directory for conditional access and/or Multi-Factor Authentication is required, you must use 'Azure AD service-principal' to authenticate after 6/8/2021. Refer this link for detailed steps.

If you see `Office 365 auth with OAuth failed` in the error message, it means that your server might have some configurations not compatible with OAuth.	• Contact Dynamics support team with the detailed error message for help. • Use the service principal authentication, and you can refer to this article: Example: Dynamics online using Azure AD service-principal and certificate authentication.

CAUSE ANALYSIS	RECOMMENDATION
If you see `Unable to retrieve authentication parameters from the serviceUri` in the error message, it means that either you input the wrong Dynamics service URL or proxy/firewall to intercept the traffic.	• Make sure you have put the correct service URI in the linked service. • If you use the Self Hosted IR, make sure that the firewall/proxy does not intercept the requests to the Dynamics server.
If you see `An unsecured or incorrectly secured fault was received from the other party` in the error message, it means that unexpected responses were gotten from the server side.	• Make sure your username and password are correct if you use the Office 365 authentication. • Make sure you have input the correct service URI. • If you use regional CRM URL (URL has a number after 'crm'), make sure you use the correct regional identifier. • Contact the Dynamics support team for help.
If you see `No Organizations Found` in the error message, it means that either your organization name is wrong or you used a wrong CRM region identifier in the service URL.	• Make sure you have input the correct service URI. • If you use the regional CRM URL (URL has a number after 'crm'), make sure that you use the correct regional identifier. • Contact the Dynamics support team for help.
If you see `401 Unauthorized` and AAD-related error message, it means that there's an issue with the service principal.	Follow the guidance in the error message to fix the service principal issue.
For other errors, usually the issue is on the server side.	Use XrmToolBox to make connection. If the error persists, contact the Dynamics support team for help.

Error code: DynamicsOperationFailed

- **Message**: `Dynamics operation failed with error code: %code;, error message: %message;.`
- **Cause**: The operation failed on the server side.
- **Recommendation**: Extract the error code of the dynamics operation from the error message: `Dynamics operation failed with error code: {code}`, and refer to the article Web service error codes for more detailed information. You can contact the Dynamics support team if necessary.

Error code: DynamicsInvalidFetchXml

- **Message**: `The Fetch Xml query specified is invalid.`
- **Cause**: There is an error existed in the fetch XML.
- **Recommendation**: Fix the error in the fetch XML.

Error code: DynamicsMissingKeyColumns

- **Message**:
  `Input DataSet must contain keycolumn(s) in Upsert/Update scenario. Missing key column(s): %column;`
- **Cause**: The source data does not contain the key column for the sink entity.
- **Recommendation**: Confirm that key columns are in the source data or map a source column to the key column on the sink entity.

Error code: DynamicsPrimaryKeyMustBeGuid

- **Message**: `The primary key attribute '%attribute;' must be of type guid.`
- **Cause**: The type of the primary key column is not 'Guid'.
- **Recommendation**: Make sure that the primary key column in the source data is of 'Guid' type.

Error code: DynamicsAlternateKeyNotFound

- **Message**: `Cannot retrieve key information of alternate key '%key;' for entity '%entity;'.`
- **Cause**: The provided alternate key does not exist, which may be caused by wrong key names or insufficient permissions.
- **Recommendation**:
  - Fix typos in the key name.
  - Make sure that you have sufficient permissions on the entity.

Error code: DynamicsInvalidSchemaDefinition

- **Message**: `The valid structure information (column name and type) are required for Dynamics source.`
- **Cause**: Sink columns in the column mapping miss the 'type' property.
- **Recommendation**: You can add the 'type' property to those columns in the column mapping by using JSON editor on the portal.

Error code: UserErrorUnsupportedAttributeType

- **Message**: `The attribute type 'Lookup' of field %attributeName; is not supported`
- **Cause**: When loading data to Dynamics sink, Azure Data Factory imposes validation on lookup attribute's metadata. However, there's the known issue of certain Dynamics entities not having valid lookup attribute metadata that holds a list of targets, which would fail the validation.
- **Recommendation**: Contact Dynamics support team to mitigate the issue.

The copy activity from the Dynamics 365 reads more rows than the actual number

- **Symptoms**: The copy activity from the Dynamics 365 reads more rows than the actual number.
- **Cause**: The Dynamics 365 server always indicates more available records.
- **Recommendation**: Use **XrmToolBox** to test the FetchXML with paging. **XrmToolBox** with some installed tools can get records count. For more information, see XrmToolBox.

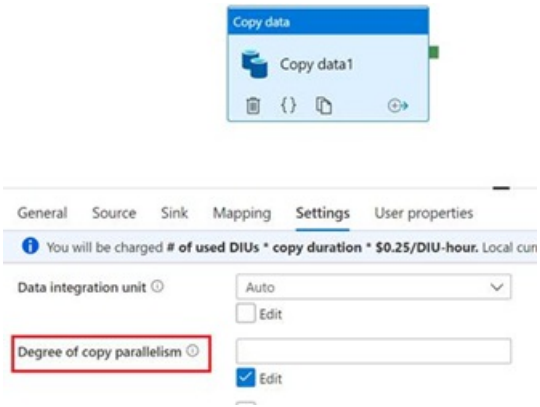
Cannot access virtual columns from Dynamics sources in the copy activity

- **Symptoms**: You cannot access virtual columns from Dynamics sources in the copy activity.
- **Cause**: The virtual column is not supported now.
- **Recommendation**: For the Option Set value, follow the options below to get it:

- You can get the object type code by referring to How to Find the Object Type Code for Any Entity and Dynamics 365 blog.
- You can link the StringMap entity to your target entity and get the associated values.

The parallel copy in a Dynamics CRM data store

- **Symptoms**: You don't know if it is possible to configure the parallel copy in a Dynamics CRM data store, and you also don't know the range of values that can be set in the "Degree of copy parallelism" section.
- **Recommendation**: The parallel copy controls parallelism, and the "Degree of copy parallelism" section can be set to non-zero value. Large number may cause throttling on dynamics server side, which may reduce the throughput, but now the throttling is handled by taking the public SDK.



Dynamics type conversion

- **Symptoms**: You try to convert the GUID to a string in the Dynamics source, but you encounter an error.
- **Cause**: When Dynamics is used as the source, the type conversion is not supported.
- **Recommendation**: Enable the staging and retry.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the FTP, SFTP and HTTP connectors in Azure Data Factory and Azure Synapse

8/19/2022 • 7 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the FTP, SFTP and HTTP connectors in Azure Data Factory and Azure Synapse.

FTP

Error code: FtpFailedToConnectToFtpServer

- **Message**:
  ```
  Failed to connect to FTP server. Please make sure the provided server information is correct, and try again.
  ```
- **Cause**: An incorrect linked service type might be used for the FTP server, such as using the Secure FTP (SFTP) linked service to connect to an FTP server.
- **Recommendation**: Check the port of the target server. FTP uses port 21.

Error code: FtpFailedToReadFtpData

- **Message**:
  ```
  Failed to read data from ftp: The remote server returned an error: 227 Entering Passive Mode (*,*,*,*,*,*).
  ```
- **Cause**: Port range between 1024 to 65535 is not open for data transfer under passive mode supported by the data factory or Synapse pipeline.
- **Recommendation**: Check the firewall settings of the target server. Open port 1024-65535 or port range specified in FTP server to SHIR/Azure IR IP address.

SFTP

Error code: SftpOperationFail

- **Message**: `Failed to '%operation;'. Check detailed error from SFTP.`
- **Cause**: A problem with the SFTP operation.
- **Recommendation**: Check the error details from SFTP.

Error code: SftpRenameOperationFail

- **Message**:
  ```
  Failed to rename the temp file. Your SFTP server doesn't support renaming temp file, set "useTempFileRename" as false in copy sink to disable uploading to temp file.
  ```
- **Cause**: Your SFTP server doesn't support renaming the temp file.
- **Recommendation**: Set "useTempFileRename" as false in the copy sink to disable uploading to the temp file.

Error code: SftpInvalidSftpCredential

- **Message**: `Invalid SFTP credential provided for '%type;' authentication type.`

- **Cause**: Private key content is fetched from the Azure key vault or SDK, but it's not encoded correctly.
- **Recommendation**:

  If the private key content is from your key vault, the original key file can work if you upload it directly to the SFTP linked service.

  For more information, see Copy data from and to the SFTP server by using data factory or Synapse pipelines. The private key content is base64 encoded SSH private key content.

  Encode *entire* original private key file with base64 encoding, and store the encoded string in your key vault. The original private key file is the one that can work on the SFTP linked service if you select **Upload** from the file.

  Here are some samples you can use to generate the string:

  - Use C# code:

    ```
    byte[] keyContentBytes = File.ReadAllBytes(Private Key Path);
    string keyContent = Convert.ToBase64String(keyContentBytes, Base64FormattingOptions.None);
    ```

  - Use Python code:

    ```
    import base64
    rfd = open(r'{Private Key Path}', 'rb')
    keyContent = rfd.read()
    rfd.close()
    print base64.b64encode(Key Content)
    ```

  - Use a third-party base64 conversion tool. We recommend the Encode to Base64 format tool.

- **Cause**: The wrong key content format was chosen.
- **Recommendation**:

  PKCS#8 format SSH private key (start with "-----BEGIN ENCRYPTED PRIVATE KEY-----") is currently not supported to access the SFTP server.

  To convert the key to traditional SSH key format, starting with "-----BEGIN RSA PRIVATE KEY-----", run the following commands:

  ```
  openssl pkcs8 -in pkcs8_format_key_file -out traditional_format_key_file
  chmod 600 traditional_format_key_file
  ssh-keygen -f traditional_format_key_file -p
  ```

- **Cause**: Invalid credentials or private key content.
- **Recommendation**: To see whether your key file or password is correct, double-check with tools such as WinSCP.

SFTP copy activity failed

- **Symptoms**:
  - Error code: UserErrorInvalidColumnMappingColumnNotFound
  - Error message: `Column 'AccMngr' specified in column mapping cannot be found in source data.`
- **Cause**: The source doesn't include a column named "AccMngr."
- **Resolution**: To determine whether the "AccMngr" column exists, double-check your dataset

configuration by mapping the destination dataset column.

Error code: SftpFailedToConnectToSftpServer

- **Message**: `Failed to connect to SFTP server '%server;'.`
- **Cause**: If the error message contains the string "Socket read operation has timed out after 30,000 milliseconds", one possible cause is that an incorrect linked service type is used for the SFTP server. For example, you might be using the FTP linked service to connect to the SFTP server.
- **Recommendation**: Check the port of the target server. By default, SFTP uses port 22.
- **Cause**: If the error message contains the string "Server response does not contain SSH protocol identification", one possible cause is that the SFTP server throttled the connection. Multiple connections are created to download from the SFTP server in parallel, and sometimes it will encounter SFTP server throttling. Ordinarily, different servers return different errors when they encounter throttling.
- **Recommendation**:

  Specify the maximum number of concurrent connections of the SFTP dataset as 1 and rerun the copy activity. If the activity succeeds, you can be sure that throttling is the cause.

  If you want to promote the low throughput, contact your SFTP administrator to increase the concurrent connection count limit, or you can do one of the following:

  - If you're using Self-hosted IR, add the Self-hosted IR machine's IP to the allowlist.
  - If you're using Azure IR, add Azure Integration Runtime IP addresses. If you don't want to add a range of IPs to the SFTP server allowlist, use Self-hosted IR instead.

Error code: SftpPermissionDenied

- **Message**: `Permission denied to access '%path;'`
- **Cause**: The specified user does not have read or write permission to the folder or file when operating.
- **Recommendation**: Grant the user with permission to read or write to the folder or files on SFTP server.

Error code: SftpAuthenticationFailure

- **Message**:

  ```
  Meet authentication failure when connect to Sftp server '%server;' using '%type;' authentication type.
  Please make sure you are using the correct authentication type and the credential is valid. For more
  details, see our troubleshooting docs.
  ```

- **Cause**: The specified credential (your password or private key) is invalid.
- **Recommendation**: Check your credential.
- **Cause**: The specified authentication type is not allowed or not sufficient to complete the authentication in your SFTP server.
- **Recommendation**: Apply the following options to use the correct authentication type:

  - If your server requires a password, use "Basic".
  - If your server requires a private key, use "SSH public key authentication".
  - If your server requires both "password" and "private key", use "Multiple factor authentication".
- **Cause**: Your SFTP server requires "keyboard-interactive" for authentication, but you provided "password".
- **Recommendation**:

  "keyboard-interactive" is a special authentication method, which is different from "password". It means that when logging into a server, you must enter the password manually, and you cannot use the

previously saved password. But Azure Data Factory (ADF) is a scheduled data transfer service, and there is no pop-up input box allowing you to provide the password at the runtime.

As a compromise, an option is provided to simulate the input in the background instead of your real manual input, which is equivalent to changing the "keyboard-interactive" to "password". If you can accept this security concern, follow the steps below to enable it:

1. On the ADF portal, hover on the SFTP linked service, and open its payload by selecting the code button.
2. Add `"allowKeyboardInteractiveAuth": true` in the "typeProperties" section.

Unable to connect to SFTP due to key exchange algorithms provided by SFTP are not supported in ADF

- **Symptoms**: You are unable to connect to SFTP via ADF and meet the following error message: `Failed to negotiate key exchange algorithm.`
- **Cause**: The key exchange algorithms provided by the SFTP server are not supported in ADF. The key exchange algorithms supported by ADF are:
  - curve25519-sha256
  - curve25519-sha256@libssh.org
  - ecdh-sha2-nistp256
  - ecdh-sha2-nistp384
  - ecdh-sha2-nistp521
  - diffie-hellman-group-exchange-sha256
  - diffie-hellman-group-exchange-sha1
  - diffie-hellman-group16-sha512
  - diffie-hellman-group14-sha256
  - diffie-hellman-group14-sha1
  - diffie-hellman-group1-sha1

  The latest version to support above key exchange algorithms on SHIR is version 5.19.

Error Code: SftpInvalidHostKeyFingerprint

- **Message**:
  ```
  Host key finger-print validation failed. Expected fingerprint is '<value in linked service>', real finger-print is '<server real value>'
  ```
- **Cause**: Azure Data Factory now supports more secure host key algorithms in SFTP connector. For the newly added algorithms, it requires to get the corresponding fingerprint in the SFTP server.

  The newly supported algorithms are:
  - ssh-ed25519
  - ecdsa-sha2-nistp256
  - ecdsa-sha2-nistp384
  - ecdsa-sha2-nistp521
- **Recommendation**: Get a valid fingerprint using the Host Key Name in `real finger-print` from the error message in the SFTP server. You can run the command to get the fingerprint on your SFTP server. For example: run `ssh-keygen -E md5 -lf <keyFilePath>` in Linux server to get the fingerprint. The command may vary among different server types.

HTTP

Error code: HttpFileFailedToRead

- **Message**: `Failed to read data from http server. Check the error from http server:%message;`

- **Cause**: This error occurs when a data factory or a Synapse pipeline talks to the HTTP server, but the HTTP request operation fails.
- **Recommendation**: Check the HTTP status code in the error message, and fix the remote server issue.

Error code: HttpSourceUnsupportedStatusCode

- **Message**: `Http source doesn't support HTTP Status Code '%code;'.`
- **Cause**: This error happens when Azure Data Factory requests HTTP source but gets unexpected status code.
- **Recommendation**: For more information about HTTP status code, see this document.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Hive connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Hive connector in Azure Data Factory and Azure Synapse.

The performance difference between the ODBC connector and the Hive connector

- **Symptoms**: The concern about the performance difference between the ODBC connector and the Hive connector.
- **Cause**: The ODBC connector needs its own driver, which may cause the performance difference due to the third-party driver quality.
- **Recommendation**: Use the Hive connector firstly.

Unexpected response received from the server when connecting to the Hive server via ODBC

- **Symptoms**: When connecting to the Hive server using ODBC linked service, you meet this error message:

  ```
  ERROR [HY000] [Cloudera][DriverSupport] (1110) Unexpected response received from server. Please ensure the server host and port specified for the connection are correct and confirm if SSL should be enabled for the connection.
  ```

- **Cause**: You use the Kerberos authentication that is not supported in Azure Data Factory.
- **Recommendation**: Try the following steps. If they do not work, check the provided driver to resolve this issue.

  1. The **kr5.ini** file is in the **C:\Program Files\MIT\Kerberos\bin** folder.
  2. Add the `KRB5_CONFIG` and `KRB5CCNAME` to the system variable as well.
  3. Edit the **krb.ini** file.
  4. Shut down and restart the VM and the SHIR from the machine.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Oracle connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Oracle connector in Azure Data Factory and Azure Synapse.

Oracle

Error code: ArgumentOutOfRangeException

- **Message**: `Hour, Minute, and Second parameters describe an un-representable DateTime.`
- **Cause**: In Azure Data Factory and Synapse pipelines, DateTime values are supported in the range from 0001-01-01 00:00:00 to 9999-12-31 23:59:59. However, Oracle supports a wider range of DateTime values, such as the BC century or min/sec>59, which leads to failure.
- **Recommendation**:

  To see whether the value in Oracle is in the supported range of dates, run `select dump(<column name>)`.

  To learn the byte sequence in the result, see How are dates stored in Oracle?.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the ORC format connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the ORC format connector in Azure Data Factory and Azure Synapse.

Error code: OrcJavaInvocationException

- **Message**: `An error occurred when invoking Java, message: %javaException;.`
- **Causes and recommendations**: Different causes may lead to this error. Check below list for possible cause analysis and related recommendation.

CAUSE ANALYSIS	RECOMMENDATION
When the error message contains the strings "java.lang.OutOfMemory", "Java heap space", and "doubleCapacity", it's usually a memory management issue in an old version of integration runtime.	If you're using Self-hosted Integration Runtime, we recommend that you upgrade to the latest version.
When the error message contains the string "java.lang.OutOfMemory", the integration runtime doesn't have enough resources to process the files.	Limit the concurrent runs on the integration runtime. For Self-hosted IR, scale up to a powerful machine with memory equal to or larger than 8 GB.
When the error message contains the string "NullPointerReference", the cause might be a transient error.	Retry the operation. If the problem persists, contact support.
When the error message contains the string "BufferOverflowException", the cause might be a transient error.	Retry the operation. If the problem persists, contact support.
When the error message contains the string "java.lang.ClassCastException:org.apache.hadoop.hive.serde2.io.HiveCharWritable can't be cast to org.apache.hadoop.io.Text", the cause might be a type conversion issue inside Java Runtime. Usually, it means that the source data can't be handled well in Java Runtime.	This is a data issue. Try to use a string instead of char or varchar in ORC format data.

Error code: OrcDateTimeExceedLimit

- **Message**:

```
The Ticks value '%ticks;' for the datetime column must be between valid datetime ticks range -
621355968000000000 and 2534022144000000000.
```

- **Cause**: If the datetime value is '0001-01-01 00:00:00', it could be caused by the differences between the Julian calendar and the Gregorian calendar.
- **Recommendation**: Check the ticks value and avoid using the datetime value '0001-01-01 00:00:00'.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Parquet format connector in Azure Data Factory and Azure Synapse

8/19/2022 • 5 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Parquet format connector in Azure Data Factory and Azure Synapse.

Error code: ParquetJavaInvocationException

- **Message**: `An error occurred when invoking java, message: %javaException;.`
- **Causes and recommendations**: Different causes may lead to this error. Check below list for possible cause analysis and related recommendation.

CAUSE ANALYSIS	RECOMMENDATION
When the error message contains the strings "java.lang.OutOfMemory", "Java heap space", and "doubleCapacity", it's usually a memory management issue in an old version of Integration Runtime.	If you are using Self-hosted IR and the version is earlier than 3.20.7159.1, we recommend that you upgrade to the latest version.
When the error message contains the string "java.lang.OutOfMemory", the integration runtime doesn't have enough resources to process the files.	Limit the concurrent runs on the integration runtime. For Self-hosted IR, scale up to a powerful machine with memory that's equal to or greater than 8 GB.
When the error message contains the string "NullPointerReference", it might be a transient error.	Retry the operation. If the problem persists, contact support.

Error code: ParquetInvalidFile

- **Message**: `File is not a valid Parquet file.`
- **Cause**: This is a Parquet file issue.
- **Recommendation**: Check to see whether the input is a valid Parquet file.

Error code: ParquetNotSupportedType

- **Message**: `Unsupported Parquet type. PrimitiveType: %primitiveType; OriginalType: %originalType;.`
- **Cause**: The Parquet format is not supported in Azure Data Factory and Synapse pipelines.
- **Recommendation**: Double-check the source data by going to Supported file formats and compression codecs by copy activity.

Error code: ParquetMissedDecimalPrecisionScale

- **Message**: `Decimal Precision or Scale information is not found in schema for column: %column;.`
- **Cause**: The number precision and scale were parsed, but no such information was provided.

- **Recommendation**: The source doesn't return the correct precision and scale information. Check the issue column for the information.

Error code: ParquetInvalidDecimalPrecisionScale

- **Message**: `Invalid Decimal Precision or Scale. Precision: %precision; Scale: %scale;.`
- **Cause**: The schema is invalid.
- **Recommendation**: Check the issue column for precision and scale.

Error code: ParquetColumnNotFound

- **Message**: `Column %column; does not exist in Parquet file.`
- **Cause**: The source schema is a mismatch with the sink schema.
- **Recommendation**: Check the mappings in the activity. Make sure that the source column can be mapped to the correct sink column.

Error code: ParquetInvalidDataFormat

- **Message**: `Incorrect format of %srcValue; for converting to %dstType;.`
- **Cause**: The data can't be converted into the type that's specified in mappings.source.
- **Recommendation**: Double-check the source data or specify the correct data type for this column in the copy activity column mapping. For more information, see Supported file formats and compression codecs by the copy activity.

Error code: ParquetDataCountNotMatchColumnCount

- **Message**:
  ```
  The data count in a row '%sourceColumnCount;' does not match the column count '%sinkColumnCount;' in given schema.
  ```
- **Cause**: A mismatch between the source column count and the sink column count.
- **Recommendation**: Double-check to ensure that the source column count is same as the sink column count in 'mapping'.

Error code: ParquetDataTypeNotMatchColumnType

- **Message**: `The data type %srcType; is not match given column type %dstType; at column '%columnIndex;'.`
- **Cause**: The data from the source can't be converted to the type that's defined in the sink.
- **Recommendation**: Specify a correct type in mapping.sink.

Error code: ParquetBridgeInvalidData

- **Message**: `%message;`
- **Cause**: The data value has exceeded the limit.
- **Recommendation**: Retry the operation. If the issue persists, contact us.

Error code: ParquetUnsupportedInterpretation

- **Message**: `The given interpretation '%interpretation;' of Parquet format is not supported.`
- **Cause**: This scenario isn't supported.
- **Recommendation**: 'ParquetInterpretFor' should not be 'sparkSql'.

Error code: ParquetUnsupportFileLevelCompressionOption

- **Message**: `File level compression is not supported for Parquet.`
- **Cause**: This scenario isn't supported.
- **Recommendation**: Remove 'CompressionType' in the payload.

Error code: UserErrorJniException

- **Message**: `Cannot create JVM: JNI return code [-6][JNI call failed: Invalid arguments.]`
- **Cause**: A Java Virtual Machine (JVM) can't be created because some illegal (global) arguments are set.
- **Recommendation**: Log in to the machine that hosts *each node* of your self-hosted IR. Check to ensure that the system variable is set correctly, as follows: `_JAVA_OPTIONS "-Xms256m -Xmx16g" with memory bigger than 8 G`. Restart all the IR nodes, and then rerun the pipeline.

Arithmetic overflow

- **Symptoms**: Error message occurred when you copy Parquet files: `Message = Arithmetic Overflow., Source = Microsoft.DataTransfer.Common`
- **Cause**: Currently only the decimal of precision <= 38 and length of integer part <= 20 are supported when you copy files from Oracle to Parquet.
- **Resolution**: As a workaround, you can convert any columns with this problem into VARCHAR2.

No enum constant

- **Symptoms**: Error message occurred when you copy data to Parquet format: `java.lang.IllegalArgumentException:field ended by ';'`, or: `java.lang.IllegalArgumentException:No enum constant org.apache.parquet.schema.OriginalType.test`.
- **Cause**:

  The issue could be caused by white spaces or unsupported special characters (such as,;{}()\n\t=) in the column name, because Parquet doesn't support such a format.

  For example, a column name such as *contoso(test)* will parse the type in brackets from code `Tokenizer st = new Tokenizer(schemaString, " ;{}()\n\t");`. The error is thrown because there is no such "test" type.

  To check supported types, go to the GitHub apache/parquet-mr site.

- **Resolution**:

  Double-check to see whether:

  - There are white spaces in the sink column name.
  - The first row with white spaces is used as the column name.
  - The type OriginalType is supported. Try to avoid using these special characters: `,;{}()\n\t=`.

Error code: ParquetDateTimeExceedLimit

- **Message**:
  ```
  The Ticks value '%ticks;' for the datetime column must be between valid datetime ticks range -
  621355968000000000 and 2534022144000000000.
  ```
- **Cause**: If the datetime value is '0001-01-01 00:00:00', it could be caused by the difference between Julian Calendar and Gregorian Calendar. For more details, reference Difference between Julian and proleptic Gregorian calendar dates.
- **Resolution**: Check the ticks value and avoid using the datetime value '0001-01-01 00:00:00'.

Error code: ParquetInvalidColumnName

- **Message**: `The column name is invalid. Column name cannot contain these character:[,;{}()\n\t=]`
- **Cause**: The column name contains invalid characters.
- **Resolution**: Add or modify the column mapping to make the sink column name valid.

The file created by the copy data activity extracts a table that contains a varbinary (max) column

- **Symptoms**: The Parquet file created by the copy data activity extracts a table that contains a varbinary (max) column.
- **Cause**: This issue is caused by the Parquet-mr library bug of reading large column.
- **Resolution**: Try to generate smaller files (size < 1G) with a limitation of 1000 rows per file.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the REST connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the REST connector in Azure Data Factory and Azure Synapse.

Error code: RestSinkCallFailed

- **Message**: `Rest Endpoint responded with Failure from server. Check the error from server:%message;`
- **Cause**: This error occurs when a data factory or Synapse pipeline talks to the REST endpoint over HTTP protocol, and the request operation fails.
- **Recommendation**: Check the HTTP status code or the message in the error message and fix the remote server issue.

Error code: RestSourceCallFailed

- **Message**:

  ```
  The HttpStatusCode %statusCode; indicates failure.&#xA;Request URL: %requestUri;&#xA;Response payload:%payload;
  ```

- **Cause**: This error occurs when Azure Data Factory talks to the REST endpoint over HTTP protocol, and the request operation fails.
- **Recommendation**: Check the HTTP status code or the request URL or the response payload in the error message and fix the remote server issue.

Error code: RestSinkUNSupportedCompressionType

- **Message**: `User Configured CompressionType is Not Supported By Azure Data Factory:%message;`
- **Recommendation**: Check the supported compression types for the REST sink.

Unexpected network response from the REST connector

- **Symptoms**: The endpoint sometimes receives an unexpected response (400, 401, 403, 500) from the REST connector.
- **Cause**: The REST source connector uses the URL and HTTP method/header/body from the linked service/dataset/copy source as parameters when it constructs an HTTP request. The issue is most likely caused by some mistakes in one or more specified parameters.
- **Resolution**:
  - Use 'curl' in a Command Prompt window to see whether the parameter is the cause (**Accept** and **User-Agent** headers should always be included):

    ```
    curl -i -X <HTTP method> -H <HTTP header1> -H <HTTP header2> -H "Accept: application/json" -H "User-Agent: azure-data-factory/2.0" -d '<HTTP body>' <URL>
    ```

If the command returns the same unexpected response, fix the preceding parameters with 'curl' until it returns the expected response.

You can also use 'curl--help' for more advanced usage of the command.

- If only the REST connector returns an unexpected response, contact Microsoft support for further troubleshooting.
- Note that 'curl' might not be suitable to reproduce an SSL certificate validation issue. In some scenarios, the 'curl' command was executed successfully without encountering any SSL certificate validation issues. But when the same URL is executed in a browser, no SSL certificate is actually returned for the client to establish trust with server.

  Tools like **Postman** and **Fiddler** are recommended for the preceding case.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the SharePoint Online list connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the SharePoint Online list connector in Azure Data Factory and Azure Synapse.

Error code: SharePointOnlineAuthFailed

- **Message**: `The access token generated failed, status code: %code;, error message: %message;.`
- **Cause**: The service principal ID and key might not be set correctly.
- **Recommendation**: Check your registered application (service principal ID) and key to see whether they're set correctly.

Connection failed after granting permission in SharePoint Online List

Symptoms

You granted permission to your data factory in SharePoint Online List, but you still fail with the following error message:

```
Failed to get metadata of odata service, please check if service url and credential is correct and your application has permission to the resource. Expected status code: 200, actual status code: Unauthorized, response is : {"error":"invalid_request","error_description":"Token type is not allowed."}.
```

Cause

The SharePoint Online List uses ACS to acquire the access token to grant access to other applications. But for the tenant built after November 7, 2018, ACS is disabled by default.

Recommendation

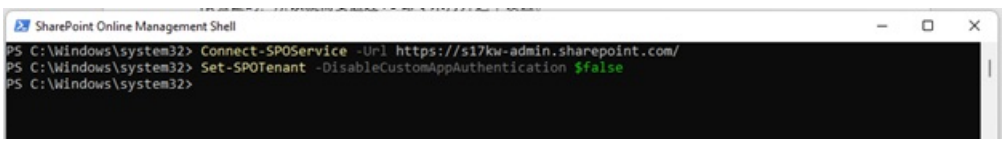
You need to enable ACS to acquire the access token. Take the following steps:

1. Download SharePoint Online Management Shell, and ensure that you have a tenant admin account.
2. Run the following command in the SharePoint Online Management Shell. Replace `<tenant name>` with your tenant name and add `-admin` after it.

   ```
   Connect-SPOService -Url https://<tenant name>-admin.sharepoint.com/
   ```

3. Enter your tenant admin information in the pop-up authentication window.
4. Run the following command:

   ```
   Set-SPOTenant -DisableCustomAppAuthentication $false
   ```



5. Use ACS to get the access token.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the Snowflake connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the Snowflake connector in Azure Data Factory and Azure Synapse.

Error message: IP % is not allowed to access Snowflake. Contact your local security administrator.

- **Symptoms**: The copy activity fails with the following error:

```
Job failed due to reason: net.snowflake.client.jdbc.SnowflakeSQLException: IP % is not allowed to
access Snowflake. Contact your local security administrator.
```

- **Cause**: It's a connectivity issue and usually caused by firewall IP issues when integration runtimes access your Snowflake.

- **Recommendation**:

  - If you configure a self-hosted integration runtime to connect to Snowflake, make sure to add your self-hosted integration runtime IPs to the allowed list in Snowflake.
  - If you use an Azure Integration Runtime and the access is restricted to IPs approved in the firewall rules, you can add Azure Integration Runtime IPs to the allowed list in Snowflake.
  - If you use a managed private endpoint and a network policy is in place on your Snowflake account, ensure Managed VNet CIDR is allowed. For more steps, refer to How To: Set up a managed private endpoint from Azure Data Factory or Synapse to Snowflake.

Error message: Failed to access remote file: access denied.

- **Symptoms**: The copy activity fails with the following error:

```
ERROR [42501] Failed to access remote file: access denied. Please check your credentials,Source
=SnowflakeODBC_sb64.dll..
```

- **Cause**: The error pops up by the Snowflake COPY command and is caused by missing access permission on source/sink when execute Snowflake COPY commands.

- **Recommendation**: Check your source/sink to make sure that you have granted proper access permission to Snowflake.

  - Direct copy: Make sure to grant access permission to Snowflake in the other source/sink.
  - Staged copy: The staging Azure Blob storage linked service must use shared access signature authentication. When you generate the shared access signature, make sure to set the allowed permissions and IP addresses to Snowflake in the staging Azure Blob storage. To learn more about this, see this article.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot the XML format connector in Azure Data Factory and Azure Synapse

8/19/2022 • 2 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article provides suggestions to troubleshoot common problems with the XML format connector in Azure Data Factory and Azure Synapse.

Error code: XmlSinkNotSupported

- **Message**: `Write data in XML format is not supported yet, choose a different format!`
- **Cause**: An XML dataset was used as a sink dataset in your copy activity.
- **Recommendation**: Use a dataset in a different format from that of the sink dataset.

Error code: XmlAttributeColumnNameConflict

- **Message**:

  ```
  Column names %attrNames;' for attributes of element '%element;' conflict with that for corresponding child elements, and the attribute prefix used is '%prefix;'.
  ```

- **Cause**: An attribute prefix was used, which caused the conflict.
- **Recommendation**: Set a different value for the "attributePrefix" property.

Error code: XmlValueColumnNameConflict

- **Message**:

  ```
  Column name for the value of element '%element;' is '%columnName;' and it conflicts with the child element having the same name.
  ```

- **Cause**: One of the child element names was used as the column name for the element value.
- **Recommendation**: Set a different value for the "valueColumn" property.

Error code: XmlInvalid

- **Message**: `Input XML file '%file;' is invalid with parsing error '%error;'.`
- **Cause**: The input XML file is not well formed.
- **Recommendation**: Correct the XML file to make it well formed.

Next steps

For more troubleshooting help, try these resources:

- Connector troubleshooting guide
- Data Factory blog
- Data Factory feature requests
- Azure videos

- Microsoft Q&A page
- Stack Overflow forum for Data Factory
- Twitter information about Data Factory

Troubleshoot CI-CD, Azure DevOps, and GitHub issues in Azure Data Factory and Synapse Analytics

8/19/2022 • 15 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

In this article, let us explore common troubleshooting methods for Continuous Integration-Continuous Deployment (CI-CD), Azure DevOps and GitHub issues in Azure Data Factory and Synapse Analytics.

If you have questions or issues in using source control or DevOps techniques, here are a few articles you may find useful:

- Refer to Source Control to learn how source control is practiced in the service.
- Refer to CI-CD to learn more about how DevOps CI-CD is practiced in the service.

Common errors and messages

Connect to Git repository failed due to different tenant

Issue

Sometimes you encounter Authentication issues like HTTP status 401. Especially when you have multiple tenants with guest account, things could become more complicated.

Cause

The token was obtained from the original tenant, but the service is in guest tenant trying to use the token to visit DevOps in guest tenant. This type of token access isn't the expected behavior.

Recommendation

You should use the token issued from guest tenant. For example, you have to assign the same Azure Active Directory to be your guest tenant and your DevOps, so it can correctly set token behavior and use the correct tenant.

Template parameters in the parameters file aren't valid

Issue

If we delete a trigger in Dev branch, which is already available in Test or Production branch with **same** configuration (like frequency and interval), then release pipeline deployment succeeds and corresponding trigger will be deleted in respective environments. But if you have **different** configuration (like frequency and interval) for trigger in Test/Production environments and if you delete the same trigger in Dev, then deployment fails with an error.

Cause

CI/CD Pipeline fails with the following error:

```
2020-07-20T11:19:02.1276769Z ##[error]Deployment template validation failed: 'The template parameters
'Trigger_Salesforce_properties_typeProperties_recurrence_frequency,
Trigger_Salesforce_properties_typeProperties_recurrence_interval,
Trigger_Salesforce_properties_typeProperties_recurrence_startTime,
Trigger_Salesforce_properties_typeProperties_recurrence_timeZone' in the parameters file are not valid; they
are not present in the original template and can therefore not be provided at deployment time. The only
supported parameters for this template are 'factoryName, PlanonDWH_connectionString,
PlanonKeyVault_properties_typeProperties_baseUrl
```

Recommendation

The error occurs because we often delete a trigger, which is parameterized, therefore, the parameters will not be available in the Azure Resource Manager (ARM) template (because the trigger does not exist anymore). Since

the parameter is not in the ARM template anymore, we have to update the overridden parameters in the DevOps pipeline. Otherwise, each time the parameters in the ARM template change, they must update the overridden parameters in the DevOps pipeline (in the deployment task).

Updating property type is not supported

Issue

CI/CD release pipeline failing with the following error:

```
2020-07-06T09:50:50.8716614Z There were errors in your deployment. Error code: DeploymentFailed.
2020-07-06T09:50:50.8760242Z ##[error]At least one resource deployment operation failed. Please list
deployment operations for details. Please see https://aka.ms/DeployOperations for usage details.
2020-07-06T09:50:50.8771655Z ##[error]Details:
2020-07-06T09:50:50.8772837Z ##[error]DataFactoryPropertyUpdateNotSupported: Updating property type is not
supported.
2020-07-06T09:50:50.8774148Z ##[error]DataFactoryPropertyUpdateNotSupported: Updating property type is not
supported.
2020-07-06T09:50:50.8775530Z ##[error]Check out the troubleshooting guide to see if your issue is addressed:
https://docs.microsoft.com/azure/devops/pipelines/tasks/deploy/azure-resource-group-
deployment#troubleshooting
2020-07-06T09:50:50.8776801Z ##[error]Task failed while creating or updating the template deployment.
```

Cause

This error is due to an integration runtime with the same name in the target service instance, but with a different type. Integration Runtime needs to be of the same type during deployment.

Recommendation

- Refer to the Best Practices for CI/CD
- Integration runtimes don't change often and are similar across all stages in your CI/CD, so the service expects you to have the same name and type of integration runtime across all stages of CI/CD. If the name and types & properties are different, make sure to match the source and target integration runtime configuration and then deploy the release pipeline.
- If you want to share integration runtimes across all stages, consider using a ternary factory just to contain the shared integration runtimes. You can use this shared factory in all of your environments as a linked integration runtime type.

Document creation or update failed because of invalid reference

Issue

When trying to publish changes, you get following error message:

```
"error": { "code": "BadRequest", "message": "The document creation or update failed because of invalid reference
'<entity>'.", "target": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/<rgname>/providers/Microsoft.DataFactory/factories/<datafactory>/pipelines/<pipeline>",
"details": null }
```

Cause

You have detached the Git configuration and set it up again with the "Import resources" flag selected, which sets the service as "in sync". This means no change during publication.

Resolution

Detach Git configuration and set it up again, and make sure NOT to check the "import existing resources" checkbox.

Data factory move failing from one resource group to another

Issue

You are unable to move a data factory from one Resource Group to another, failing with the following error:

```
{ "code": "ResourceMoveProviderValidationFailed", "message": "Resource move validation failed. Please see
details. Diagnostic information: timestamp 'xxxxxxxxxxxxZ', subscription id 'xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx', tracking id 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx', request correlation id 'xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx'.", "details": [ { "code": "BadRequest", "target": "Microsoft.DataFactory/factories",
"message": "One of the resources contain integration runtimes that are either SSIS-IRs in
starting/started/stopping state, or Self-Hosted IRs which are shared with other resources. Resource move is
not supported for those resources." } ] }
```

Resolution

You can delete the SSIS-IR and Shared IRs to allow the move operation. If you don't want to delete the integration runtimes, then the best way is to follow the copy and clone document to do the copy and after it's done, delete the old data factory.

Unable to export and import ARM template

Issue

Unable to export and import ARM template. No error was on the portal, however, in the browser trace, you could see the following error:

```
Failed to load resource: the server responded with a status code of 401 (Unauthorized)
```

Cause

You have created a customer role as the user and it did not have the necessary permission. When the UI is loaded, a series of exposure control values is checked. In this case, the user's access role does not have permission to access *queryFeaturesValue* API. To access this API, the global parameters feature is turned off. The ARM export code path is partly relying on the global parameters feature.

Resolution

In order to resolve the issue, you need to add the following permission to your role: *Microsoft.DataFactory/factories/queryFeaturesValue/action*. This permission is included by default in the **Data Factory Contributor** role for Data Factory, and the **Contributor** role In Synapse Analytics.

Cannot automate publishing for CI/CD

Cause

Until recently, the it was only possible to publish a pipeline for deployments by clicking the UI in the Portal. Now, this process can be automated.

Resolution

CI/CD process has been enhanced. The **Automated** publish feature takes, validates, and exports all ARM template features from the UI. It makes the logic consumable via a publicly available npm package @microsoft/azure-data-factory-utilities. This method allows you to programmatically trigger these actions instead of having to go to the UI and click a button. This method gives your CI/CD pipelines a **true** continuous integration experience. Follow CI/CD Publishing Improvements for details.

Cannot publish because of 4-MB ARM template limit

Issue

You can't deploy because you hit Azure Resource Manager limit of 4-MB total template size. You need a solution to deploy after crossing the limit.

Cause

Azure Resource Manager restricts template size to be 4 MB. Limit the size of your template to 4 MB, and each parameter file to 64 KB. The 4-MB limit applies to the final state of the template after it has been expanded with iterative resource definitions, and values for variables and parameters. But, you have crossed the limit.

Resolution

For small to medium solutions, a single template is easier to understand and maintain. You can see all the resources and values in a single file. For advanced scenarios, linked templates enable you to break down the solution into targeted components. Follow best practice at Using Linked and Nested Templates.

DevOps API limit of 20 MB causes ADF trigger twice or more instead of once

Issue

While publishing ADF resources, the azure pipeline triggers twice or more instead of once.

Cause

Azure DevOps has the 20 MB REST API limit. When the ARM template exceeds this size, ADF internally splits the template file into multiple files with linked templates to solve this issue. As a side effect, this split could result in customer's triggers being run more than once.

Resolution

Use ADF **Automated publish** (preferred) or **manual trigger** method to trigger once instead of twice or more.

Cannot connect to GIT Enterprise

Issue

You can't connect to GIT Enterprise because of permission issues. You can see error like **422 - Unprocessable Entity.**

Cause

- You haven't configured Oauth for the service.
- Your URL is misconfigured. The repoConfiguration should be of type FactoryGitHubConfiguration

Resolution

You grant Oauth access to the service at first. Then, you have to use correct URL to connect to GIT Enterprise. The configuration must be set to the customer organization(s). For example, the service will try *https://hostname/api/v3/search/repositories?q=user%3<customer credential>...*. at first and fail. Then, it will try *https://hostname/api/v3/orgs/<org>/<repo>...*, and succeed.

Cannot recover from a deleted instance

Issue

An instance of the service, or the resource group containing it, was deleted and needs to be recovered.

Cause

It is possible to recover the instance only if source control was configured for it with DevOps or Git. This action will bring all the latest published resources, but **will not** restore any unpublished pipelines, datasets, or linked services. If there is no Source control, recovering a deleted instance from the Azure backend isn't possible because once the service receives the delete command, the instance is permanently deleted without any backup.

Resolution

To recover a deleted service instance that has source control configured, refer the steps below:

- Create a new instance of the service.
- Reconfigure Git with the same settings, but make sure to import existing resources to the selected repository, and choose New branch.
- Create a pull request to merge the changes to the collaboration branch and publish.
- If there was a Self-hosted Integration Runtime in a deleted data factory or Synapse workspace, a new instance of the IR must be created in a new factory or workspace. The on-premises or virtual machine IR instance must be uninstalled and reinstalled, and a new key obtained. After setup of the new IR is completed, the Linked Service must be updated to point to new IR and the connected tested again, or it will fail with error **invalid reference.**

Can't deploy to different stage using automatic publish method

Issue

Customer followed all necessary steps like installing NPM package and setting up a higher stage using Azure DevOps, but deployment still fails.

Cause

While npm packages can be consumed in various ways, one of the primary benefits is being consumed via

Azure Pipeline. On each merge into your collaboration branch, a pipeline can be triggered that first validates all of the code and then exports the ARM template into a build artifact that can be consumed by a release pipeline. In Starter pipeline, YAML file should be valid and complete.

Resolution

Following section is not valid because package.json folder is not valid.

```
- task: Npm@1
  inputs:
    command: 'custom'
    workingDir: '$(Build.Repository.LocalPath)/<folder-of-the-package.json-file>' #replace with the
package.json folder
    customCommand: 'run build validate $(Build.Repository.LocalPath) /subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/testResourceGroup/providers/Microsoft.DataFactory/factories/yourFactoryName'
  displayName: 'Validate'
```

It should have DataFactory included in customCommand like *'run build validate $(Build.Repository.LocalPath)/DataFactory/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/resourceGroups/testResourceGroup/providers/Microsoft.DataFactory/factories/yourFactoryName'*. Make sure the generated YAML file for higher stage should have required JSON artifacts.

Extra left "[" displayed in published JSON file

Issue

When publishing with DevOps, there is an extra "[" displayed. The service adds one more "[" in an ARMTemplate in DevOps automatically. You will see an expression like "[[" in JSON file.

Cause

Because [is a reserved character for ARM, an extra [is added automatically to escape "[".

Resolution

This is normal behavior during the publishing process for CI/CD.

Perform CI/CD during progress/queued stage of pipeline run

Issue

You want to perform CI/CD during progress and queuing stage of pipeline run.

Cause

When pipeline is in progress/queued stage, you have to monitor the pipeline and activities at first. Then, you can decide to wait until pipeline to finish or you can cancel the pipeline run.

Resolution

You can monitor the pipeline using **SDK**, **Azure Monitor** or Monitor. Then, you can follow CI/CD Best Practices to guide you further.

Perform UNIT TESTING during development and deployment

Issue

You want to perform unit testing during development and deployment of your pipelines.

Cause

During development and deployment cycles, you may want to unit test your pipeline before you manually or automatically publish your pipeline. Test automation allows you to run more tests, in less time, with guaranteed repeatability. Automatically retesting all your pipelines before deployment gives you some protection against regression faults. Automated testing is a key component of CI/CD software development approaches: inclusion of automated tests in CI/CD deployment pipelines can significantly improve quality. In long run, tested pipeline artifacts are reused saving you cost and time.

Resolution

Because customers may have different unit testing requirements with different skillsets, usual practice is to

follow following steps:

1. Setup Azure DevOps CI/CD project or develop .NET/PYTHON/REST type SDK driven test strategy.
2. For CI/CD, create a build artifact containing all scripts and deploy resources in release pipeline. For an SDK driven approach, develop Test units using PyTest in Python, Nunit in C# using .NET SDK and so on.
3. Run unit tests as part of release pipeline or independently with ADF Python/PowerShell/.NET/REST SDK.

For example, you want to delete duplicates in a file and then store curated file as table in a database. To test the pipeline, you set up a CI/CD project using Azure DevOps. You set up a TEST pipeline stage where you deploy your developed pipeline. You configure TEST stage to run Python tests for making sure table data is what you expected. If you do not use CI/CD, you use **Nunit** to trigger deployed pipelines with tests you want. Once you are satisfied with the results, you can finally publish the pipeline to a production instance.

Pipeline runs temporarily fail after CI/CD deployment or authoring updates

Issue

After some amount of time, new pipeline runs begin to succeed without any user action after temporary failures.

Cause

There are several scenarios, which can trigger this behavior, all of which involve a new version of a dependent resource being called by the old version of the parent resource. For example, suppose an existing child pipeline called by "Execute pipeline" is updated to have required parameters and the existing parent pipeline is updated to pass these parameters. If the deployment occurs during a parent pipeline execution, but before the **Execute Pipeline** activity, the old version of the pipeline will call the new version of the child pipeline, and the expected parameters will not be passed. This will cause the pipeline to fail with a *UserError*. This can also occur with other types of dependencies, such as if a breaking change is made to linked service during a pipeline run that references it.

Resolution

New runs of the parent pipeline will automatically begin succeeding, so typically no action is needed. However, to prevent these errors, customers should consider dependencies while authoring and planning deployments to avoid breaking changes.

Cannot parameterize integration run time in linked service

Issue

Need to parameterize linked service integration run time

Cause

This feature isn't supported.

Resolution

You have to select manually and set an integration runtime. You can use PowerShell API to change as well. This change can have downstream implications.

Update/change Integration runtime during CI/CD.

Issue

Changing Integration runtime name during CI/CD deployment.

Cause

Parameterizing an entity reference (Integration runtime in Linked service, Dataset in activity, Linked Service in dataset) isn't supported. Changing the runtime name during deployment will cause the depended resource (Resource referencing the Integration runtime) to become malformed with invalid reference.

Resolution

Data Factory requires you to have the same name and type of integration runtime across all stages of CI/CD.

ARM template deployment failing with error DataFactoryPropertyUpdateNotSupported

Issue

ARM template deployment fails with an error such as DataFactoryPropertyUpdateNotSupported: Updating

property type isn't supported.

Cause

The ARM template deployment is attempting to change the type of an existing integration runtime. This isn't allowed and will cause a deployment failure because data factory requires the same name and type of integration runtime across all stages of CI/CD.

Resolution

If you want to share integration runtimes across all stages, consider using a ternary factory just to contain the shared integration runtimes. You can use this shared factory in all of your environments as a linked integration runtime type. For more information, refer to Continuous integration and delivery - Azure Data Factory

GIT publish may fail because of PartialTempTemplates files

Issue

When you've 1000 s of old temporary ARM json files in PartialTemplates folder, publish may fail.

Cause

On publish, ADF fetches every file inside each folder in the collaboration branch. In the past, publishing generated two folders in the publish branch: PartialArmTemplates and LinkedTemplates. PartialArmTemplates files are no longer generated. However, because there can be many old files (thousands) in the PartialArmTemplates folder, this may result in many requests being made to GitHub on publish and the rate limit being hit.

Resolution

Delete the PartialTemplates folder and republish. You can delete the temporary files in that folder as well.

Include global parameters in ARM template option does not work

Issue

If you are using old default parameterization template, new way to include global paramaters from **Manage Hub** will not work.

Cause

Default parameterization template should include all values from global parameter list.

Resolution

Use updated default parameterization template. as one time migration to new method of including global parameters. This template references to all values in global parameter list.

Error code: InvalidTemplate

Issue

Message says *Unable to parse expression.* The expression passed in the dynamic content of an activity isn't being processed correctly because of a syntax error.

Cause

Dynamic content is not written as per expression language requirements.

Resolution

- For debug run, check expressions in pipeline within current git branch.
- For Triggered run, check expressions in pipeline within *Live* mode .

Next steps

For more help with troubleshooting, try the following resources:

- Data Factory blog
- Data Factory feature requests
- Azure videos
- Stack overflow forum for Data Factory

- Twitter information about Data Factory

Troubleshoot Azure Data Factory and Synapse Analytics security and access control issues

8/19/2022 • 9 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article explores common troubleshooting methods for security and access control in Azure Data Factory and Synapse Analytics pipelines.

Common errors and messages

Connectivity issue in the copy activity of the cloud datastore

Symptoms

Various error messages might be returned when connectivity issues occur in the source or sink datastore.

Cause

The problem is usually caused by one of the following factors:

- The proxy setting in the self-hosted integration runtime (IR) node, if you're using a self-hosted IR.
- The firewall setting in the self-hosted IR node, if you're using a self-hosted IR.
- The firewall setting in the cloud datastore.

Resolution

- To ensure that this is a connectivity issue, check the following points:
  - The error is thrown from the source or sink connectors.
  - The failure is at the start of the copy activity.
  - The failure is consistent for Azure IR or the self-hosted IR with one node, because it could be a random failure in a multiple-node self-hosted IR if only some of the nodes have the issue.
- If you're using a **self-hosted IR**, check your proxy, firewall, and network settings, because connecting to the same datastore could succeed if you're using an Azure IR. To troubleshoot this scenario, see:
  - Self-hosted IR ports and firewalls
  - Azure Data Lake Storage connector
- If you're using an **Azure IR**, try to disable the firewall setting of the datastore. This approach can resolve the issues in the following two situations:
  - Azure IR IP addresses are not in the allow list.
  - The *Allow trusted Microsoft services to access this storage account* feature is turned off for Azure Blob Storage and Azure Data Lake Storage Gen 2.
  - The *Allow access to Azure services* setting isn't enabled for Azure Data Lake Storage Gen1.

If none of the preceding methods works, contact Microsoft for help.

Invalid or empty authentication key issue after public network access is disabled

Symptoms

After you disable public network access for the service, the self-hosted integration runtime throws following errors: `The Authentication key is invalid or empty.` or

```
Cannot connect to the data factory. Please check whether the factory has enabled public network access or the machine is hosted in a approved private endpoint Virtual Network.
```

Cause

The problem is most likely caused by a Domain Name System (DNS) resolution issue, because disabling public connectivity and establishing a private endpoint prevents reconnection.

To verify whether the service's fully qualified domain name (FQDN) is resolved to the public IP address, do the following:

1. Confirm that you've created the Azure virtual machine (VM) in the same virtual network as the service's private endpoint.
2. Run PsPing and Ping from the Azure VM to the service FQDN:

```
psping.exe <dataFactoryName>.<region>.datafactory.azure.net:443
ping <dataFactoryName>.<region>.datafactory.azure.net
```

> **NOTE**
> You must specify a port for the PsPing command. Port 443 is shown here but is not required.

3. Check to see whether both commands resolve to an Azure Data Factory public IP that's based on a specified region. The IP should be in the following format: `xxx.xxx.xxx.0`

Resolution

To resolve the issue, do the following:

- As option, we would like to suggest you to manually add a "Virtual Network link" under the "Private link DNS Zone" for the service. For details, refer to the Azure Private Link article. The instruction is for configuring the private DNS zone or custom DNS server to resolve the service FQDN to a private IP address.
- However, if you don't want to configure the private DNS zone or custom DNS server, try the following temporary solution:

  1. Change the host file in Windows, and map the private IP (the service's private endpoint) to the service FQDN.

     In the Azure VM, go to `C:\Windows\System32\drivers\etc`, and then open the *host* file in Notepad. Add the line that maps the private IP to the FQDN at the end of the file, and save the change.

     ```
     # localhost name resolution is handled within DNS itself.
     #       127.0.0.1       localhost
     #       ::1             localhost

           10.0.0.1        <dataFactoryName>.<region>.datafactory.azure.net
     ```

  2. Rerun the same commands as in the preceding verification steps to check the response, which should contain the private IP.

  3. Re-register the self-hosted integration runtime, and the issue should be resolved.

Unable to register IR authentication key on Self-hosted VMs due to private link

Symptoms

You're unable to register the IR authentication key on the self-hosted VM because the private link is enabled. You receive the following error message:

"Failed to get service token from ADF service with key ********** and time cost is: 0.1250079 second, the error code is: InvalidGatewayKey, activityId is: XXXXXXX and detailed error message is Client IP address is not valid private ip Cause Data factory couldn't access the public network thereby not able to reach out to the cloud to make the successful connection."

Cause

The issue could be caused by the VM in which you're trying to install the self-hosted IR. To connect to the cloud, ensure that public network access is enabled.

Resolution

Solution 1

To resolve the issue, do the following:

1. Go to the Factories - Update page.
2. At the upper right, select the **Try it** button.
3. Under **Parameters**, complete the required information.
4. Under **Body**, paste the following property:

   ```
   { "tags": { "publicNetworkAccess":"Enabled" } }
   ```

5. Select **Run** to run the function.
6. Under **Parameters**, complete the required information.
7. Under **Body**, paste the following property:

   ```
   { "tags": { "publicNetworkAccess":"Enabled" } }
   ```

8. Select **Run** to run the function.
9. Confirm that **Response Code: 200** is displayed. The property you pasted should be displayed in the

JSON definition as well.

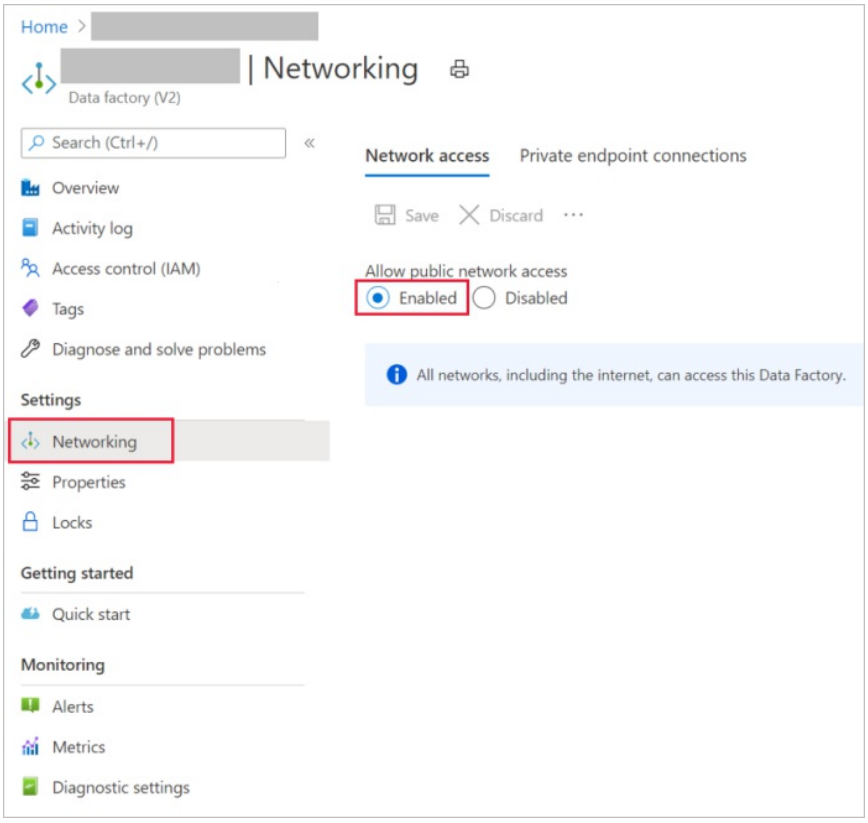
10. Add the IR authentication key again in the integration runtime.

Solution 2

To resolve the issue, go to Azure Private Link.

Try to enable public network access on the user interface, as shown in the following screenshot:

- Azure Data Factory
- Synapse Analytics



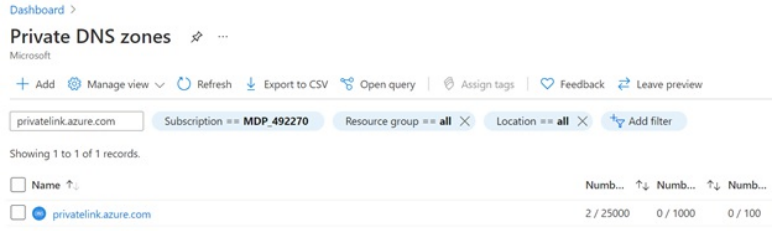
Service private DNS zone overrides Azure Resource Manager DNS resolution causing 'Not found' error

Cause

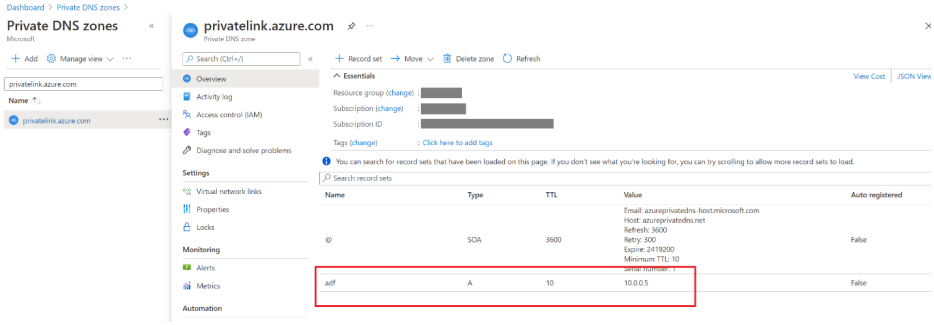
Both Azure Resource Manager and the service are using the same private zone creating a potential conflict on customer's private DNS with a scenario where the Azure Resource Manager records will not be found.

Solution

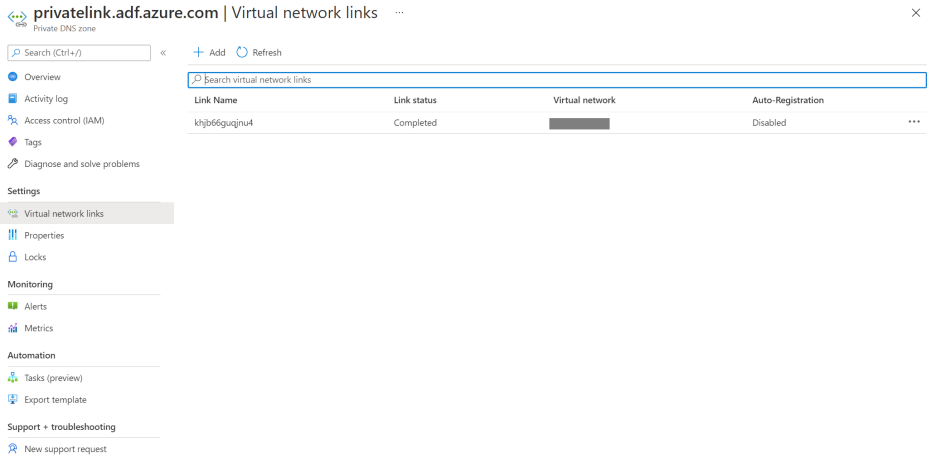
1. Find Private DNS zones **privatelink.azure.com** in Azure portal.



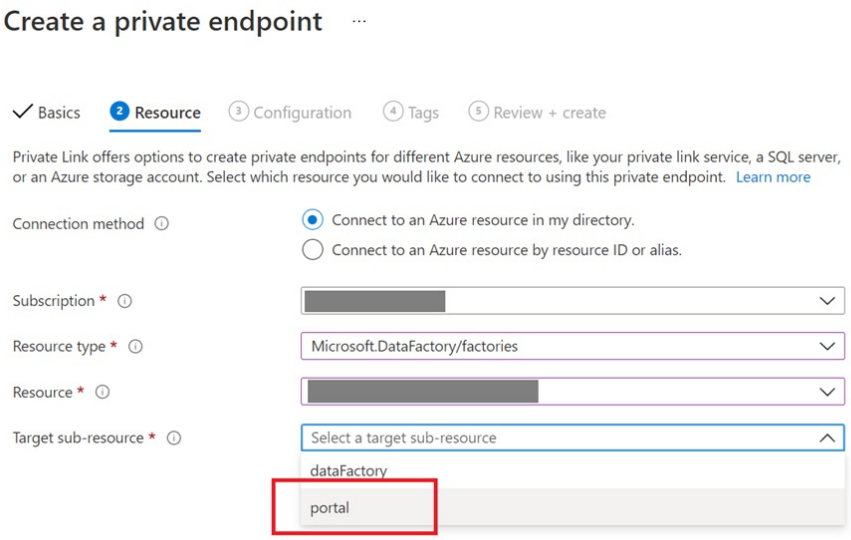
2. Check if there is an A record **adf**.



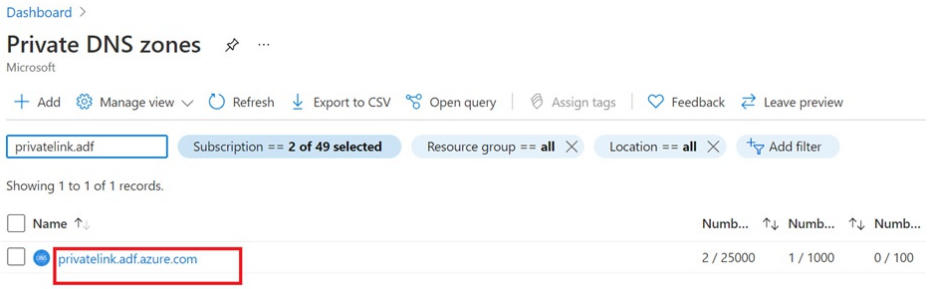
3. Go to **Virtual network links**, delete all records.



4. Navigate to your service in Azure portal and recreate the private endpoint for the portal.



5. Go back to Private DNS zones, and check if there is a new private DNS zone **privatelink.adf.azure.com**.



Connection error in public endpoint

Symptoms

When copying data with Azure Blob Storage account public access, pipeline runs randomly fail with following error.

For example: The Azure Blob Storage sink was using Azure IR (public, not Managed VNet) and the Azure SQL Database source was using the Managed VNet IR. Or source/sink use Managed VNet IR only with storage public access.

```
<LogProperties><Text>Invoke callback url with req:
"ErrorCode=AzureBlobFailedToCreateContainer,'Type=Microsoft.DataTransfer.Common.Shared.HybridDeliveryException,Message=Unable
to create Azure Blob container. Endpoint: XXXXXXX/, Container Name:
test.,Source=Microsoft.DataTransfer.ClientLibrary,''Type=Microsoft.WindowsAzure.Storage.StorageException,Message=Unable to
connect to the remote server,Source=Microsoft.WindowsAzure.Storage,''Type=System.Net.WebException,Message=Unable to connect
to the remote server,Source=System,''Type=System.Net.Sockets.SocketException,Message=A connection attempt failed because the
connected party did not properly respond after a period of time, or established connection failed because connected host has
failed to respond public ip:443,Source=System,'","Details":null}}</Text></LogProperties>.
```

Cause

The service may still use Managed VNet IR, but you could encounter such error because the public endpoint to Azure Blob Storage in Managed VNet is not reliable based on the testing result, and Azure Blob Storage and Azure Data Lake Gen2 are not supported to be connected through public endpoint from the service's Managed Virtual Network according to Managed virtual network & managed private endpoints.

Solution

- Having private endpoint enabled on the source and also the sink side when using the Managed VNet IR.
- If you still want to use the public endpoint, you can switch to public IR only instead of using the Managed VNet IR for the source and the sink. Even if you switch back to public IR, the service may still use the Managed VNet IR if the Managed VNet IR is still there.

Internal error while trying to Delete a data factory or Synapse workspace with Customer Managed Key (CMK) and User Assigned Managed Identity (UA-MI)

Symptoms

```
{\"error\":{\"code\":\"InternalError\",\"message\":\"Internal error has occurred.\"}}
```

Cause

If you are performing any operations related to CMK, you should complete all operations related to the service first, and then external operations (like Managed Identities or Key Vault operations). For example, if you want to delete all resources, you need to delete the service instance first, and then delete the key vault. If you delete the key vault first, this error will occur since the service can't read the required objects anymore, and it won't be able to validate if deletion is possible or not.

Solution

There are three possible ways to solve the issue. They are as follows:

- You revoked the service's access to Key vault where the CMK key was stored. You can reassign access to the following permissions: **Get, Unwrap Key, and Wrap Key**. These permissions are required to enable customer-managed keys. Please refer to Grant access to customer-managed keys. Once the permission is provided, you should be able to delete the service.
- Customer deleted Key Vault / CMK before deleting the service. CMK in the service should have "Soft Delete" enabled and "Purge Protect" enabled which has default retention policy of 90 days. You can restore the deleted key.
  Please review Recover deleted Key and Deleted Key Value
- User Assigned Managed Identity (UA-MI) was deleted before the service. You can recover from this by using REST API calls, you can do this in an http client of your choice in any programming language. If you have not anything already set up for REST API calls with Azure authentication, the easiest way to do this would be by using POSTMAN/Fiddler. Please follow following steps.

  1. Make a GET call using Method: GET Url like
     ```
     https://management.azure.com/subscriptions/YourSubscription/resourcegroups/YourResourceGroup/providers/Microsoft.DataFactory/factorie
     api-version=2018-06-01
     ```
  2. You need to create a new User Managed Identity with a different Name (same name may work, but just to be sure, it's safer to use a different name than the one in the GET response)
  3. Modify the encryption.identity property and identity.userassignedidentities to point to the newly created managed identity. Remove the clientId and principalId from the userAssignedIdentity object.
  4. Make a PUT call to the same url passing the new body. It is very important that you are passing whatever you got in the GET response, and only modify the identity. Otherwise they would override other settings unintentionally.
  5. After the call succeeds, you will be able to see the entities again and retry deleting.

Sharing Self-hosted Integration Runtime

Sharing a self-hosted IR from a different tenant is not supported

Symptoms

You might notice other data factories (on different tenants) as you're attempting to share the self-hosted IR from the UI, but you can't share it across data factories that are on different tenants.

Cause

The self-hosted IR can't be shared across tenants.

Next steps

For more help with troubleshooting, try the following resources:

- Private Link for Data Factory
- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack overflow forum for Data Factory

- Twitter information about Data Factory

Troubleshoot self-hosted integration runtime

8/19/2022 • 32 minutes to read • Edit Online

APPLIES TO: Azure Data Factory Azure Synapse Analytics

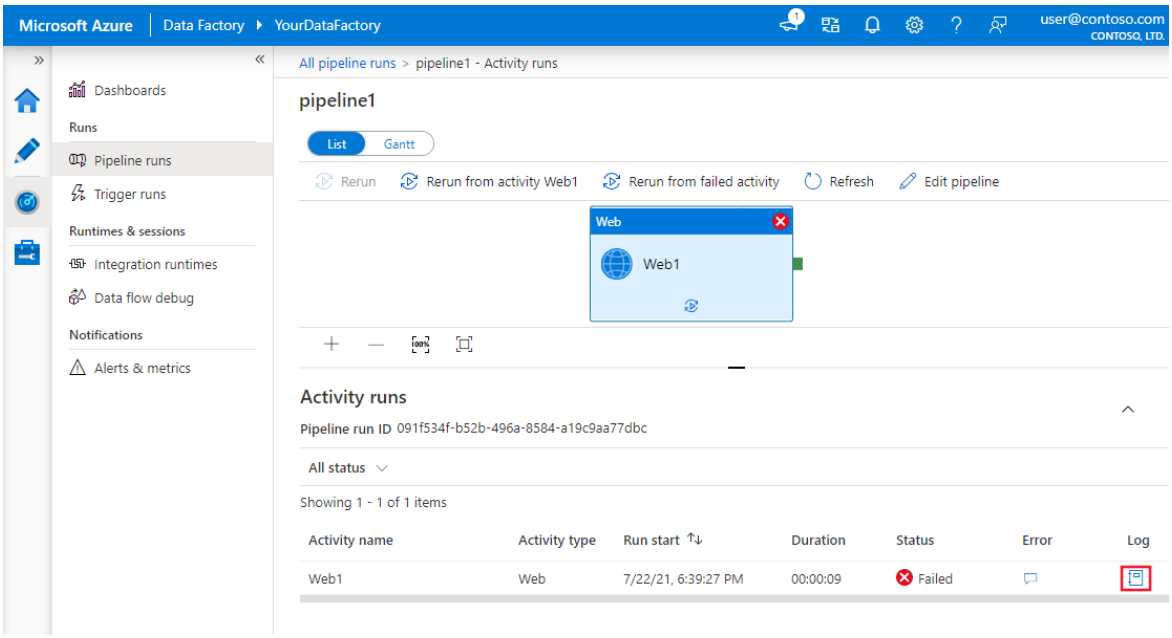
This article explores common troubleshooting methods for self-hosted integration runtime (IR) in Azure Data Factory and Synapse workspaces.

Gather self-hosted IR logs

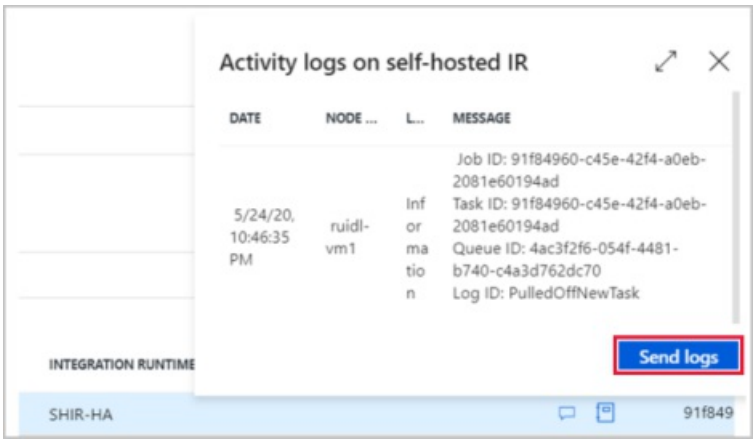
For failed activities that are running on a self-hosted IR or a shared IR, the service supports viewing and uploading error logs. To get the error report ID, follow the instructions here, and then enter the report ID to search for related known issues.

1. On the Monitor page for the service UI, select **Pipeline runs**.
2. Under **Activity runs**, in the **Error** column, select the highlighted button to display the activity logs, as shown in the following screenshot:

   - Azure Data Factory
   - Azure Synapse

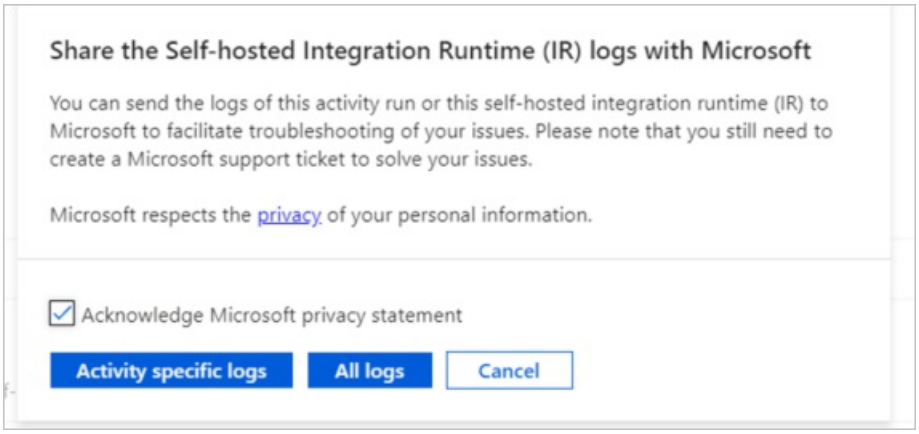


The activity logs are displayed for the failed activity run.



3. For further assistance, select **Send logs**.

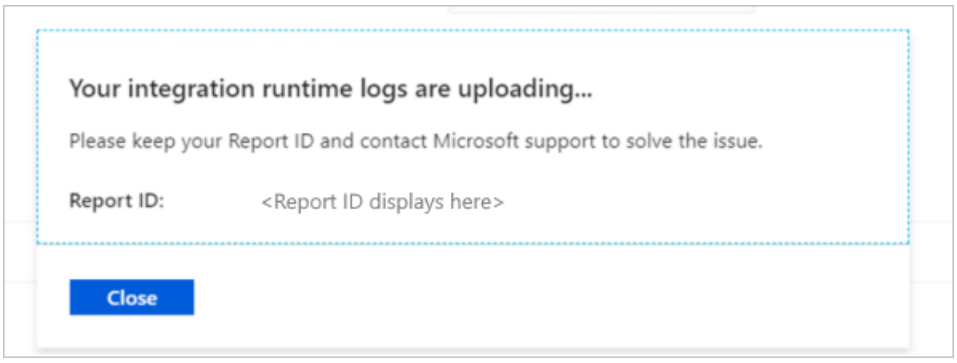
   The **Share the self-hosted integration runtime (IR) logs with Microsoft** window opens.



4. Select which logs you want to send.

   - For a *self-hosted IR*, you can upload logs that are related to the failed activity or all logs on the self-hosted IR node.
   - For a *shared IR*, you can upload only logs that are related to the failed activity.

5. When the logs are uploaded, keep a record of the Report ID for later use if you need further assistance to solve the issue.



> **NOTE**
>
> Log viewing and uploading requests are executed on all online self-hosted IR instances. If any logs are missing, make sure that all the self-hosted IR instances are online.

Self-hosted IR general failure or error

Out of memory issue

- Symptoms

  An OutOfMemoryException (OOM) error occurs when you try to run a lookup activity with a linked IR or a self-hosted IR.

- Cause

  A new activity can throw an OOM error if the IR machine experiences momentary high memory usage. The issue might be caused by a large volume of concurrent activity, and the error is by design.

- Resolution

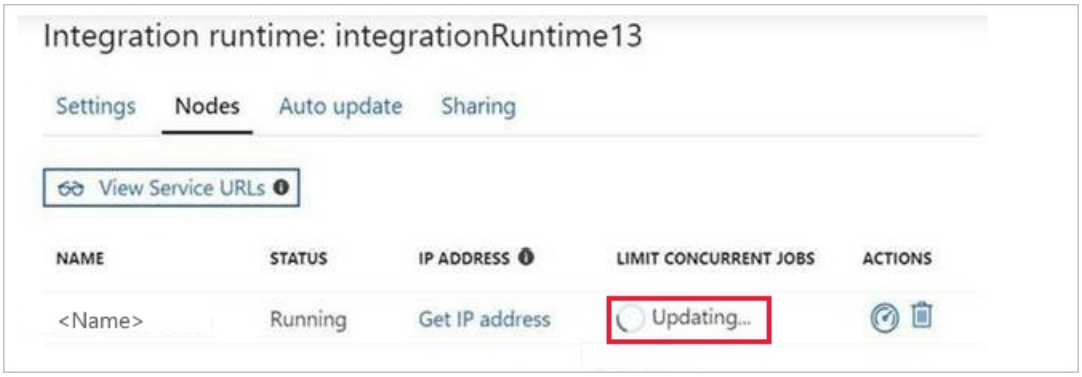
  Check the resource usage and concurrent activity execution on the IR node. Adjust the internal and trigger time of activity runs to avoid too much execution on a single IR node at the same time.

Concurrent jobs limit issue

- Symptoms

  When you try to increase the concurrent jobs limit from UI, the process hangs in *Updating* status.

  Example scenario: The maximum concurrent jobs value is currently set to 24, and you want to increase the count so that your jobs can run faster. The minimum value that you can enter is 3, and the maximum value is 32. You increase the value from 24 to 32 and then select the **Update** button. The process gets stuck in *Updating* status, as shown in the following screenshot. You refresh the page, and the value is still displayed as 24. It hasn't been updated to 32 as you had expected.



- Cause

  The limit on the number of concurrent jobs depends on the computer's logic core and memory. Try to adjust the value downward to a value such as 24, and then view the result.

  > **TIP**
  > - To learn more about logic core count and to determine your machine's logic core count, see Four ways to find the number of cores in your CPU on Windows 10.
  > - To learn how to calculate the math.log, go to the Logarithm calculator.

Self-hosted IR high availability (HA) SSL certificate issue

- Symptoms

  The self-hosted IR work node has reported the following error:

  "Failed to pull shared states from primary node net.tcp://abc.cloud.corp.Microsoft.com:8060/ExternalService.svc/. Activity ID: XXXXX The X.509 certificate CN=abc.cloud.corp.Microsoft.com, OU=test, O=Microsoft chain building failed. The certificate that was used has a trust chain that cannot be verified. Replace the certificate or change the certificateValidationMode. The revocation function was unable to check revocation because the

revocation server was offline."

- **Cause**

  When you handle cases that are related to an SSL/TLS handshake, you might encounter some issues related to certificate chain verification.

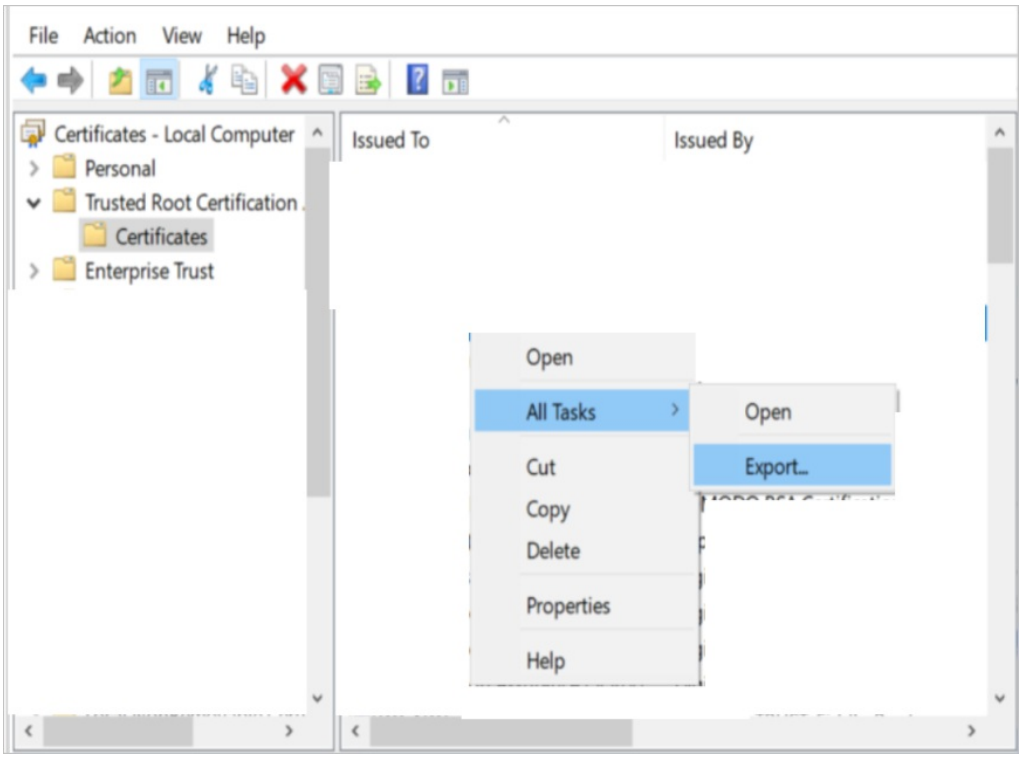
- **Resolution**

  - Here's a quick, intuitive way to troubleshoot an X.509 certificate chain build failure:

    1. Export the certificate, which needs to be verified. To do so, do the following:

       a. In Windows, select **Start**, start typing **certificates**, and then select **Manage computer certificates**.

       b. In File Explorer, on the left pane, search for the certificate that you want to check, right-click it, and then select **All tasks** > **Export**.

    2. Copy the exported certificate to the client machine.

    3. On the client side, in a Command Prompt window, run the following command. Be sure to replace *<certificate path>* and *<output txt file path>* with the actual paths.

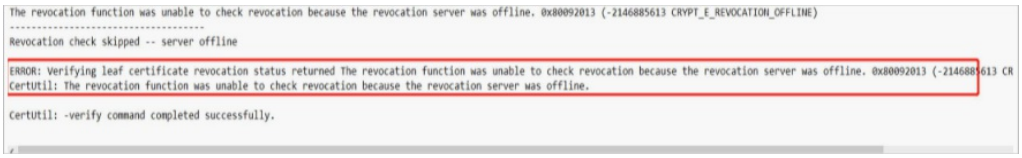
       ```
       Certutil -verify -urlfetch    <certificate path>    >     <output txt file path>
       ```

       For example:

       ```
       Certutil -verify -urlfetch c:\users\test\desktop\servercert02.cer >
       c:\users\test\desktop\Certinfo.txt
       ```

    4. Check for errors in the output TXT file. You can find the error summary at the end of the TXT file.

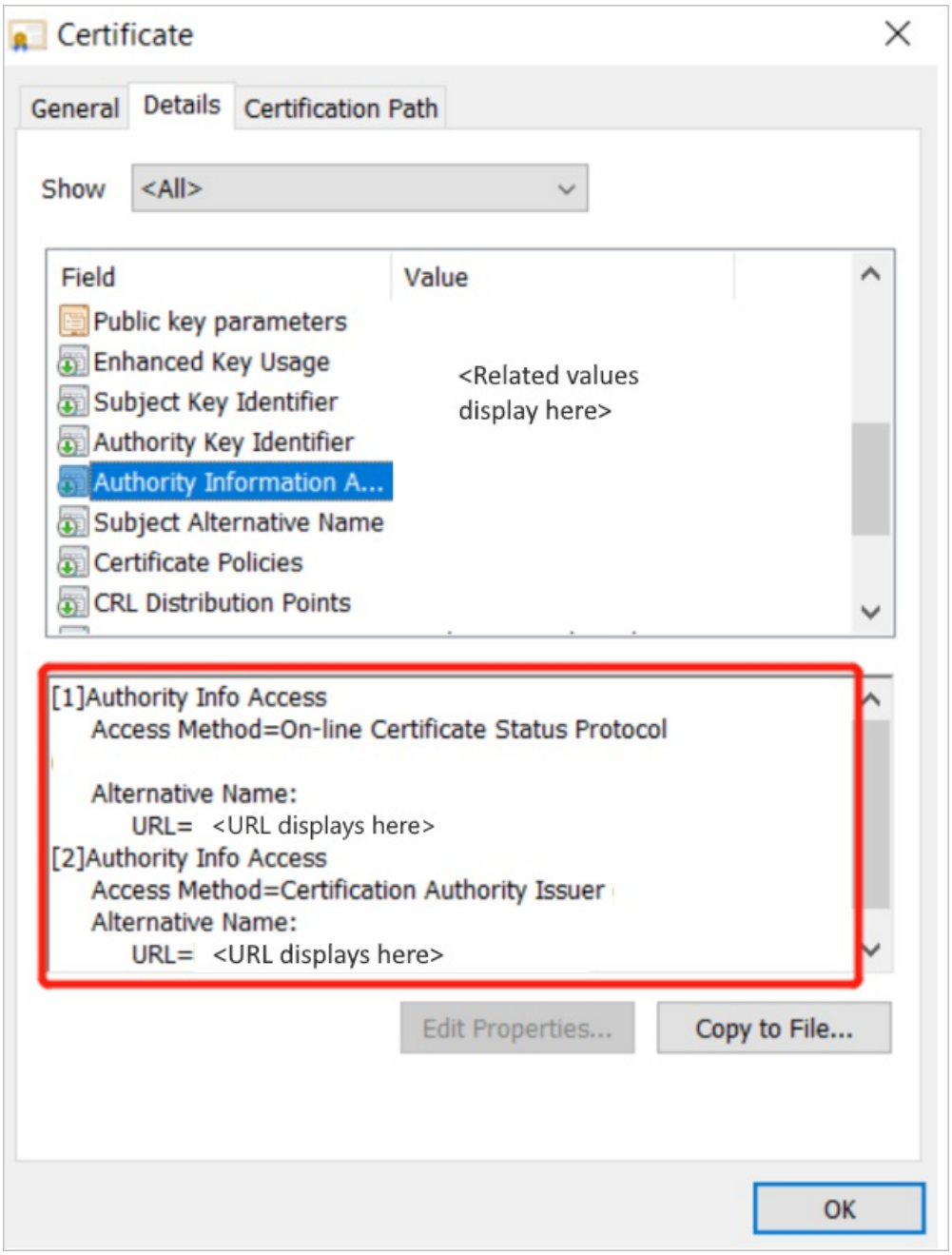
       For example:



If you don't see an error at the end of the log file, as shown in the following screenshot, you can consider that the certificate chain has been built successfully on the client machine.

- If an AIA (Authority Information Access), CDP (CRL Distribution Point), or OCSP (Online Certificate Status Protocol) file name extension is configured in the certificate file, you can check it in a more intuitive way:

  1. Get this information by checking the certificate details, as shown in the following screenshot:

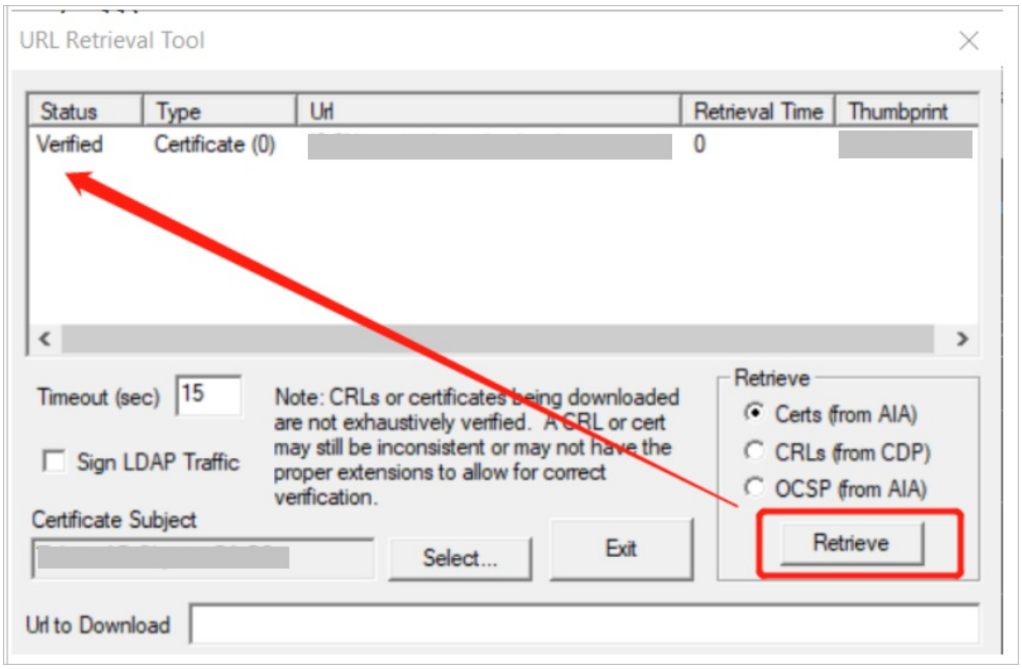


2. Run the following command. Be sure to replace *<certificate path>* with the actual path of the certificate.

```
Certutil   -URL    <certificate path>
```

   The URL Retrieval tool opens.

3. To verify certificates with AIA, CDP, and OCSP file name extensions, select **Retrieve**.



You've built the certificate chain successfully if the certificate status from AIA is *Verified* and the certificate status from CDP or OCSP is *Verified*.

If you fail when you try to retrieve AIA or CDP, work with your network team to get the client machine ready to connect to the target URL. It will be enough if either the HTTP path or the Lightweight Directory Access Protocol (LDAP) path can be verified.

Self-hosted IR could not load file or assembly

- Symptoms

  You get the following error message:

  "Could not load file or assembly 'XXXXXXXXXXXXXXXX, Version=4.0.2.0, Culture=neutral, PublicKeyToken=XXXXXXXXX' or one of its dependencies. The system cannot find the file specified. Activity ID: 92693b45-b4bf-4fc8-89da-2d3dc56f27c3"

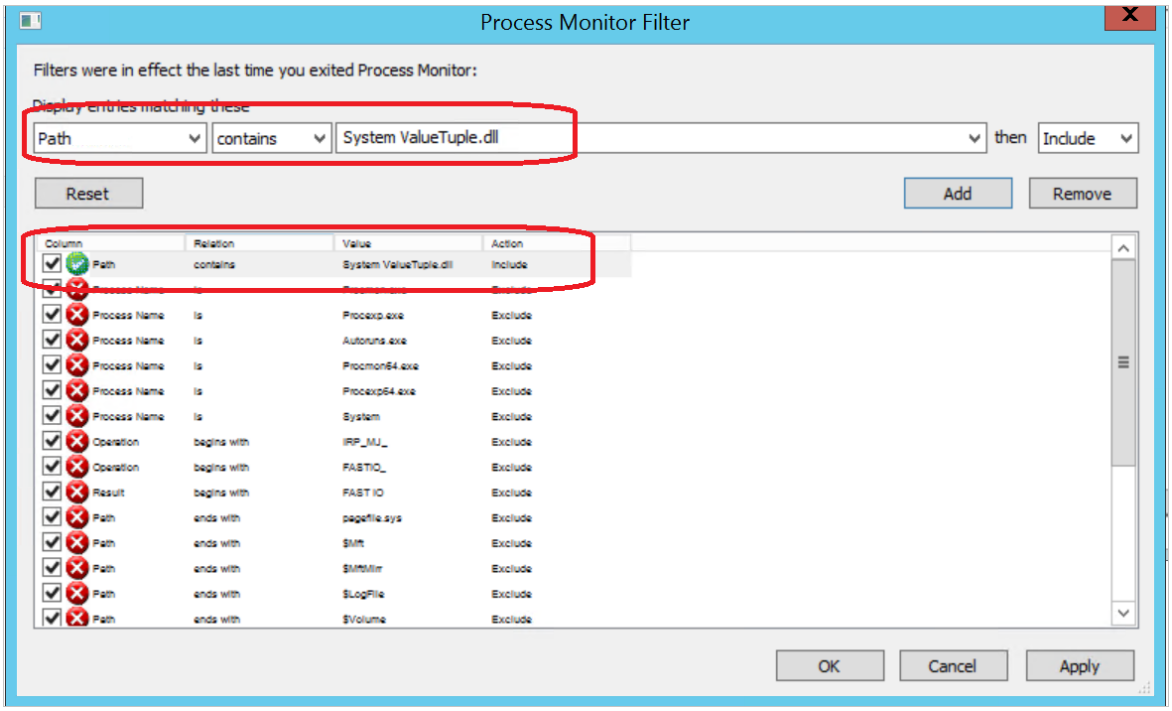
  Here is a more specific error message:

  "Could not load file or assembly 'System.ValueTuple, Version=4.0.2.0, Culture=neutral, PublicKeyToken=XXXXXXXXX' or one of its dependencies. The system cannot find the file specified. Activity ID: 92693b45-b4bf-4fc8-89da-2d3dc56f27c3"

- Cause

  In Process Monitor, you can view the following result:

  > **TIP**
  > In Process Monitor, you can set filters as shown in following screenshot.
  >
  > The preceding error message says that the DLL System.ValueTuple is not located in the related *Global Assembly Cache* (GAC) folder, in the *C:\Program Files\Microsoft Integration Runtime\4.0\Gateway* folder, or in the *C:\Program Files\Microsoft Integration Runtime\4.0\Shared* folder.
  >
  > Basically, the process loads the DLL first from the *GAC* folder, then from the *Shared* folder, and finally from the *Gateway* folder. Therefore, you can load the DLL from any path that's helpful.



- **Resolution**

  You'll find the *System.ValueTuple.dll* file in the *C:\Program Files\Microsoft Integration Runtime\4.0\Gateway\DataScan* folder. To resolve the issue, copy the *System.ValueTuple.dll* file to the *C:\Program Files\Microsoft Integration Runtime\4.0\Gateway* folder.

  You can use the same method to resolve other missing file or assembly issues.

- **More information about this issue**

  The reason why you see the *System.ValueTuple.dll* under *%windir%\Microsoft.NET\assembly* and *%windir%\assembly* is that this is a .NET behavior.

  In the following error, you can clearly see that the *System.ValueTuple* assembly is missing. This issue arises when the application tries to check the *System.ValueTuple.dll* assembly.

  "<LogProperties><ErrorInfo>[{"Code":0,"Message":"The type initializer for 'Npgsql.PoolManager' threw an exception.","EventType":0,"Category":5,"Data":{},"MsgId":null,"ExceptionType":"System.TypeInitializationException","Source":"Npgsql","StackTrace":"","InnerEventInfos":[{"Code":0,"Message":"Could not load file or assembly 'System.ValueTuple, Version=4.0.2.0, Culture=neutral, PublicKeyToken=XXXXXXXXX' or one of its dependencies. The system cannot find the file specified.","EventType":0,"Category":5,"Data":{},"MsgId":null,"ExceptionType":"System.IO.FileNotFoundException","Source":"Npgsql","StackTrace":"","InnerEventInfos":[]}]}]</ErrorInfo></LogProperties>"

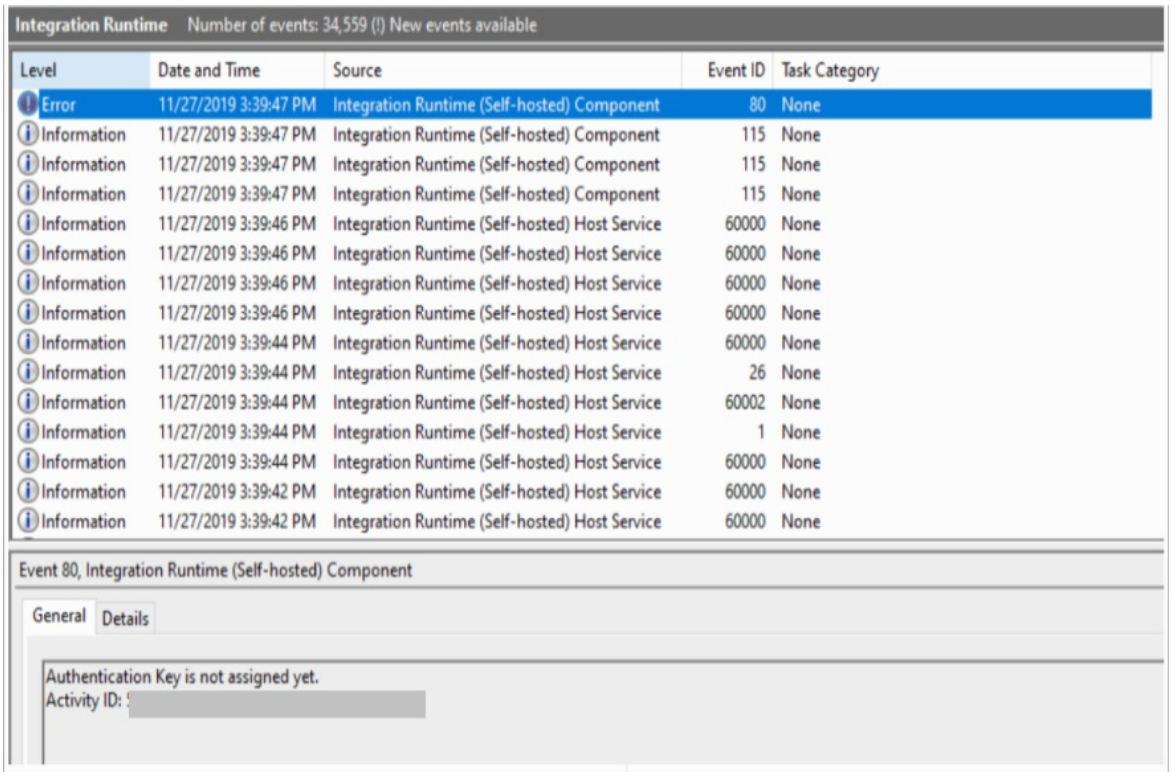
  For more information about GAC, see Global Assembly Cache.

Self-hosted integration runtime Authentication Key is missing

- **Symptoms**

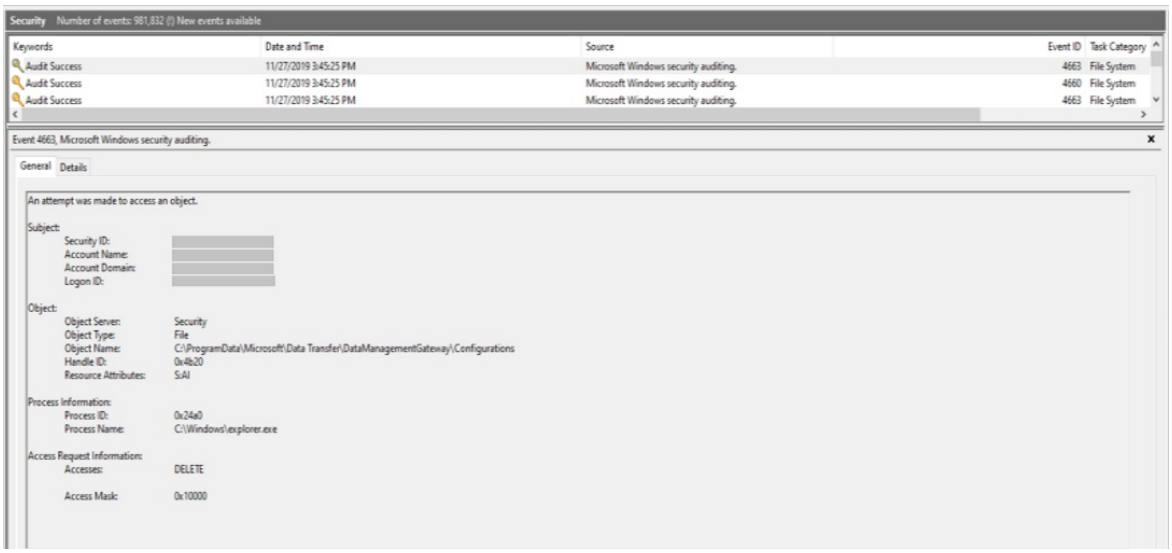
  The self-hosted integration runtime suddenly goes offline without an Authentication Key, and the Event Log displays the following error message:

  "Authentication Key is not assigned yet"



- **Cause**
  - The self-hosted IR node or logical self-hosted IR in the Azure portal was deleted.
  - A clean uninstall was performed.
- **Resolution**

  If neither of the preceding causes applies, you can go to the *%programdata%\Microsoft\Data Transfer\DataManagementGateway* folder to see whether the *Configurations* file has been deleted. If it was deleted, follow the instructions in the Netwrix article Detect who deleted a file from your Windows file servers.



Can't use self-hosted IR to bridge two on-premises datastores

- **Symptoms**

  After you create self-hosted IRs for both the source and destination datastores, you want to connect the two IRs to finish a copy activity. If the datastores are configured in different virtual networks, or the datastores can't understand the gateway mechanism, you receive either of the following errors:

  - "The driver of source cannot be found in destination IR"

- "The source cannot be accessed by the destination IR"

- **Cause**

  The self-hosted IR is designed as a central node of a copy activity, not a client agent that needs to be installed for each datastore.

  In this case, you should create the linked service for each datastore with the same IR, and the IR should be able to access both datastore through the network. It doesn't matter whether the IR is installed at the source datastore or the destination datastore, or on a third machine. If two linked services are created with different IRs but used in the same copy activity, the destination IR is used, and you need to install the drivers for both datastores on the destination IR machine.

- **Resolution**

  Install drivers for both the source and destination datastores on the destination IR, and make sure that it can access the source datastore.

  If the traffic can't pass through the network between two datastores (for example, they're configured in two virtual networks), you might not finish copying in one activity even with the IR installed. If you can't finish copying in a single activity, you can create two copy activities with two IRs, each in a VENT:

  - Copy one IR from datastore 1 to Azure Blob Storage
  - Copy another IR from Azure Blob Storage to datastore 2.

  This solution could simulate the requirement to use the IR to create a bridge that connects two disconnected datastores.

Credential sync issue causes credential loss from HA

- **Symptoms**

  If the data source credential "XXXXXXXXXX" is deleted from the current integration runtime node with payload, you receive the following error message:

  "When you delete the link service on Azure portal, or the task has the wrong payload, please create new link service with your credential again."

- **Cause**

  Your self-hosted IR is built in HA mode with two nodes, but the nodes aren't in a credentials sync state. This means that the credentials stored in the dispatcher node aren't synced to other worker nodes. If any failover happens from the dispatcher node to the worker node, and the credentials exist only in the previous dispatcher node, the task will fail when you're trying to access credentials, and you'll receive the preceding error.

- **Resolution**

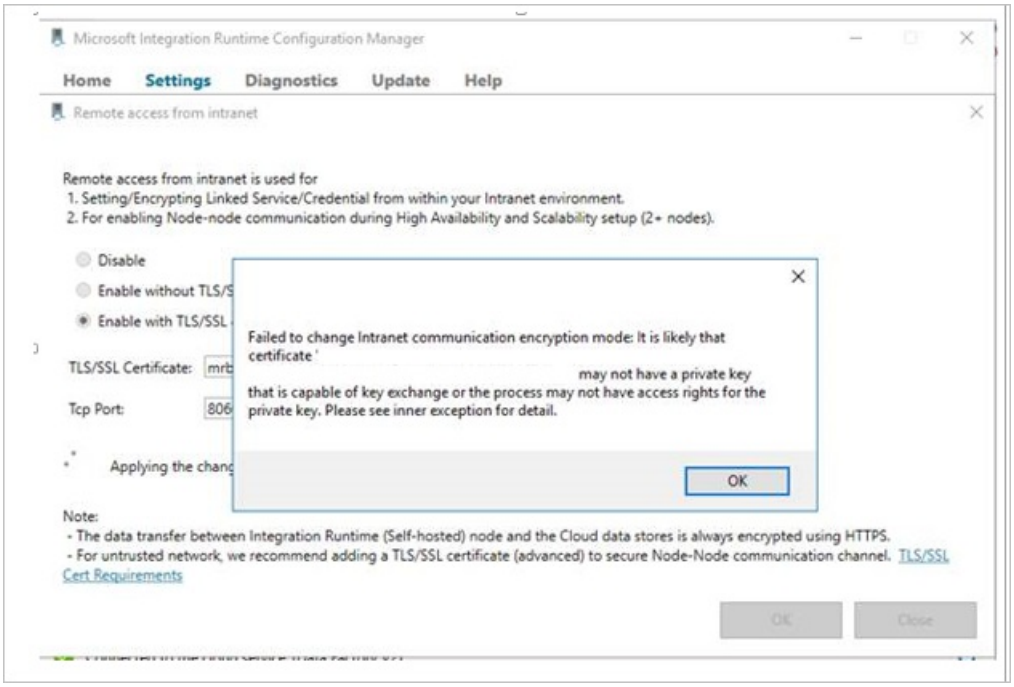
  The only way to avoid this issue is to make sure that the two nodes are in credentials sync state. If they aren't in sync, you have to reenter the credentials for the new dispatcher.

Can't choose the certificate because the private key is missing

- **Symptoms**

  - You've imported a PFX file to the certificate store.
  - When you selected the certificate through the IR Configuration Manager UI, you received the following error message:

    "Failed to change intranet communication encryption mode. It is likely that certificate '<*certificate name*>' may not have a private key that is capable of key exchange or the process may not have access rights for the private key. Please see inner exception for detail."



- Cause
  - The user account has a low privilege level and can't access the private key.
  - The certificate was generated as a signature but not as a key exchange.
- Resolution
  - To operate the UI, use an account with appropriate privileges for accessing the private key.
  - Import the certificate by running the following command:

```
certutil -importpfx FILENAME.pfx AT_KEYEXCHANGE
```

Self-hosted integration runtime nodes out of the sync issue

- Symptoms

  Self-hosted integration runtime nodes try to sync the credentials across nodes but get stuck in the process and encounter the error message below after a while:

  "The Integration Runtime (Self-hosted) node is trying to sync the credentials across nodes. It may take several minutes."

  > **NOTE**
  > If this error appears for over 10 minutes, please check the connectivity with the dispatcher node.

- Cause

  The reason is that the worker nodes do not have access to the private keys. This can be confirmed from the self-hosted integration runtime logs below:

```
[14]0460.3404::05/07/21-00:23:32.2107988 [System] A fatal error occurred when attempting to access the TLS server credential private key. The error code returned from the cryptographic module is 0x8009030D. The internal error state is 10001.
```

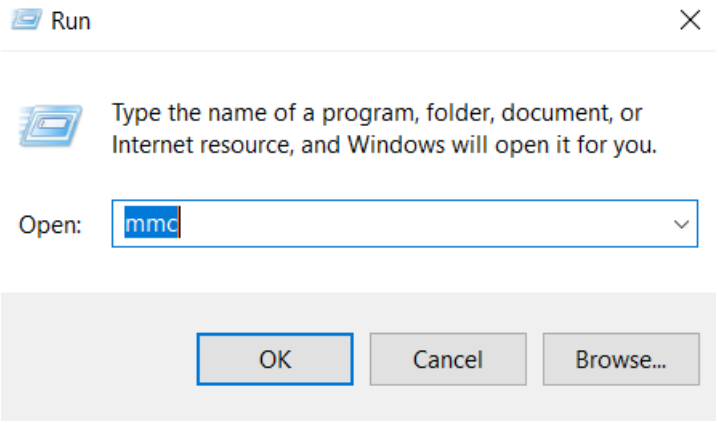
  You have no issue with the sync process when you use the service principal authentication in the linked service. However, when you switch the authentication type to account key, the syncing issue started. This is because the self-hosted integration runtime service runs under a service account (NT

SERVICE\DIAHostService) and it need to be added to the private key permissions.

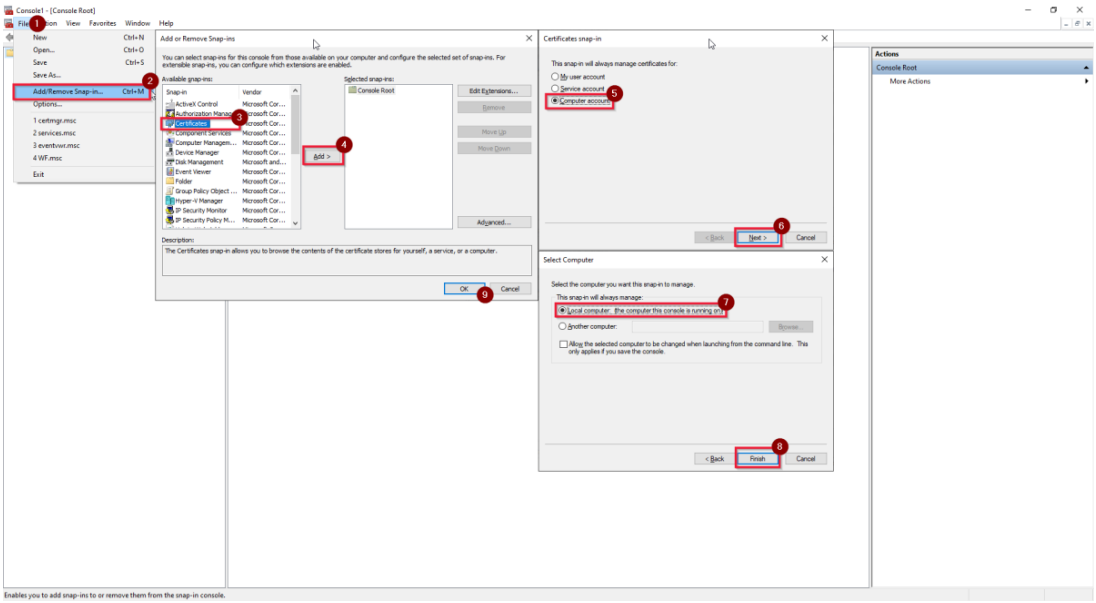
- **Resolution**

  To solve this issue, you need to add the self-hosted integration runtime service account (NT SERVICE\DIAHostService) to the private key permissions. You can apply the following steps:

  1. Open your Microsoft Management Console (MMC) Run Command.

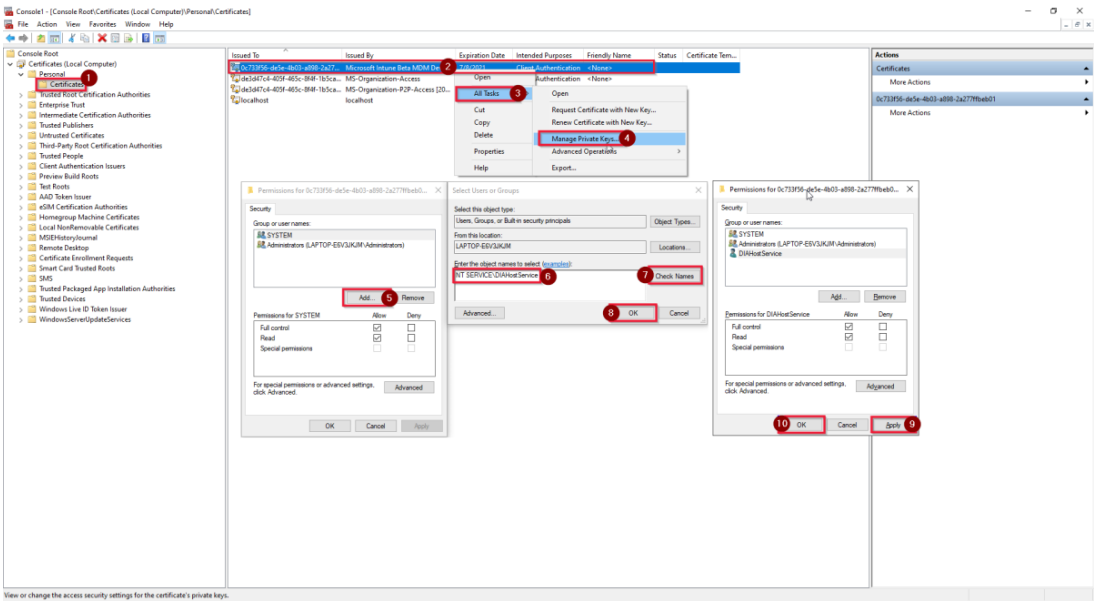
     

  2. In the MMC pane, apply the following steps:

     a. Select **File**.
     b. Choose **Add/Remove Snap-in** in th drop-down menu.
     c. Select **Certificates** in the "Available snap-ins" pane.
     d. Select **Add**.
     e. In the pop-up "Certificates snap-in" pane, choose **Computer account**.
     f. Select **Next**.
     g. In the "Select Computer" pane, choose **Local computer: the computer this console is running on**.
     h. Select **Finish**.
     i. Select **OK** in the "Add or Remove Snap-ins" pane.

  3. In the pane of MMC, move on with the following steps:



a. From the left folder list, select **Console Root -> Certificates (Local Computer) -> Personal -> Certificates**.
b. Right-click the **Microsoft Intune Beta MDM**.
c. Select **All Tasks** in the drop-down list.
d. Select **Manage Private Keys**.
e. Select **Add** under "Group or user names".
f. Select **NT SERVICE\DIAHostService** to grant it full control access to this certificate, apply and safe.
g. Select **Check Names** and then select **OK**.
h. In the "Permissions" pane, select **Apply** and then select **OK**.

UserErrorJreNotFound error message when you run a copy activity to Azure

- **Symptoms**

  When you try to copy content to Microsoft Azure by using a Java-based tool or program (for example, copying ORC or Parquet format files), you receive an error message that resembles the following:

  > ErrorCode=UserErrorJreNotFound,'Type=Microsoft.DataTransfer.Common.Shared.HybridDeliveryException,Message=Java Runtime Environment is not found. Go to `http://go.microsoft.com/fwlink/?LinkId=808605` to download and install on your Integration Runtime (Self-hosted) node machine. Note 64-bit Integration Runtime requires 64-bit JRE and 32-bit Integration Runtime requires 32-bit JRE.,Source=Microsoft.DataTransfer.Common,''Type=System.DllNotFoundException,Message=Unable to load DLL 'jvm.dll': The specified module could not be found. (Exception from HRESULT: 0x8007007E),Source=Microsoft.DataTransfer.Richfile.HiveOrcBridge

- **Cause**

  This issue occurs for either of the following reasons:

  - Java Runtime Environment (JRE) isn't installed correctly on your Integration Runtime server.
  - Your Integration Runtime server lacks the required dependency for JRE.

  By default, Integration Runtime resolves the JRE path by using registry entries. Those entries should be automatically set during JRE installation.

- **Resolution**

Follow the steps in this section carefully. Serious problems might occur if you modify the registry incorrectly. Before you modify it, back up the registry for restoration in case problems occur.

To fix this issue, follow these steps to verify the status of the JRE installation:

1. Make sure that Integration Runtime (Diahost.exe) and JRE are installed on the same platform. Check the following conditions:

   - 64-bit JRE for 64-bit ADF Integration Runtime should be installed in the folder: `C:\Program Files\Java\`

     > **NOTE**
     > The folder is not `C:\Program Files (x86)\Java\`

   - JRE 7 and JRE 8 are both compatible for this copy activity. JRE 6 and versions that are earlier than JRE 6 have not been validated for this use.

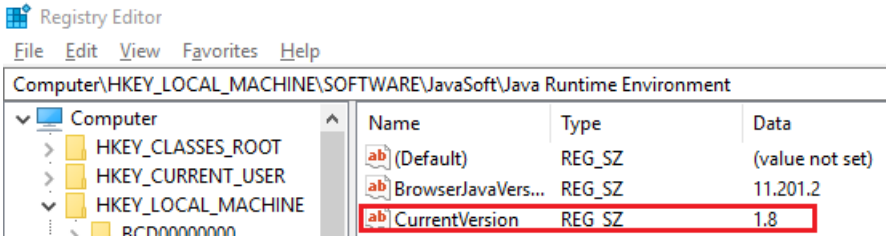
2. Check the registry for the appropriate settings. To do this, follow these steps:

   a. In the **Run** menu, type **Regedit**, and then press Enter.

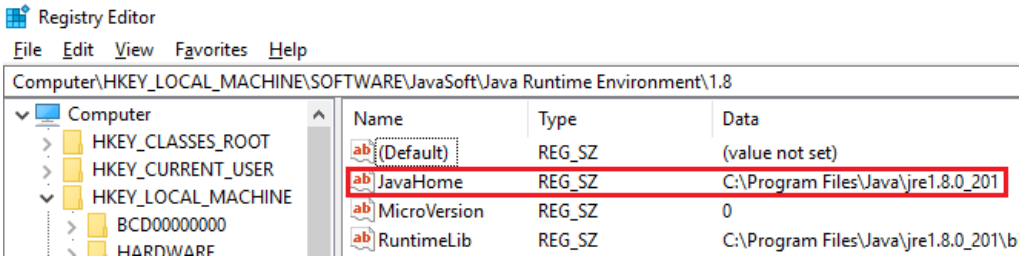
   b. In the navigation pane, locate the following subkey:

   `HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment`.

   In the **Details** pane, there should be a Current Version entry that shows the JRE version (for example, 1.8).

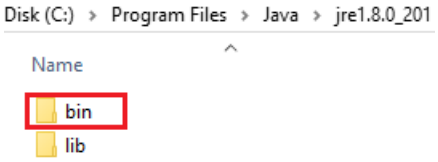


   c. In the navigation pane, locate a subkey that is an exact match for the version (for example 1.8) under the JRE folder. In the details pane, there should be a **JavaHome** entry. The value of this entry is the JRE installation path.



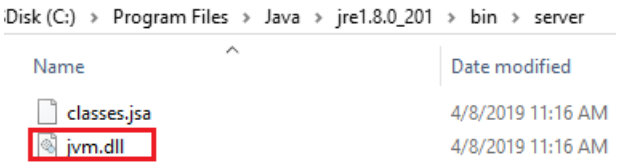
3. Locate the bin\server folder in the following path:

   `C:\Program Files\Java\jre1.8.0_74`



4. Check whether this folder contains a jvm.dll file. If it does not, check for the file in the `bin\client`

folder.



NOTE

- If any of these configurations are not as described in these steps, use the JRE windows installer to fix the problems.
- If all the configurations in these steps are correct as described, there may be a VC++ runtime library missing in the system. You can fix this problem by installing the VC++ 2010 Redistributable Package.

Self-hosted IR setup

Integration runtime registration error

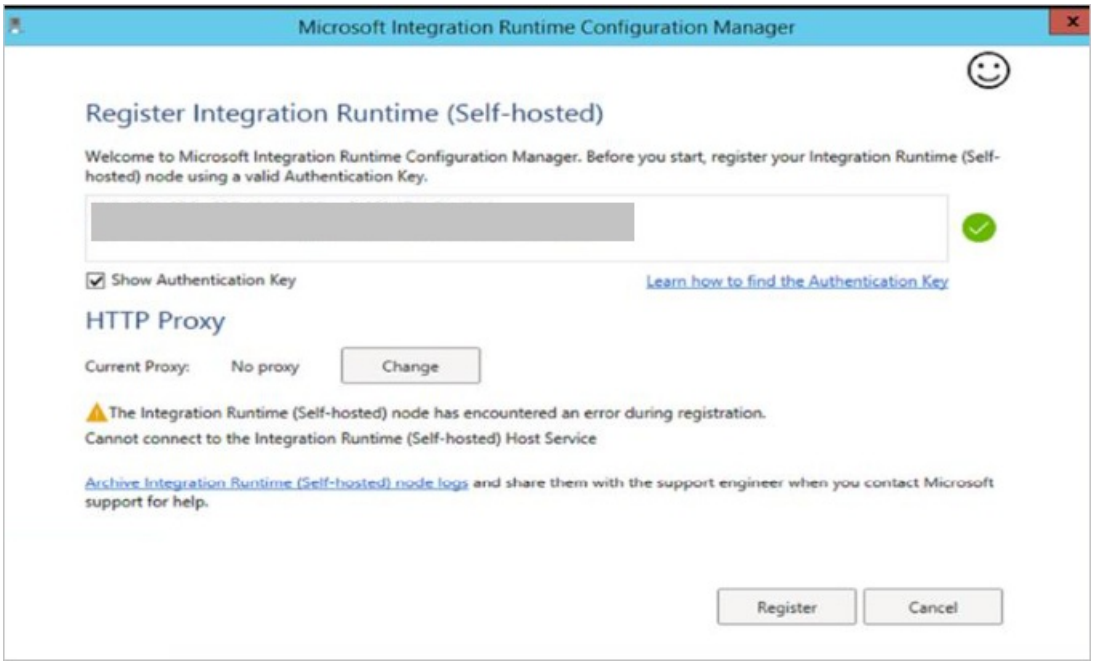
- Symptoms

  You might occasionally want to run a self-hosted IR in a different account for either of the following reasons:

  - Company policy disallows the service account.
  - Some authentication is required.

  After you change the service account on the service pane, you might find that the integration runtime stops working, and you get the following error message:

  "The Integration Runtime (Self-hosted) node has encountered an error during registration. Cannot connect to the Integration Runtime (Self-hosted) Host Service."



- Cause

  Many resources are granted only to the service account. When you change the service account to another account, the permissions of all dependent resources remain unchanged.

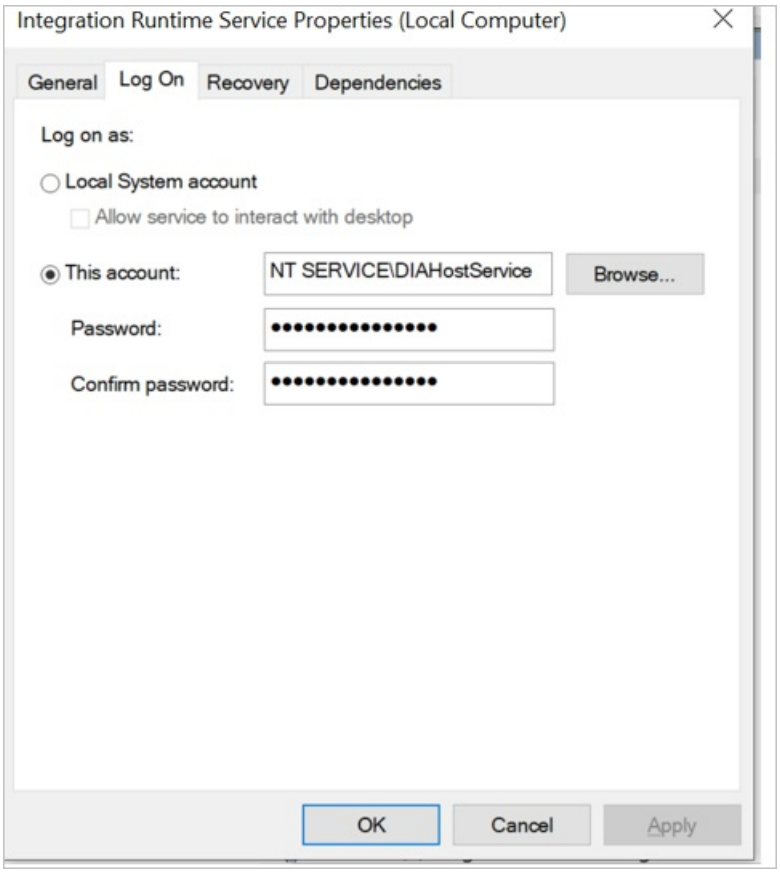
- Resolution

Go to the integration runtime event log to check the error.

```
DEBUG:
A runtime error has occurred
Access to the path 'C:\ProgramData\Microsoft\Data Transfer\DataManagementGateway\5s0u2ut1.ecv' is denied.

Exception detail:

System.UnauthorizedAccessException: Access to the path 'C:\ProgramData\Microsoft\Data Transfer\DataManagementGateway\5s0u2ut1.ecv' is denied.
  at System.IO.__Error.WinIOError(Int32 errorCode, String maybeFullPath)
  at System.IO.FileStream.Init(String path, FileMode mode, FileAccess access, Int32 rights, Boolean useRights, FileShare share, Int32 bufferSize, FileOptions options,
SECURITY_ATTRIBUTES secAttrs, String msgPath, Boolean bFromProxy, Boolean useLongPath, Boolean checkHost)
  at System.IO.FileStream..ctor(String path, FileMode mode, FileAccess access, FileShare share)
  at Microsoft.DataTransfer.GatewayCommon.EncryptedSerializer.WriteToFile(Byte[] encryptedData)
  at Microsoft.DataTransfer.GatewayCommon.PropertiesPersistent.SetProperty(PropertyValue value, Boolean notify)
  at Microsoft.DataTransfer.DIAgentHost.HostService.OnStart(Boolean isRestart, String gatewayDll, Action exit)
```

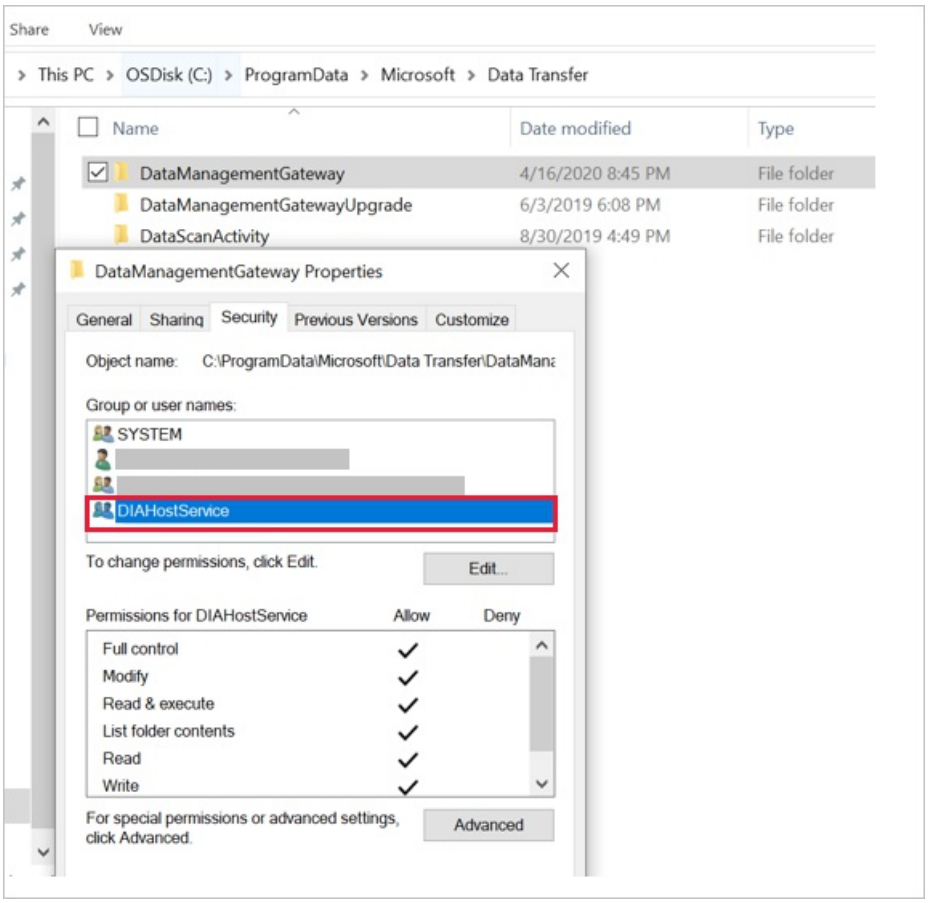
- If the error in the event log is "UnauthorizedAccessException," do the following:

  1. Check the *DIAHostService* logon service account in the Windows service panel.



  2. Check to see whether the logon service account has read/write permissions for the *%programdata%\Microsoft\DataTransfer\DataManagementGateway* folder.

     - By default, if the service logon account hasn't been changed, it should have read/write permissions.



- If you've changed the service logon account, mitigate the issue by doing the following:

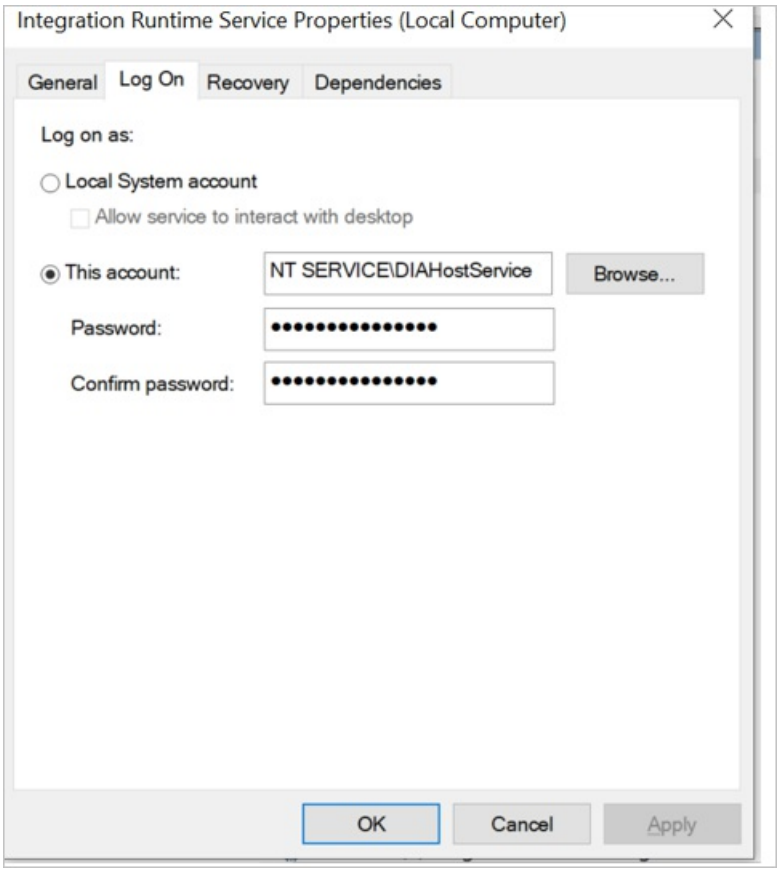
  a. Perform a clean uninstallation of the current self-hosted IR.
  b. Install the self-hosted IR bits.
  c. Change the service account by doing the following:

  i. Go to the self-hosted IR installation folder, and then switch to the *Microsoft Integration Runtime\4.0\Shared* folder.
  ii. Open a Command Prompt window by using elevated privileges. Replace *<user>* and *<password>* with your own username and password, and then run the following command:
  `dmgcmd.exe -SwitchServiceAccount "<user>" "<password>"`
  iii. If you want to change to the LocalSystem account, be sure to use the correct format for this account: `dmgcmd.exe -SwitchServiceAccount "NT Authority\System" ""`
  Do *not* use this format: `dmgcmd.exe -SwitchServiceAccount "LocalSystem" ""`
  iv. Optionally, because Local System has higher privileges than Administrator, you can also directly change it in "Services".
  v. You can use a local/domain user for the IR service logon account.

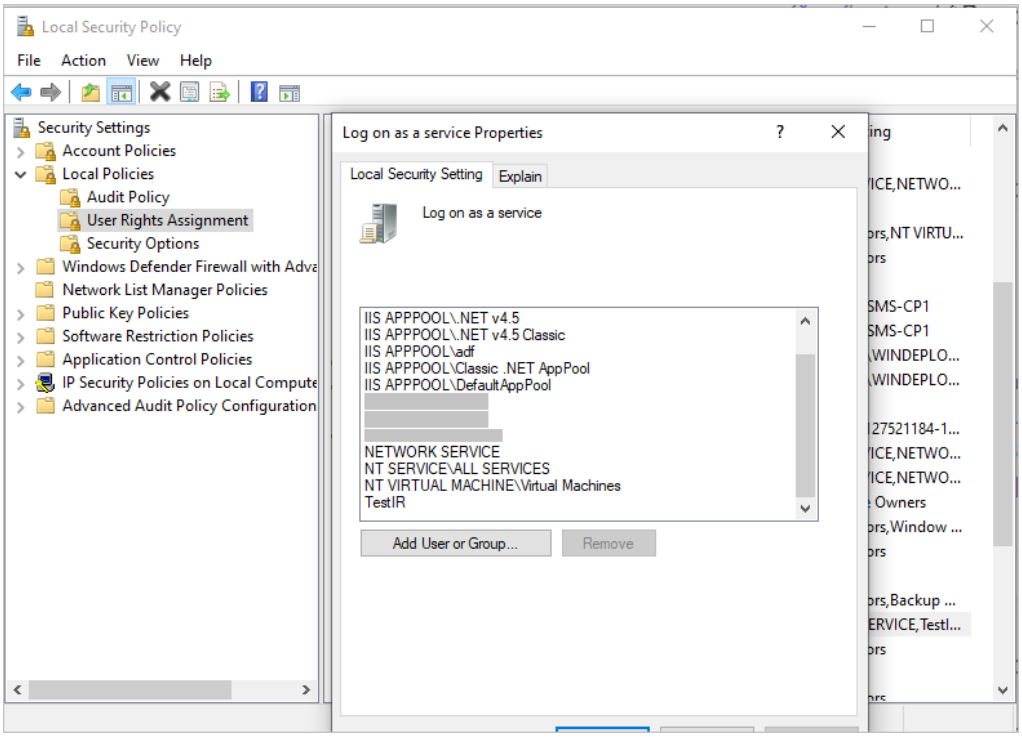
  d. Register the integration runtime.

- If the error is "Service 'Integration Runtime Service' (DIAHostService) failed to start. Verify that you have sufficient privileges to start system services," do the following:

  1. Check the *DIAHostService* logon service account in the Windows service panel.



2. Check to see whether the logon service account has **Log on as a service** permission to start the Windows service:



- **More information**

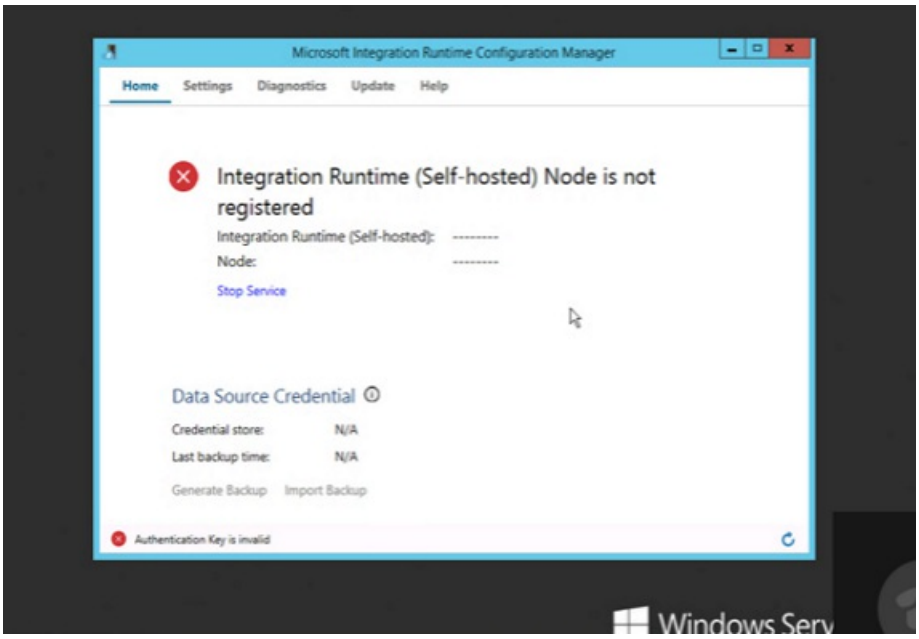
  If neither of the preceding two resolution patterns applies in your case, try to collect the following Windows event logs:

  - Applications and Services Logs > Integration Runtime
  - Windows Logs > Application

Can't find the Register button to register a self-hosted IR

- Symptoms

  When you register a self-hosted IR, the **Register** button isn't displayed on the Configuration Manager pane.



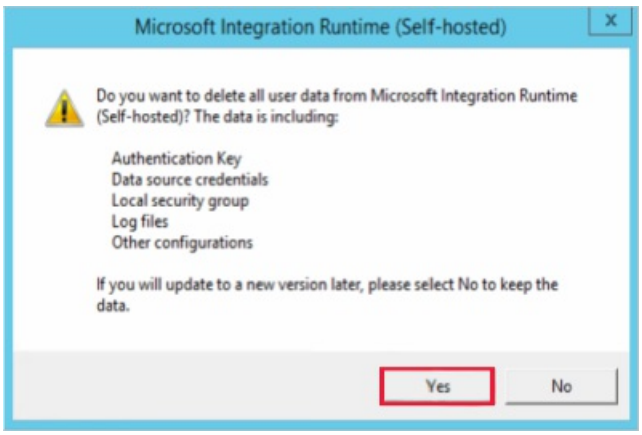
- Cause

  As of the release of Integration Runtime 3.0, the **Register** button on existing integration runtime nodes has been removed to enable a cleaner and more secure environment. If a node has been registered to an integration runtime, whether it's online or not, re-register it to another integration runtime by uninstalling the previous node, and then install and register the node.

- Resolution

  1. In Control Panel, uninstall the existing integration runtime.

     > **IMPORTANT**
     > In the following process, select **Yes**. Do not keep data during the uninstallation process.

  2. If you don't have the integration runtime installer MSI file, go to download center to download the latest integration runtime.
  3. Install the MSI file, and register the integration runtime.

Unable to register the self-hosted IR because of localhost

- Symptoms

You're unable to register the self-hosted IR on a new machine when you use get_LoopbackIpOrName.

Debug: A runtime error has occurred. The type initializer for 'Microsoft.DataTransfer.DIAgentHost.DataSourceCache' threw an exception. A non-recoverable error occurred during a database lookup.

Exception detail: System.TypeInitializationException: The type initializer for 'Microsoft.DataTransfer.DIAgentHost.DataSourceCache' threw an exception. ---> System.Net.Sockets.SocketException: A non-recoverable error occurred during a database lookup at System.Net.Dns.GetAddrInfo(String name).

- **Cause**

  The issue usually occurs when the localhost is being resolved.

- **Resolution**

  Use localhost IP address 127.0.0.1 to host the file and resolve the issue.

Self-hosted setup failed

- **Symptoms**

  You're unable to uninstall an existing IR, install a new IR, or upgrade an existing IR to a new IR.

- **Cause**

  The integration runtime installation depends on the Windows Installer service. You might experience installation problems for the following reasons:

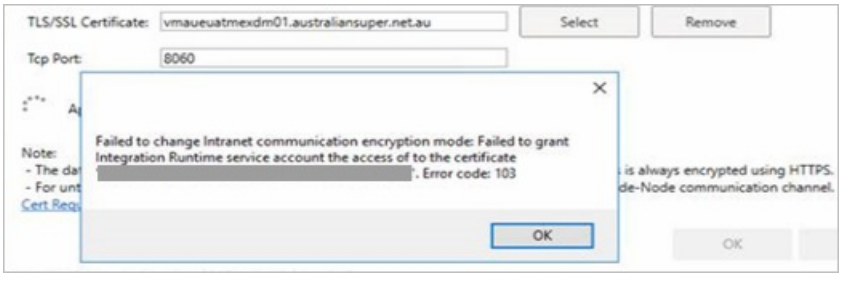
  - Insufficient available disk space.
  - Lack of permissions.
  - The Windows NT service is locked.
  - CPU utilization is too high.
  - The MSI file is hosted in a slow network location.
  - Some system files or registries were touched unintentionally.

The IR service account failed to fetch certificate access

- **Symptoms**

  When you install a self-hosted IR via Microsoft Integration Runtime Configuration Manager, a certificate with a trusted certificate authority (CA) is generated. The certificate couldn't be applied to encrypt communication between two nodes, and the following error message is displayed:

  "Failed to change Intranet communication encryption mode: Failed to grant Integration Runtime service account the access of to the certificate '<*certificate name*>'. Error code 103"



- **Cause**

  The certificate is using key storage provider (KSP) storage, which is not supported yet. To date, self-hosted IR supports only cryptographic service provider (CSP) storage.

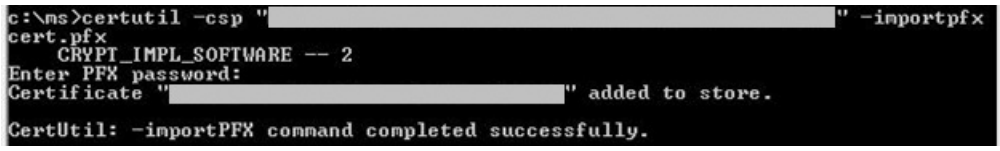
- **Resolution**

  We recommend that you use CSP certificates in this case.

  Solution 1

  To import the certificate, run the following command:

  ```
  Certutil.exe -CSP "CSP or KSP" -ImportPFX FILENAME.pfx
  ```

  Solution 2

  To convert the certificate, run the following commands:

  ```
  openssl pkcs12 -in .\xxxx.pfx -out .\xxxx_new.pem -password pass: <EnterPassword>
  openssl pkcs12 -export -in .\xxxx_new.pem -out xxxx_new.pfx
  ```

  Before and after conversion:

Self-hosted integration runtime version 5.x

For the upgrade to version 5.x of the self-hosted integration runtime, we require **.NET Framework Runtime 4.7.2** or later. On the download page, you'll find download links for the latest 4.x version and the latest two 5.x versions.

For Azure Data Factory v2 and Azure Synapse customers:

- If automatic update is on and you've already upgraded your .NET Framework Runtime to 4.7.2 or later, the self-hosted integration runtime will be automatically upgraded to the latest 5.x version.
- If automatic update is on and you haven't upgraded your .NET Framework Runtime to 4.7.2 or later, the self-hosted integration runtime won't be automatically upgraded to the latest 5.x version. The self-hosted integration runtime will stay in the current 4.x version. You can see a warning for a .NET Framework Runtime upgrade in the portal and the self-hosted integration runtime client.
- If automatic update is off and you've already upgraded your .NET Framework Runtime to 4.7.2 or later, you can manually download the latest 5.x and install it on your machine.
- If automatic update is off and you haven't upgraded your .NET Framework Runtime to 4.7.2 or later. When you try to manually install self-hosted integration runtime 5.x and register the key, you will be required to upgrade your .NET Framework Runtime version first.

For Azure Data Factory v1 customers:

- Self-hosted integration runtime 5.X doesn't support Azure Data Factory v1.
- The self-hosted integration runtime will be automatically upgraded to the latest version of 4.x. And the latest version of 4.x won't expire.
- If you try to manually install self-hosted integration runtime 5.x and register the key, you'll be notified that self-hosted integration runtime 5.x doesn't support Azure Data Factory v1.

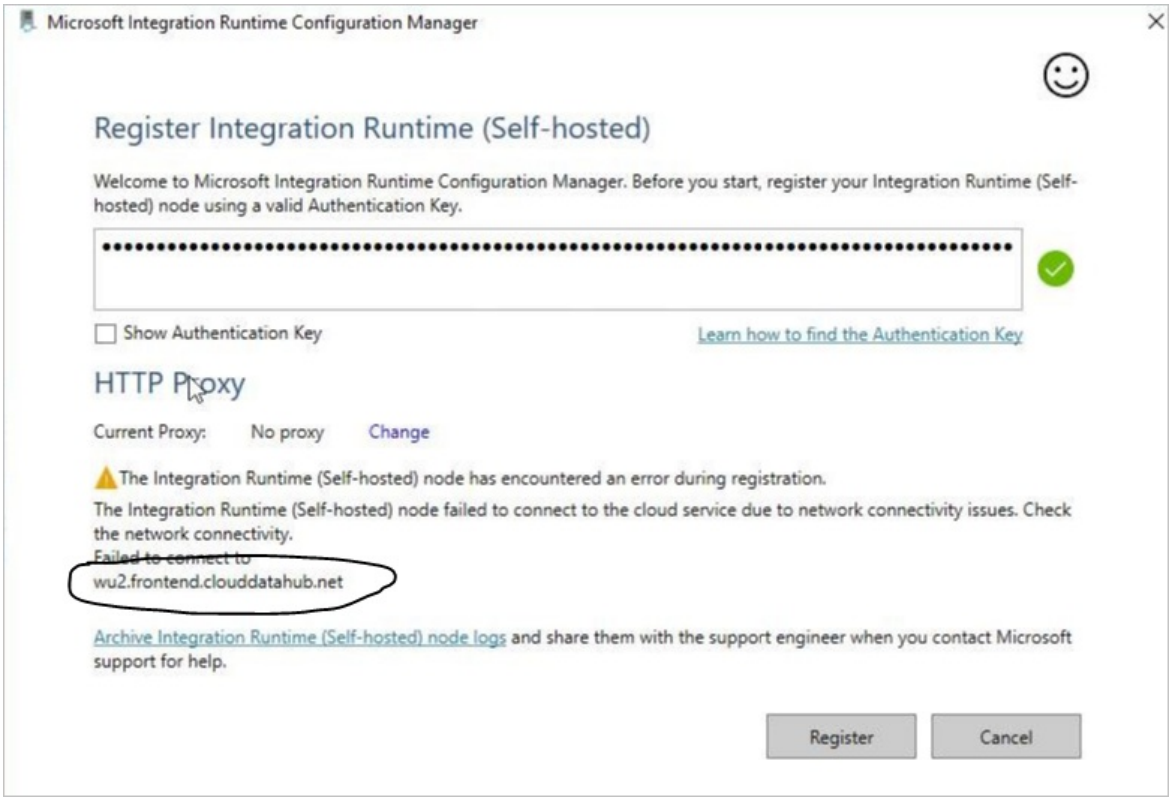
Self-hosted IR connectivity issues

Self-hosted integration runtime can't connect to the cloud service

- Symptoms

  When you attempt to register the self-hosted integration runtime, Configuration Manager displays the following error message:

  "The Integration Runtime (Self-hosted) node has encountered an error during registration."



- Cause

  The self-hosted IR can't connect to the service back end. This issue is usually caused by network settings in the firewall.

- Resolution

  1. Check to see whether the integration runtime service is running. If it is, go to step 2.

  2. If no proxy is configured on the self-hosted IR, which is the default setting, run the following PowerShell command on the machine where the self-hosted integration runtime is installed:

```
(New-Object System.Net.WebClient).DownloadString("https://wu2.frontend.clouddatahub.net/")
```

> **NOTE**
> The service URL might vary, depending on the location of your data factory or Synapse workspace instance. To find the service URL, use the Manage page of the UI in your data factory or Azure Synapse instance to find **Integration runtimes** and click your self-hosted IR to edit it. There select the **Nodes** tab and click **View Service URLs**.

The following is the expected response:

```
Exception calling "DownloadString" with "1" argument(s): "The remote server returned an error: (403) Forbidden."
At line:1 char:2
+  (New-Object System.Net.WebClient).DownloadString("https://eu.fronten ...
+  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : WebException
```

3. If you don't receive the response you had expected, use one of the following methods, as appropriate:

   - If you receive a "Remote name could not be resolved" message, there's a Domain Name System (DNS) issue. Contact your network team to fix the issue.
   - If you receive an "ssl/tls cert is not trusted" message, check the certificate to see whether it's trusted on the machine, and then install the public certificate by using Certificate Manager. This action should mitigate the issue.
   - Go to **Windows** > **Event viewer (logs)** > **Applications and Services Logs** > **Integration Runtime**, and check for any failure that's caused by DNS, a firewall rule, or company network settings. If you find such a failure, forcibly close the connection. Because every company has its own customized network settings, contact your network team to troubleshoot these issues.

4. If "proxy" has been configured on the self-hosted integration runtime, verify that your proxy server can access the service endpoint. For a sample command, see PowerShell, web requests, and proxies.

```
$user = $env:username
$webproxy = (get-itemproperty 'HKCU:\Software\Microsoft\Windows\CurrentVersion\Internet
Settings').ProxyServer
$pwd = Read-Host "Password?" -assecurestring
$proxy = new-object System.Net.WebProxy
$proxy.Address = $webproxy
$account = new-object System.Net.NetworkCredential($user,
[Runtime.InteropServices.Marshal]::PtrToStringAuto([Runtime.InteropServices.Marshal]::SecureSt
ringToBSTR($pwd)), "")
$proxy.credentials = $account
$url = "https://wu2.frontend.clouddatahub.net/"
$wc = new-object system.net.WebClient
$wc.proxy = $proxy
$webpage = $wc.DownloadData($url)
$string = [System.Text.Encoding]::ASCII.GetString($webpage)
$string
```

The following is the expected response:

```
Exception calling "DownloadString" with "1" argument(s): "The remote server returned an error: (403) Forbidden."
At line:1 char:2
+  (New-Object System.Net.WebClient).DownloadString("https://eu.fronten ...
+  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : WebException
```

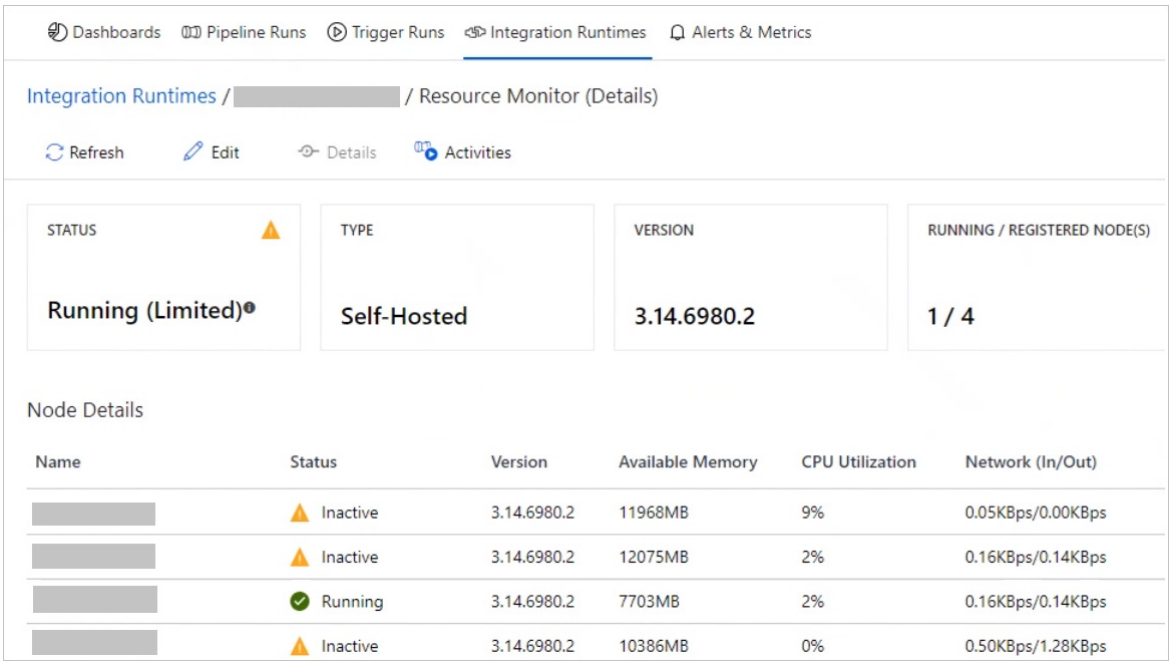
> **NOTE**
>
> Proxy considerations:
>
> - Check to see whether the proxy server needs to be put on the Safe Recipients list. If so, make sure these domains are on the Safe Recipients list.
> - Check to see whether SSL/TLS certificate "wu2.frontend.clouddatahub.net/" is trusted on the proxy server.
> - If you're using Active Directory authentication on the proxy, change the service account to the user account that can access the proxy as "Integration Runtime Service."

Error message: Self-hosted integration runtime node/logical self-hosted IR is in Inactive/ "Running (Limited)"

state

- Cause

  The self-hosted integrated runtime node might have a status of **Inactive**, as shown in the following screenshot:



  This behavior occurs when nodes can't communicate with each other.

- Resolution

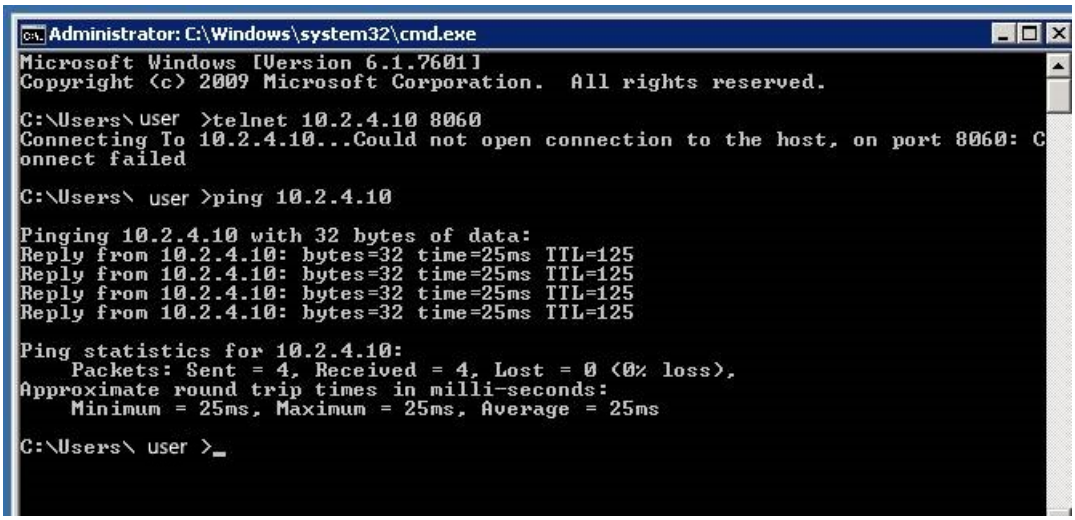
  1. Log in to the node-hosted virtual machine (VM). Under **Applications and Services Logs** > **Integration Runtime**, open Event Viewer, and filter the error logs.
  2. Check to see whether an error log contains the following error:

     ```
     System.ServiceModel.EndpointNotFoundException: Could not connect to
     net.tcp://xxxxxxx.bwld.com:8060/ExternalService.svc/WorkerManager. The connection attempt
     lasted for a time span of 00:00:00.9940994. TCP error code 10061: No connection could be made
     because the target machine actively refused it 10.2.4.10:8060.
     System.Net.Sockets.SocketException: No connection could be made because the target machine
     actively refused it.
     10.2.4.10:8060
     at System.Net.Sockets.Socket.DoConnect(EndPoint endPointSnapshot, SocketAddress socketAddress)
     at System.Net.Sockets.Socket.Connect(EndPoint remoteEP)
     at System.ServiceModel.Channels.SocketConnectionInitiator.Connect(Uri uri, TimeSpan timeout)
     ```

  3. If you see this error, run the following command in a Command Prompt window:

     ```
     telnet 10.2.4.10 8060
     ```

  4. If you receive the "Could not open connection to the host" command-line error that's shown in the following screenshot, contact your IT department for help to fix this issue. After you can successfully telnet, contact Microsoft Support if you still have issues with the integration runtime node status.



5. Check to see whether the error log contains the following entry:

```
Error log: Cannot connect to worker manager: net.tcp://xxxxxx:8060/ExternalService.svc/ No DNS
entries exist for host azranlcir01r1. No such host is known Exception detail:
System.ServiceModel.EndpointNotFoundException: No DNS entries exist for host xxxxx. --->
System.Net.Sockets.SocketException: No such host is known at System.Net.Dns.GetAddrInfo(String
name) at System.Net.Dns.InternalGetHostByName(String hostName, Boolean includeIPv6) at
System.Net.Dns.GetHostEntry(String hostNameOrAddress) at
System.ServiceModel.Channels.DnsCache.Resolve(Uri uri) --- End of inner exception stack trace
--- Server stack trace: at System.ServiceModel.Channels.DnsCache.Resolve(Uri uri)
```

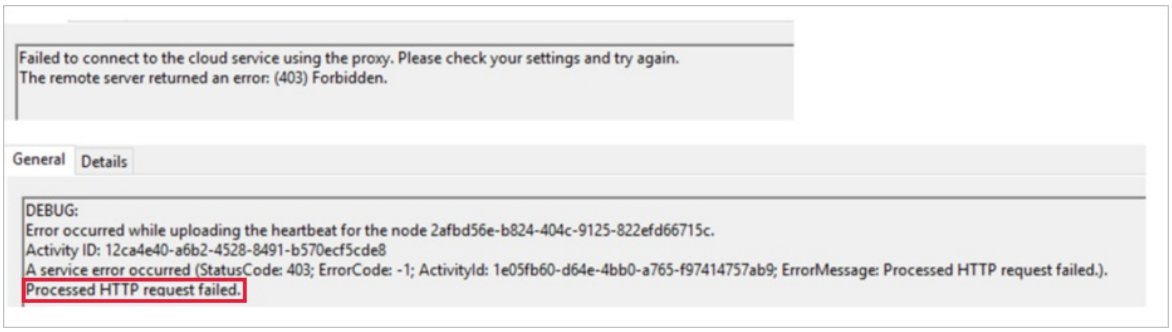
6. To resolve the issue, try one or both of the following methods:
    - Put all the nodes in the same domain.
    - Add the IP to host mapping in all the hosted VM's host files.

Connectivity issue between the self-hosted IR and your data factory or Azure Synapse instance or the self-hosted IR and the data source or sink

To troubleshoot the network connectivity issue, you should know how to collect the network trace, understand how to use it, and analyze the Microsoft Network Monitor (Netmon) trace before applying the Netmon Tools in real cases from the self-hosted IR.

- Symptoms

  You might occasionally need to troubleshoot certain connectivity issues between the self-hosted IR and your data factory or Azure Synapse instance, as shown in the following screenshot, or between the self-hosted IR and the data source or sink.



  In either instance, you might encounter the following errors:

    - "Copy failed with error:Type=Microsoft.DataTransfer.Common.Shared.HybridDeliveryException,Message=Cannot connect to SQL Server: 'IP address'"

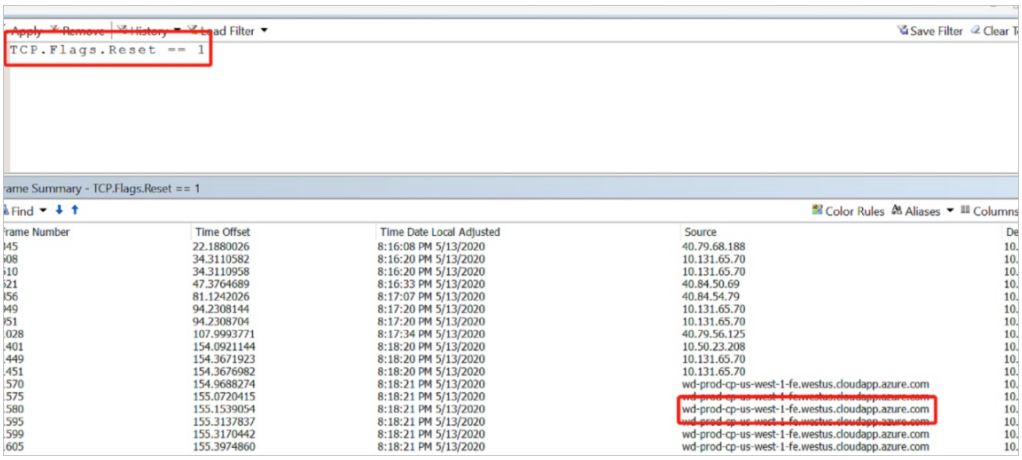
- "One or more errors occurred. An error occurred while sending the request. The underlying connection was closed: An unexpected error occurred on a receive. Unable to read data from the transport connection: An existing connection was forcibly closed by the remote host. An existing connection was forcibly closed by the remote host Activity ID."

- **Resolution**

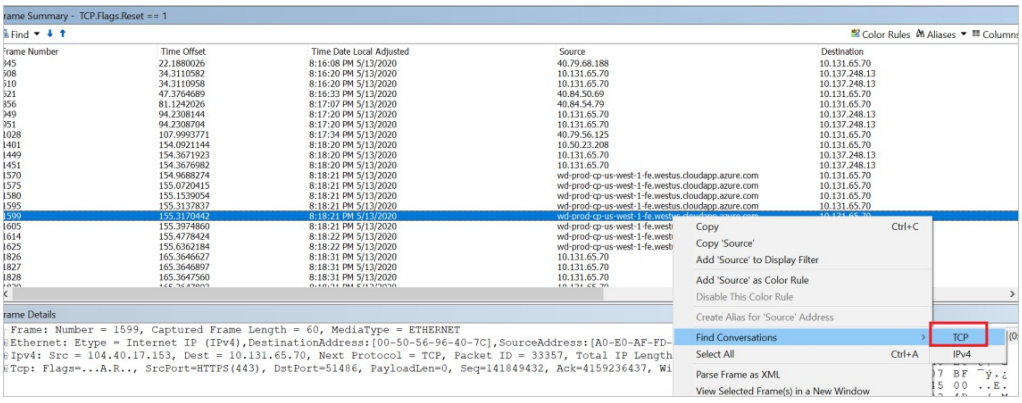
  When you encounter the preceding errors, troubleshoot them by following the instructions in this section.

  - Collect a Netmon trace for analysis:

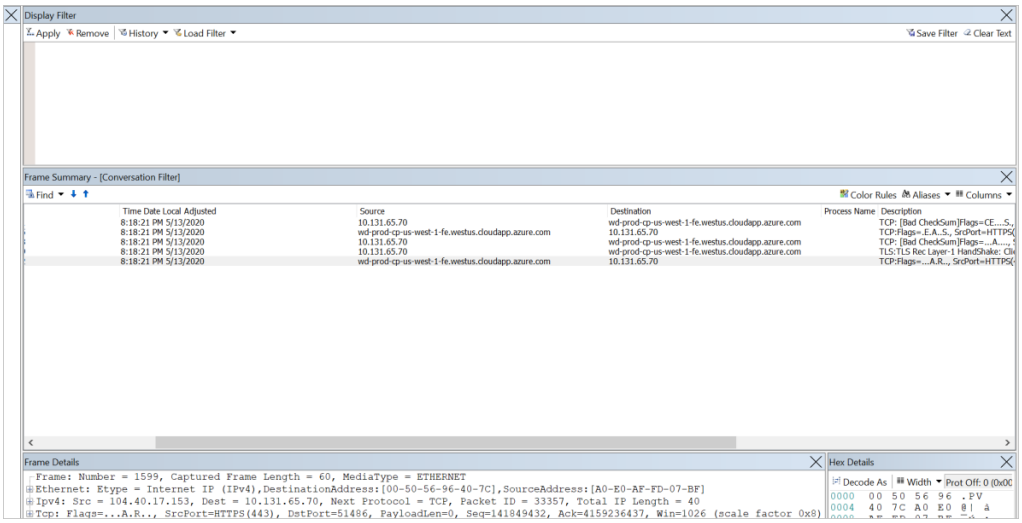
    1. You can set the filter to see a reset from the server to the client side. In the following example screenshot, you can see that the server side is the Data Factory server.



    2. When you get the reset package, you can find the conversation by following Transmission Control Protocol (TCP).



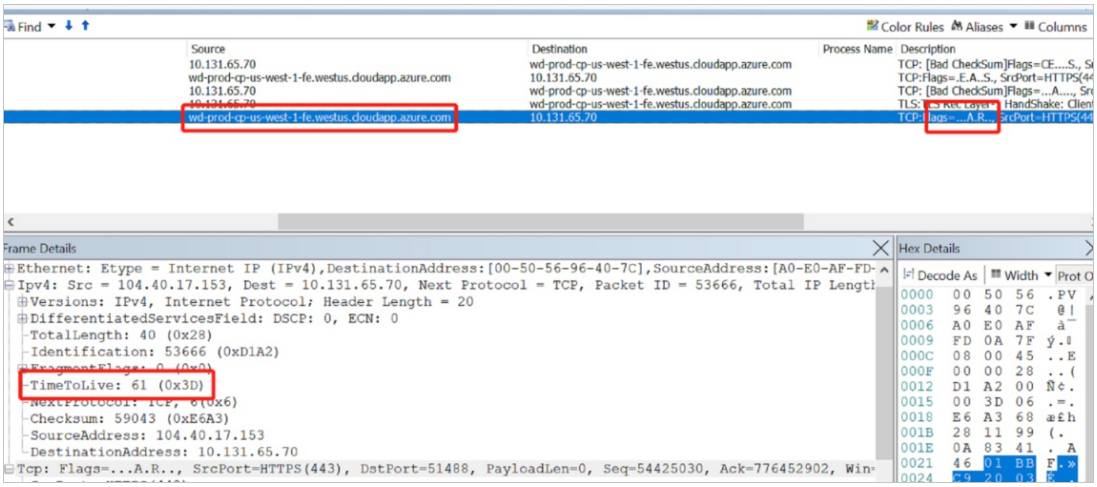
    3. Get the conversation between the client and the Data Factory server below by removing the filter.



- An analysis of the Netmon trace you've collected shows that the Time to Live (TTL)) total is 64. According to the values mentioned in the IP Time to Live (TTL) and Hop Limit Basics article, extracted in the following list, you can see that it's the Linux System that resets the package and causes the disconnection.

  Default TTL and Hop Limit values vary between different operating systems, as listed here:

  - Linux kernel 2.4 (circa 2001): 255 for TCP, User Datagram Protocol (UDP), and Internet Control Message Protocol (ICMP)
  - Linux kernel 4.10 (2015): 64 for TCP, UDP, and ICMP
  - Windows XP (2001): 128 for TCP, UDP, and ICMP
  - Windows 10 (2015): 128 for TCP, UDP, and ICMP
  - Windows Server 2008: 128 for TCP, UDP, and ICMP
  - Windows Server 2019 (2018): 128 for TCP, UDP, and ICMP
  - macOS (2001): 64 for TCP, UDP, and ICMP



  In the preceding example, the TTL is shown as 61 instead of 64, because when the network package reaches its destination, it needs to go through various hops, such as routers or network devices. The number of routers or network devices is deducted to produce the final TTL.

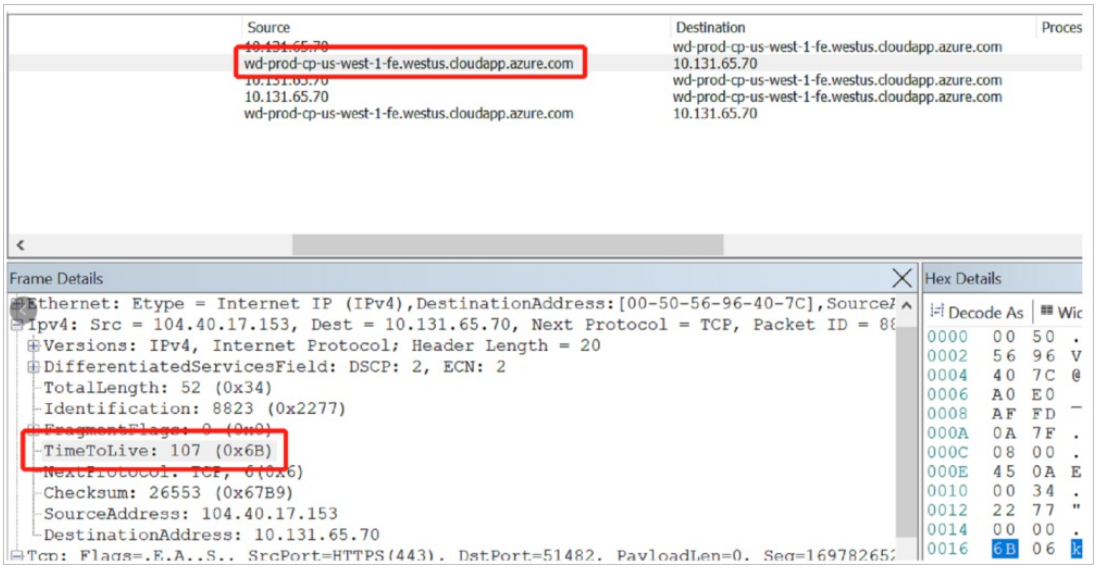
  In this case, you can see that a reset can be sent from the Linux System with TTL 64.

- To confirm where the reset device might come from, check the fourth hop from self-hosted IR.

  Network package from Linux System A with TTL 64 -> B TTL 64 minus 1 = 63 -> C TTL 63 minus 1 = 62 -> TTL 62 minus 1 = 61 self-hosted IR

- In an ideal situation, the TTL hops number would be 128, which means that the Windows

operating system is running your data factory instance. As shown in the following example, *128 minus 107 = 21 hops*, which means that 21 hops for the package were sent from the data factory instance to the self-hosted IR during the TCP 3 handshake.



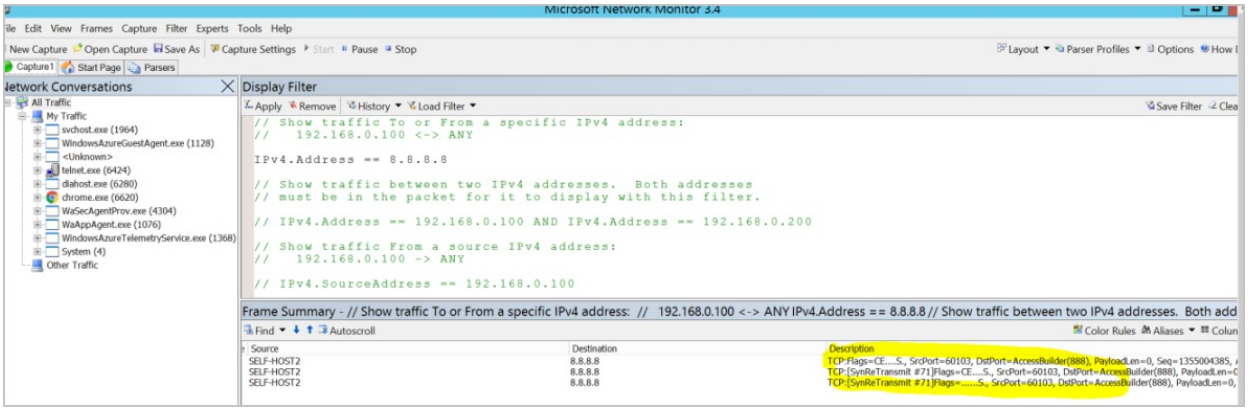
Therefore, you need to engage the network team to check to see what the fourth hop is from the self-hosted IR. If it's the firewall, as with the Linux System, check any logs to see why that device resets the package after the TCP 3 handshake.

If you're unsure where to investigate, try to get the Netmon trace from both the self-hosted IR and the firewall during the problematic time. This approach will help you figure out which device might have reset the package and caused the disconnection. In this case, you also need to engage your network team to move forward.

Analyze the Netmon trace

> **NOTE**
>
> The following instructions apply to the Netmon trace. Because Netmon trace is currently out of support, you can use Wireshark for this purpose.

When you try to telnet **8.8.8.8 888** with the collected Netmon trace, you should see the trace in the following screenshots:

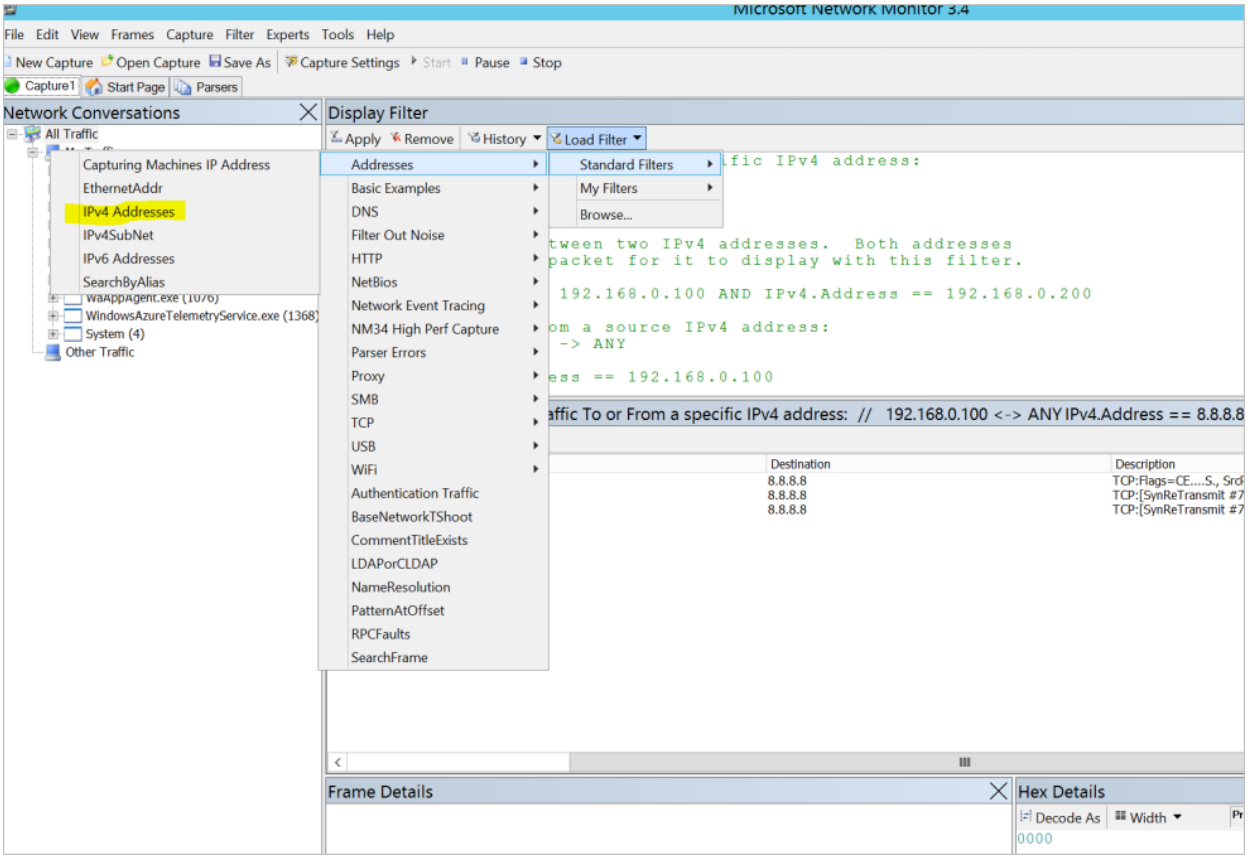


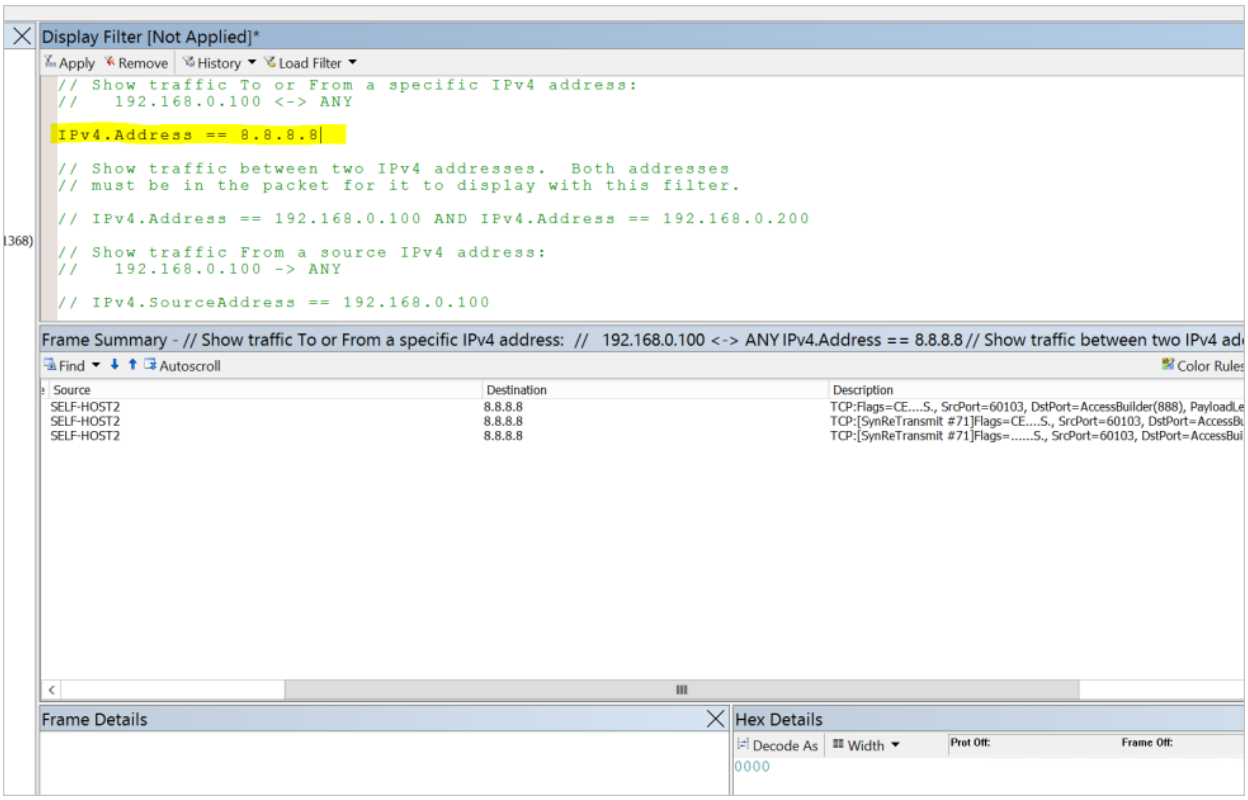
The preceding images show that you couldn't make a TCP connection to the **8.8.8.8** server side on port **888**, so you see two **SynReTransmit** additional packages there. Because source **SELF-HOST2** couldn't connect to **8.8.8.8** with the first package, it will keep trying to make the connection.

TIP

To make this connection, try the following solution:

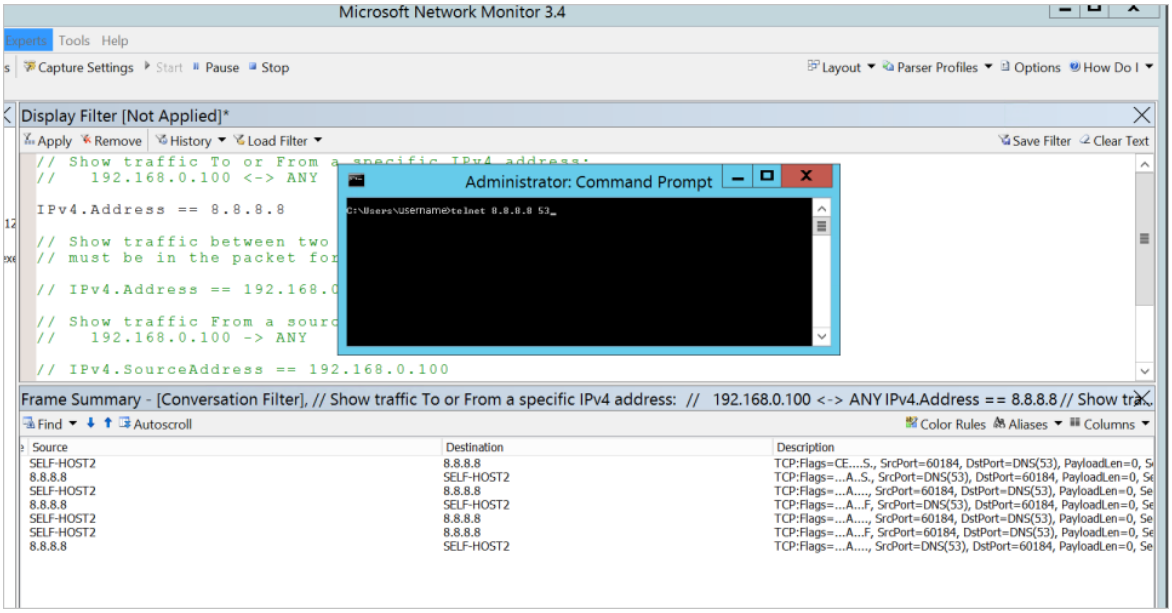
1. Select **Load Filter** > **Standard Filter** > **Addresses** > **IPv4 Addresses**.
2. To apply the filter, enter **IPv4.Address == 8.8.8.8**, and then select **Apply**. You should then see the communication from the local machine to destination **8.8.8.8**.





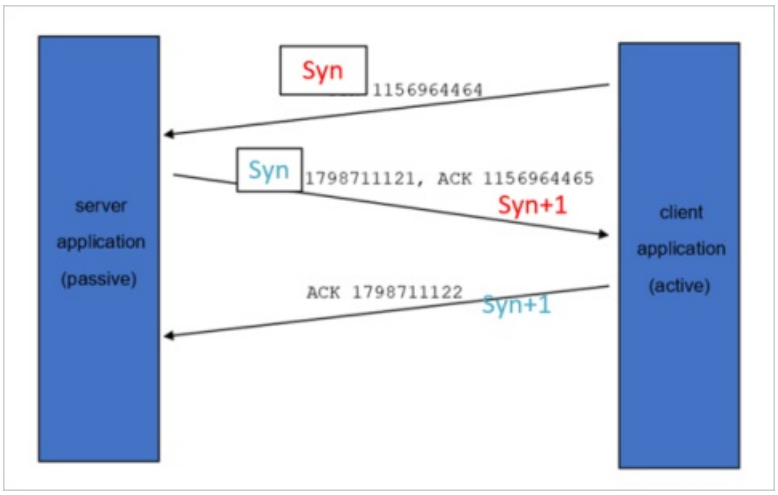
Successful scenarios are shown in the following examples:

- If you can telnet `8.8.8.8 53` without any issues, there's a successful TCP 3 handshake, and the session finishes with a TCP 4 handshake.



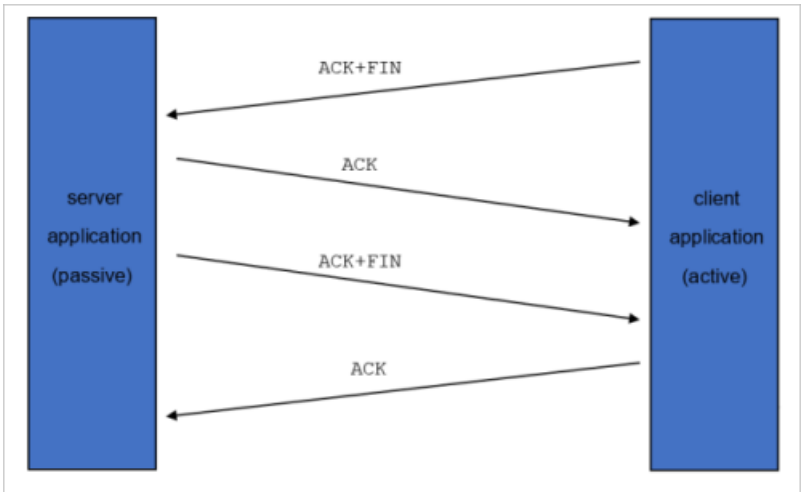

```
TCP:Flags=CE....S., SrcPort=60184, DstPort=DNS(53), PayloadLen=0, Seq=1156964464, Ack=0, Win=8192 (
TCP:Flags=...A..S., SrcPort=DNS(53), DstPort=60184, PayloadLen=0, Seq=1798711121, Ack=1156964465, W
TCP:Flags=...A...., SrcPort=60184, DstPort=DNS(53), PayloadLen=0, Seq=1156964465, Ack=1798711122, Wi
```

- The preceding TCP 3 handshake produces the following workflow:



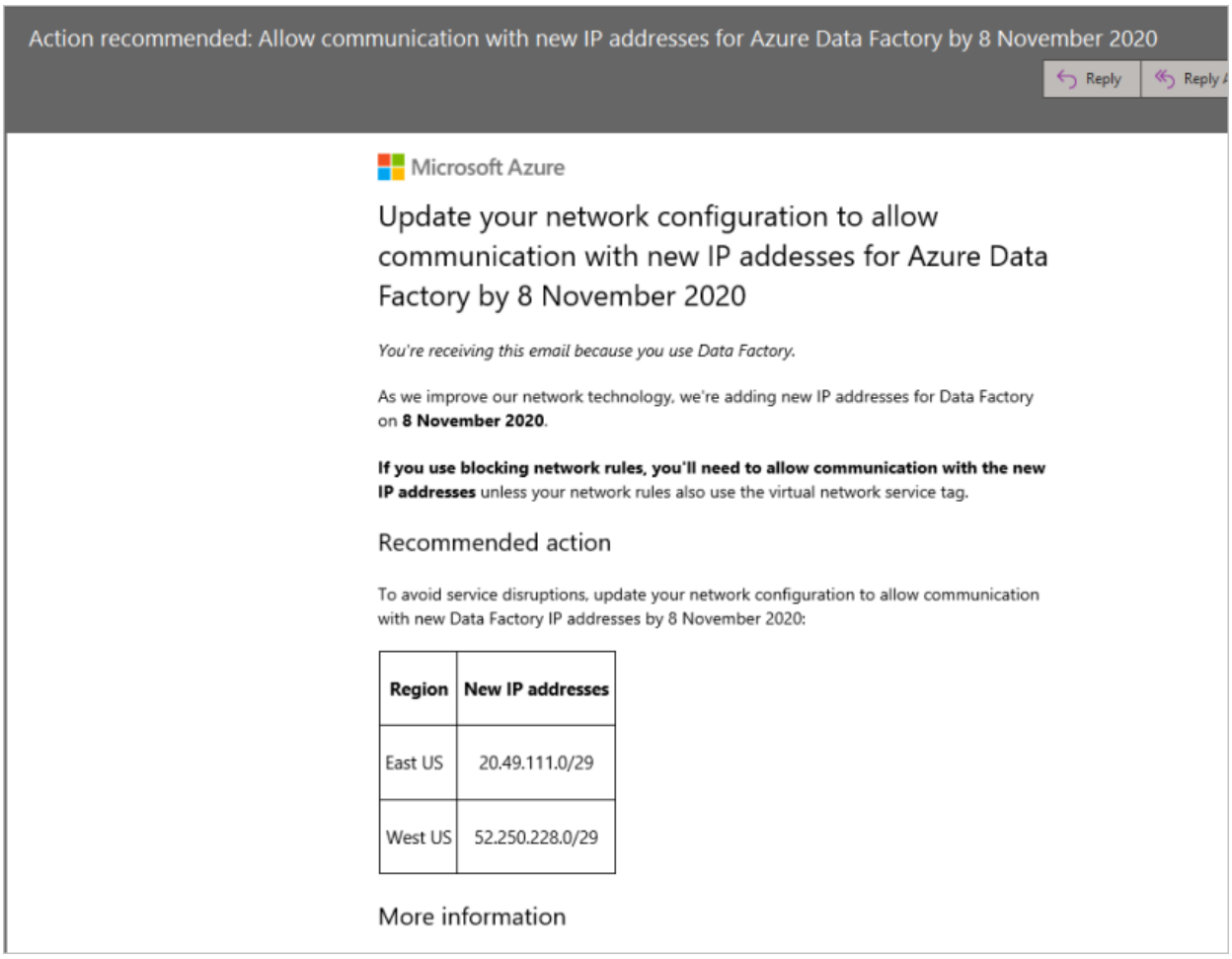
- The TCP 4 handshake to finish the session is illustrated by the following workflows:

```
TCP:Flags=...A...F, SrcPort=DNS(53), DstPort=60184, PayloadLen=0, Seq=1798711122, Ack=1156964465, Win
TCP:Flags=...A...., SrcPort=60184, DstPort=DNS(53), PayloadLen=0, Seq=1156964465, Ack=1798711123, Win
TCP:Flags=...A...F, SrcPort=60184, DstPort=DNS(53), PayloadLen=0, Seq=1156964465, Ack=1798711123, Win
TCP:Flags=...A...., SrcPort=DNS(53), DstPort=60184, PayloadLen=0, Seq=1798711123, Ack=1156964466, Win
```



Microsoft email notification about updating your network configuration

You might receive the following email notification, which recommends that you update your network configuration to allow communication with new IP addresses for Azure Data Factory by 8 November 2020:



Action recommended: Allow communication with new IP addresses for Azure Data Factory by 8 November 2020

Reply

Microsoft Azure

Update your network configuration to allow communication with new IP addesses for Azure Data Factory by 8 November 2020

You're receiving this email because you use Data Factory.

As we improve our network technology, we're adding new IP addresses for Data Factory on **8 November 2020**.

If you use blocking network rules, you'll need to allow communication with the new IP addresses unless your network rules also use the virtual network service tag.

Recommended action

To avoid service disruptions, update your network configuration to allow communication with new Data Factory IP addresses by 8 November 2020:

Region	New IP addresses
East US	20.49.111.0/29
West US	52.250.228.0/29

More information

Determine whether this notification affects you

This notification applies to the following scenarios:

Scenario 1: Outbound communication from a self-hosted integration runtime that's running on-premises behind a corporate firewall

How to determine whether you're affected:

- You *are not* affected if you're defining firewall rules based on fully qualified domain names (FQDNs) that use the approach described in Set up a firewall configuration and allowlist for IP addresses.
- You *are* affected if you're explicitly enabling the allowlist for outbound IPs on your corporate firewall.

  If you're affected, take the following action: by November 8, 2020, notify your network infrastructure team to update your network configuration to use the latest data factory IP addresses. To download the

latest IP addresses, go to Discover service tags by using downloadable JSON files.

Scenario 2: Outbound communication from a self-hosted integration runtime that's running on an Azure VM inside a customer-managed Azure virtual network

How to determine whether you're affected:

- Check to see whether you have any outbound network security group (NSG) rules in a private network that contains self-hosted integration runtime. If there are no outbound restrictions, you aren't affected.
- If you have outbound rule restrictions, check to see whether you're using service tags. If you're using service tags, you're not affected. There's no need to change or add anything, because the new IP range is under your existing service tags.

Port	Protocol	Source	Destination	Action
443	Any	VirtualNetwork	DataFactory	Allow

- You *are* affected if you're explicitly enabling the allowlist for outbound IP addresses on your NSG rules setting on the Azure virtual network.

  If you're affected, take the following action: by November 8, 2020, notify your network infrastructure team to update the NSG rules on your Azure virtual network configuration to use the latest data factory IP addresses. To download the latest IP addresses, go to Discover service tags by using downloadable JSON files.

Scenario 3: Outbound communication from SSIS Integration Runtime in a customer-managed Azure virtual network

How to determine whether you're affected:

- Check to see whether you have any outbound NSG rules in a private network that contains SQL Server Integration Services (SSIS) Integration Runtime. If there are no outbound restrictions, you aren't affected.
- If you have outbound rule restrictions, check to see whether you're using service tags. If you're using service tags, you're not affected. There's no need to change or add anything because the new IP range is under your existing service tags.
- You *are* affected if you're explicitly enabling the allowlist for outbound IP addresses on your NSG rules setting on the Azure virtual network.

  If you're affected, take the following action: by November 8, 2020, notify your network infrastructure team to update the NSG rules on your Azure virtual network configuration to use the latest data factory IP addresses. To download the latest IP addresses, go to Discover service tags by using downloadable JSON files.

Couldn't establish a trust relationship for the SSL/TLS secure channel

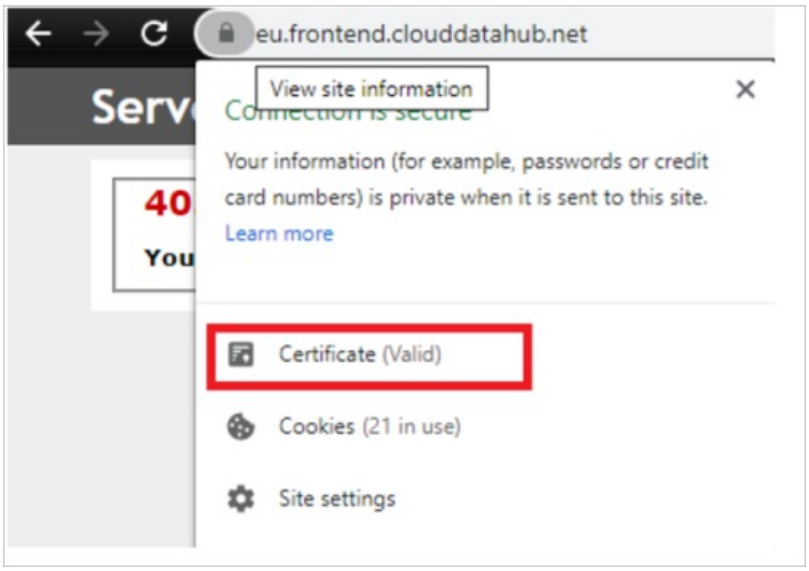
- **Symptoms**

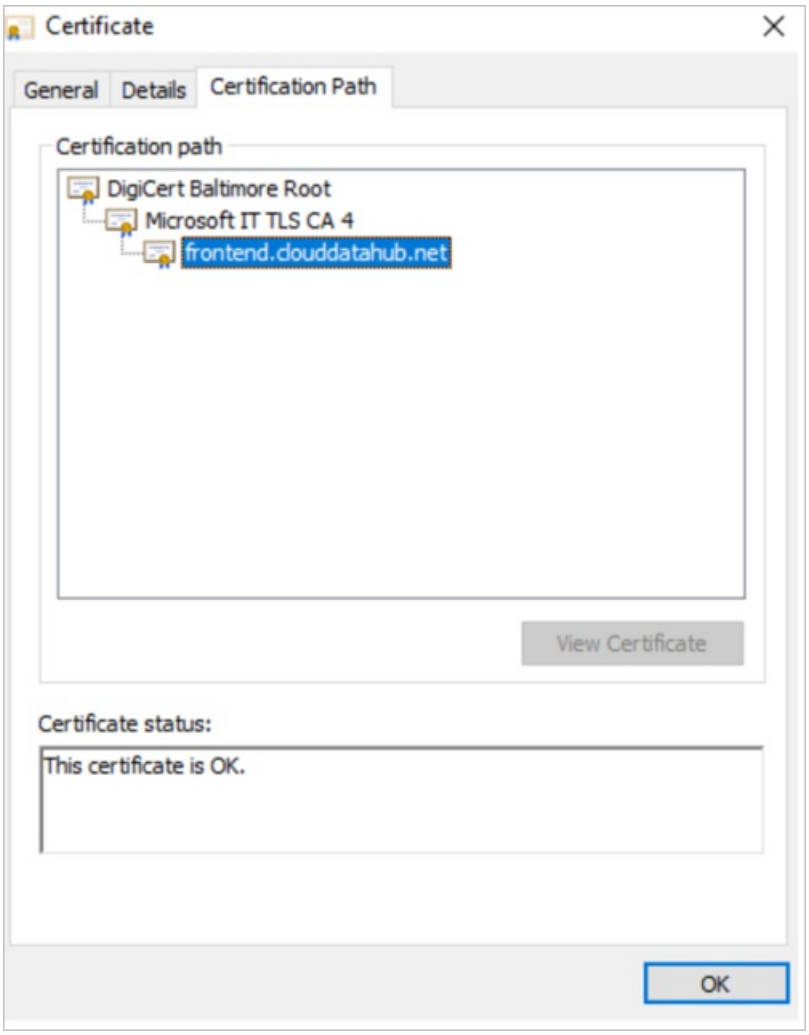
  The self-hosted IR couldn't connect to the Azure Data Factory or Azure Synapse service.

  When you check the self-hosted IR event log or the client notification logs in the CustomLogEvent table, you'll find the following error message:

  "The underlying connection was closed: Could not establish trust relationship for the SSL/TLS secure channel. The remote certificate is invalid according to the validation procedure."

  The simplest way to check the server certificate of the service is to open the service URL in your browser. For example, open the check server certificate link on the machine where the self-hosted IR is installed, and then view the server certificate information.





- **Cause**

  There are two possible reasons for this issue:

  - Reason 1: The root CA of the service's server certificate isn't trusted on the machine where the self-hosted IR is installed.
  - Reason 2: You're using a proxy in your environment, the server certificate of the service is replaced by the proxy, and the replaced server certificate isn't trusted by the machine where the self-hosted IR is installed.

- **Resolution**

  - For reason 1: Make sure that the service's server certificate and its certificate chain are trusted by the

machine where the self-hosted IR is installed.

- For reason 2: Either trust the replaced root CA on the self-hosted IR machine, or configure the proxy not to replace the service's server certificate.

For more information about trusting certificates on Windows, see Installing the trusted root certificate.

- **Additional information**
  We've rolled out a new SSL certificate, which is signed from DigiCert. Check to see whether the DigiCert Global Root G2 is in the trusted root CA.

If it isn't in the trusted root CA, download it here.

Next steps

For more help with troubleshooting, try the following resources:

- Data Factory blog
- Data Factory feature requests
- Azure videos
- Microsoft Q&A page
- Stack overflow forum for Data Factory
- Twitter information about Data Factory
- Mapping data flows performance guide

Machine Learning capabilities in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

Azure Synapse Analytics offers various machine learning capabilities. This article provides an overview of how you can apply Machine Learning in the context of Azure Synapse.

This overview covers the different capabilities in Synapse related to machine learning, from a data science process perspective.

You may be familiar with how a typical data science process looks. It's a well-known process, which most machine learning projects follow.

At a high level, the process contains the following steps:

- Business understanding (not discussed in this article)
- Data acquisition and understanding
- Modeling
- Model deployment and scoring

This article describes the Azure Synapse machine learning capabilities in different analytics engines, from a data science process perspective. For each step in the data science process, the Azure Synapse capabilities that can help are summarized.

Data acquisition and understanding

Most machine learning projects involve well-established steps, and one of these steps is to access and understand the data.

Data source and pipelines

Thanks to Azure Data Factory, a natively integrated part of Azure Synapse, there is a powerful set of tools available for data ingestion and data orchestration pipelines. This allows you to easily build data pipelines to access and transform the data into a format that can be consumed for machine learning. Learn more about data pipelines in Synapse.

Data preparation and exploration/visualization

An important part of the machine learning process is to understand the data by exploration and visualizations.

Depending on where the data is stored, Synapse offers a set of different tools to explore and prepare it for analytics and machine learning. One of the quickest ways to get started with data exploration is using Apache Spark or serverless SQL pools directly over data in the data lake.

- Apache Spark for Azure Synapse offers capabilities to transform, prepare, and explore your data at scale. These spark pools offer tools like PySpark/Python, Scala, and .NET for data processing at scale. Using powerful visualization libraries, the data exploration experience can be enhanced to help understand the data better. Learn more about how to explore and visualize data in Synapse using Spark.
- Serverless SQL pools offer a way to explore data using TSQL directly over the data lake. Serverless SQL pools also offer some built-in visualizations in Synapse Studio. Learn more about how to explore data with serverless SQL pools.

Modeling

In Azure Synapse, training machine learning models can be performed on the Apache Spark Pools with tools like PySpark/Python, Scala, or .NET.

Train models on Spark Pools with MLlib

Machine learning models can be trained with help from various algorithms and libraries. Spark MLlib offers scalable machine learning algorithms that can help solving most classical machine learning problems. For a tutorial on how to train a model using MLlib in Synapse, see Build a machine learning app with Apache Spark MLlib and Azure Synapse Analytics.

In addition to MLlib, popular libraries such as Scikit Learn can also be used to develop models. See Manage libraries for Apache Spark in Azure Synapse Analytics for details on how to install libraries on Synapse Spark Pools.

Train models with Azure Machine Learning automated ML

Another way to train machine learning models, that does not require much prior familiarity with machine learning, is to use automated ML. Automated ML is a feature that automatically trains a set of machine learning models and allows the user to select the best model based on specific metrics. Thanks to a seamless integration with Azure Machine Learning from Azure Synapse Notebooks, users can easily leverage automated ML in Synapse with passthrough Azure Active Directory authentication. This means that you only need to point to your Azure Machine Learning workspace and do not need to enter any credentials. The tutorial, Train a model in Python with automated machine learning, describes how to train models using Azure Machine Learning automated ML on Synapse Spark Pools.

Model deployment and scoring

Models that have been trained either in Azure Synapse or outside Azure Synapse can easily be used for batch scoring. Currently in Synapse, there are two ways in which you can run batch scoring.

- You can use the TSQL PREDICT function in Synapse SQL pools to run your predictions right where your data lives. This powerful and scalable function allows you to enrich your data without moving any data out of your data warehouse. A new guided machine learning model experience in Synapse Studio was introduced where you can deploy an ONNX model from the Azure Machine Learning model registry in Synapse SQL Pools for batch scoring using PREDICT.
- Another option for batch scoring machine learning models in Azure Synapse is to leverage the Apache Spark Pools for Azure Synapse. Depending on the libraries used to train the models, you can use a code experience to run your batch scoring.

Next steps

- Get started with Azure Synapse Analytics
- Create a workspace
- Quickstart: Create a new Azure Machine Learning linked service in Synapse
- Tutorial: Machine learning model scoring wizard - dedicated SQL pool

Quickstart: Use a sample notebook from the Synapse Analytics gallery

8/19/2022 • 3 minutes to read • Edit Online

In this quickstart, you'll learn how to copy a sample machine learning notebook from the Synapse Analytics gallery into your workspace, modify it, and run it. The sample notebook ingests an Open Dataset of NYC Taxi trips and uses visualization to help you prepare the data. It then trains a model to predict whether there will be a tip on a given trip.

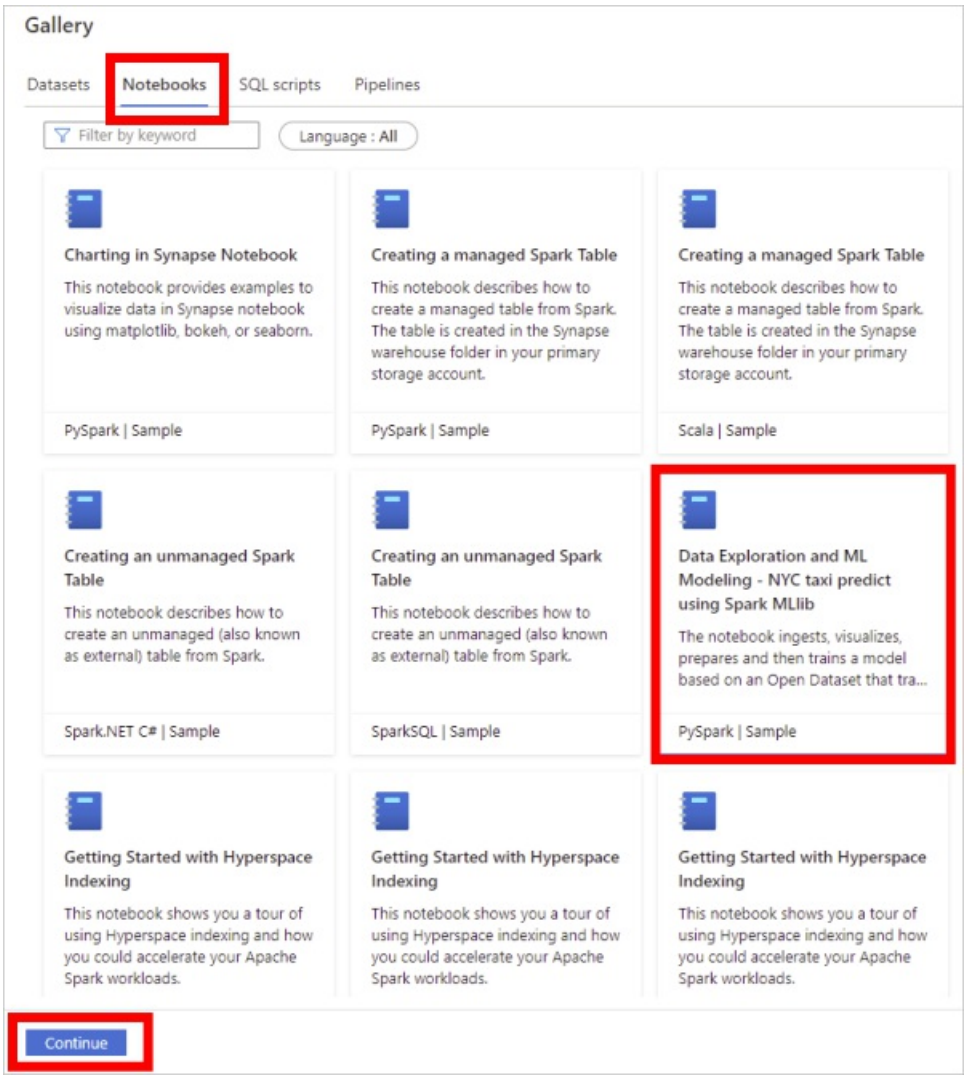
This notebook demonstrates the basic steps used in creating a model: **data import**, **data prep**, **model training**, and **evaluation**. You can use this sample as a starting point for creating a model with your own data.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Spark pool in your Azure Synapse Analytics workspace.

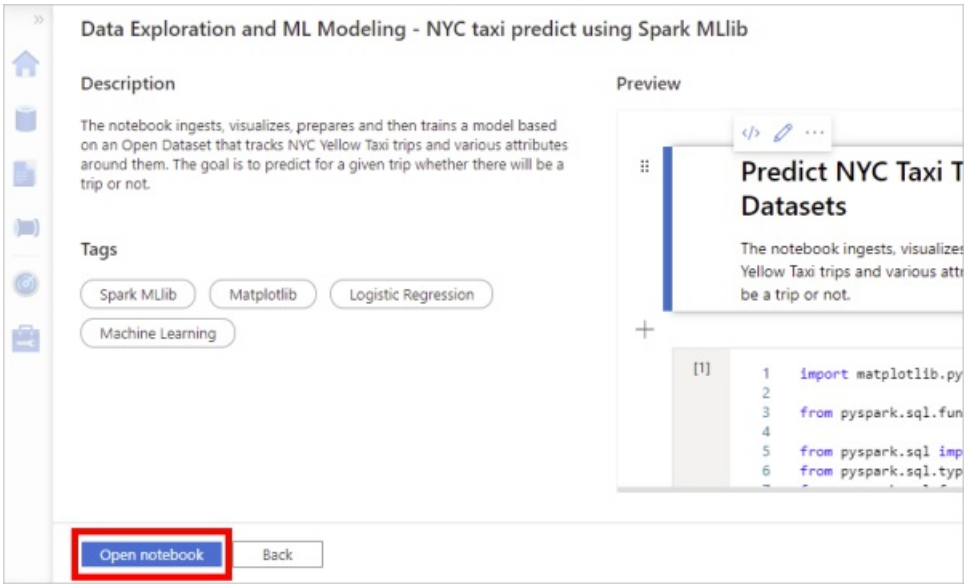
Copy the notebook to your workspace

1. Open your workspace and select **Learn** from the home page.
2. In the **Knowledge center**, select **Browse gallery**.
3. In the gallery, select **Notebooks**.
4. Find and select the notebook "Data Exploration and ML Modeling - NYC taxi predict using Spark MLib".



5. Select **Continue**.

6. On the notebook preview page, select **Open notebook**. The sample notebook is copied into your workspace and opened.



7. In the **Attach to** menu in the open notebook, select your Apache Spark pool.

Run the notebook

The notebook is divided into multiple cells that each perform a specific function. You can manually run each cell,

running cells sequentially, or select **Run all** to run all the cells.

Here are descriptions for each of the cells in the notebook:

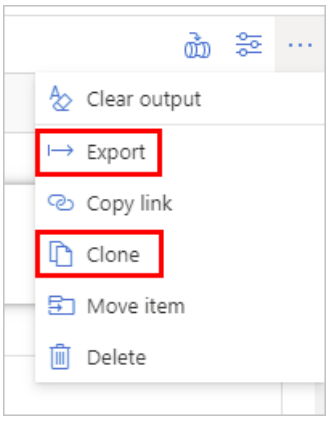
1. Import PySpark functions that the notebook uses.
2. **Ingest Date** - Ingest data from the Azure Open Dataset **NycTlcYellow** into a local dataframe for processing. The code extracts data within a specific time period - you can modify the start and end dates to get different data.
3. Downsample the dataset to make development faster. You can modify this step to change the sample size or the sampling seed.
4. **Exploratory Data Analysis** - Display charts to view the data. This can give you an idea what data prep might be needed before creating the model.
5. **Data Prep and Featurization** - Filter out outlier data discovered through visualization and create some useful derived variables.
6. **Data Prep and Featurization Part 2** - Drop unneeded columns and create some additional features.
7. **Encoding** - Convert string variables to numbers that the Logistic Regression model is expecting.
8. **Generation of Testing and Training Data Sets** - Split the data into separate testing and training data sets. You can modify the fraction and randomizing seed used to split the data.
9. **Train the Model** - Train a Logistic Regression model and display its "Area under ROC" metric to see how well the model is working. This step also saves the trained model in case you want to use it elsewhere.
10. **Evaluate and Visualize** - Plot the model's ROC curve to further evaluate the model.

Save the notebook

To save your notebook by selecting **Publish** on the workspace command bar.

Copying the sample notebook

To make a copy of this notebook, click the ellipsis in the top command bar and select **Clone** to create a copy in your workspace or **Export** to download a copy of the notebook (`.ipynb`) file.



Clean up resources

To ensure the Spark instance is shut down when you're finished, end any connected sessions (notebooks). The pool shuts down when the **idle time** specified in the Apache Spark pool is reached. You can also select **stop session** from the status bar at the upper right of the notebook.

Next steps

- Check out more Synapse sample notebooks in GitHub
- Machine learning with Apache Spark

Industry AI solutions

8/19/2022 • 2 minutes to read • Edit Online

Every industry has common industry specific problems to solve. This article will cover the industry solutions available in Azure Synapse Workspaces for quickly getting started solving common industry problems. At this time, the AI solution offered is a Retail solution for product recommendation. You can find more details below on how to use this solution as a retailer, in order to get started.

Target user

The recommendation solution targets existing data scientists and emerging data scientists who are comfortable with code and are familiar with machine learning concepts. This solution is meant to accelerate the productivity of these users for solving a specific problem in the Retail domain.

Retail product recommendation solution

The **Retail - Product recommendations** solution provides a robust and scalable recommendation engine for out-of-the-box development in Synapse. When using this solution, you will get a Notebook training a machine learning model for product recommendation.

You may need to customize the pre-configured Notebook to ensure it meets your unique business requirements.

This Retail recommendation solution can be deployed in two different modes. You can try it out with sample data, or use a database modeled using the Database Template for Retail in Synapse.

The **Retail - Product recommendations** solution provides a recommendation pipeline for content-based filtering recommendations. The content-based filtering pipeline uses the LightGBM algorithm to train a model for predicting the user preferences based on user and item features. The features can be static features like user profile and item profile, as well as dynamic features like the aggregated user behavior patterns. The content-based filtering typed recommendation system is often used for recommendations such as "personalized recommendation" or "new products you may like".

Get started

1. Open your Synapse workspace
2. From your Home screen, select **Knowledge center** in the **Discover more** section
3. In the Knowledge center, select **Browse gallery**
4. In the Gallery, select the **Database Templates** tab, scroll down to the **AI solutions** section, and select the "Retail - Product recommendations" solution. Click **Continue**.
5. You can choose between two options:
   - "Use sample data"
   - "Use my own data from a workspace database". This option can for example be a database modeled with the Retail Database template.

   Clicking **Deploy** will open a Notebook in your Synapse workspace.
6. A Notebook will now be opened in your workspace. You can attach this Notebook to a Spark pool and start exploring it. Please be aware that this Notebook is meant to be customized to your specific needs.

> **NOTE**
>
> If you choose your own database, you need to customize the Notebook to use your own table and column names.

Next steps

- Get started with Azure Synapse Analytics
- Create a workspace

Quickstart: Create a new Azure Machine Learning linked service in Synapse

8/19/2022 • 4 minutes to read • Edit Online

IMPORTANT, PLEASE NOTE THE BELOW LIMITATIONS:

- **The Azure ML integration is not currently supported in Synapse Workspaces with Data Exfiltration Protection.** If you are **not** using data exfiltration protection and want to connect to Azure ML using private endpoints, you can set up a managed AzureML private endpoint in your Synapse workspace. Read more about managed private endpoints
- **AzureML linked service is not supported with self hosted integration runtimes.** This applies to Synapse workspaces with and without Data Exfiltration Protection.

In this quickstart, you'll link an Azure Synapse Analytics workspace to an Azure Machine Learning workspace. Linking these workspaces allows you to leverage Azure Machine Learning from various experiences in Synapse.

For example, this linking to an Azure Machine Learning workspace enables these experiences:

- Run your Azure Machine Learning pipelines as a step in your Synapse pipelines. To learn more, see Execute Azure Machine Learning pipelines.
- Enrich your data with predictions by bringing a machine learning model from the Azure Machine Learning model registry and score the model in Synapse SQL pools. For more details, see Tutorial: Machine learning model scoring wizard for Synapse SQL pools.

Two types of authentication

There are two types of identities you can use when creating an Azure Machine Learning linked service in Azure Synapse.

- Synapse workspace Managed Identity
- Service Principal

In the following sections, you'll find guidance on how to create an Azure Machine Learning linked service using these two different types of authentication.

Prerequisites

- Azure subscription - Create one for free.
- Synapse Analytics workspace with an ADLS Gen2 storage account configured as the default storage. You need to be the **Storage Blob Data Contributor** of the ADLS Gen2 filesystem that you work with.
- Azure Machine Learning workspace.
- If you chose to use a Service principal, you need permissions (or request from someone who has permissions) to create a service principal and secret which you can use to create the linked service. Note that this service principal needs to be assigned the contributor role in the Azure Machine Learning workspace.
- Sign in to the Azure portal

Create a linked service using the Synapse workspace Managed identity

This section will guide you on how to create an Azure Machine Learning linked service in Azure Synapse, using the Azure Synapse workspace Managed Identity

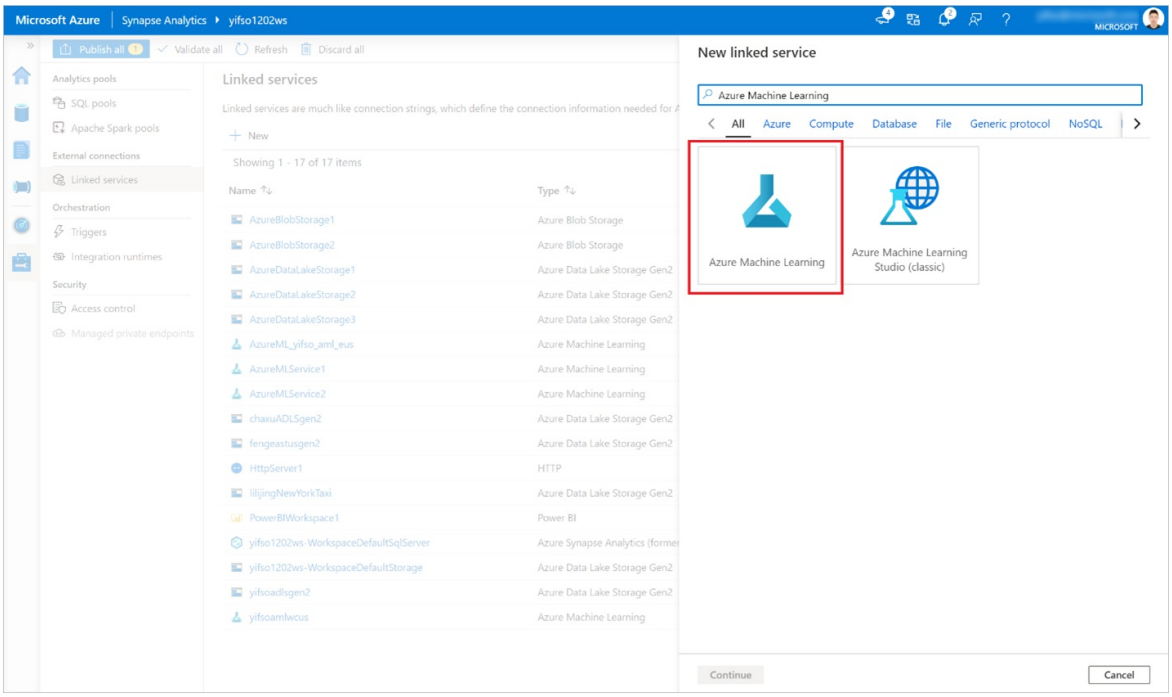
Give MSI permission to the Azure ML workspace

1. Navigate to your Azure Machine Learning workspace resource in the Azure portal and select **Access Control**

2. Create a role assignment and add your Synapse workspace Managed Service identity (MSI) as a *contributor* of the Azure Machine Learning workspace. Note that this will require being an owner of the resource group that the Azure Machine Learning workspace belongs to. If you have trouble finding your Synapse workspace MSI, search for the name of the Synapse workspace.

Create an Azure ML linked service

1. In the Synapse workspace where you want to create the new Azure Machine Learning linked service, go to **Management** > **Linked service**, and create a new linked service with type "Azure Machine Learning".



2. Fill out the form:

   - Provide the details about the Azure Machine Learning workspace you want to link to. This includes details about subscription and workspace name.

   - Select Authentication Method: **Managed Identity**

3. Click **Test Connection** to verify if the configuration is correct. If the connection test passes, click **Save**.

   If the connection test fails, make sure that the Azure Synapse workspace MSI has permissions to access this Azure Machine Learning workspace, and try again.

Create a linked service using a service principal

This section will guide you on how to create an Azure Machine Learning linked service with a service principal.

Create a service principal

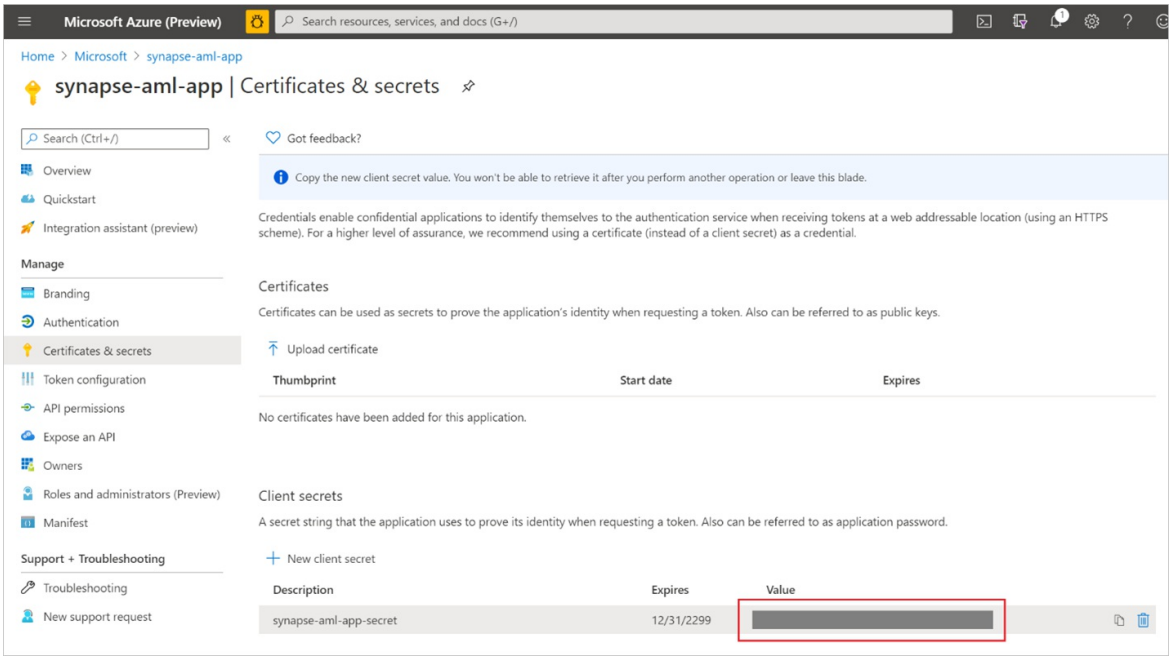
This step will create a new Service Principal. If you want to use an existing Service Principal, you can skip this step.

1. Open Azure portal.

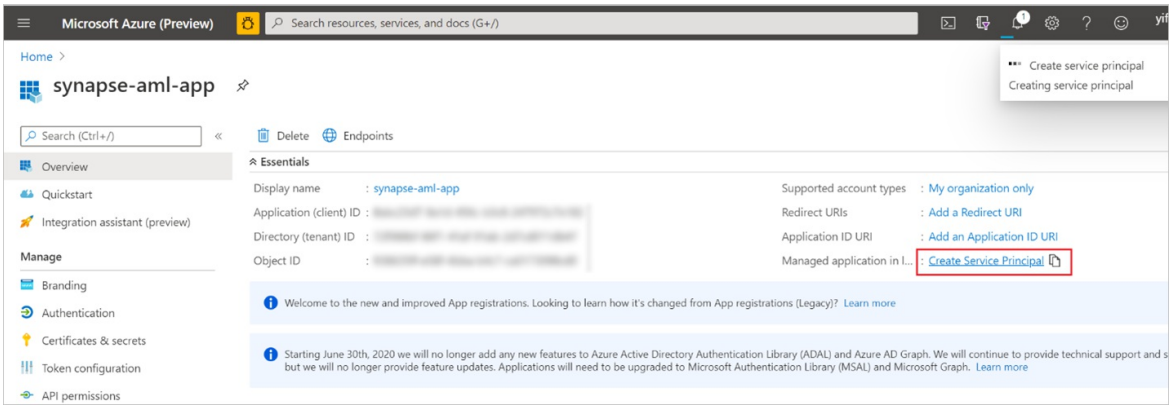
2. Go to **Azure Active Directory** -> **App registrations**.

3. Click **New registration**. Then, follow instructions to register a new application.

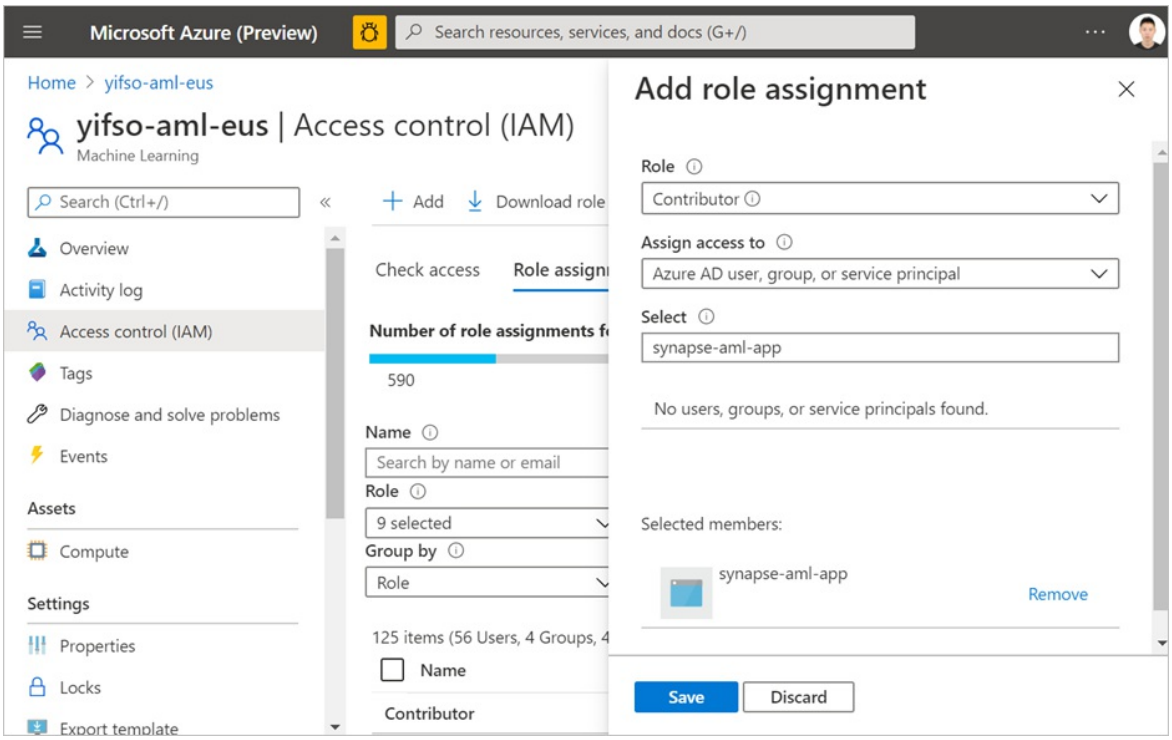
4. After the application is registered, generate a secret for the application. Go to **Your application** -> **Certificate & Secret**. Click **Add client secret** to generate a secret. Keep the secret safe and it will be used later.



5. Create a service principal for the application. Go to **Your application** -> **Overview** and then click **Create service principal**. In some cases, this service principal is automatically created.

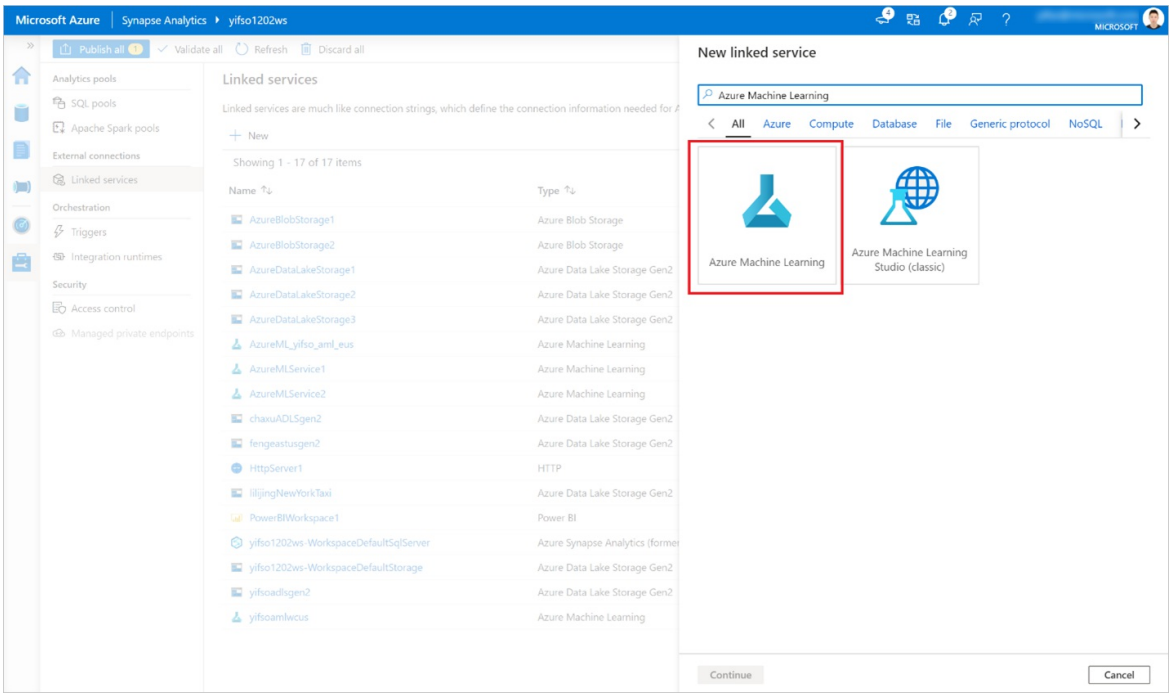


6. Add the service principal as "contributor" of the Azure Machine Learning workspace. Note that this will require being an owner of the resource group that the Azure Machine Learning workspace belongs to.



Create an Azure ML linked service

1. In the Synapse workspace where you want to create the new Azure Machine Learning linked service, go to **Management** -> **Linked service**, create a new linked service with type "Azure Machine Learning".



2. Fill out the form:

   - Provide the details about the Azure Machine Learning workspace you want to link to. This includes details about subscription and workspace name.
   - Select Authentication Method: **Service Principal**
   - Service principal ID: This is the **application (client) ID** of the Application.

> **NOTE**
>
> The ID is NOT the name of the application. You can find this ID in the overview page of the application. It should be a long string looking similar to this "81707eac-ab38-406u-8f6c-10ce76a568d5".

- Service principal key: The secret you generated in the previous section.

3. Click **Test Connection** to verify if the configuration is correct. If the connection test passes, click **Save**.

   If the connection test fails, make sure that the service principal ID and secret are correct and try again.

Next steps

- Tutorial: Machine learning model scoring wizard - dedicated SQL pool
- Machine Learning capabilities in Azure Synapse Analytics

Quickstart: Configure prerequisites for using Cognitive Services in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

In this quickstart, you'll learn how set up the prerequisites for securely using Azure Cognitive Services in Azure Synapse Analytics. Linking these Azure Cognitive Services allows you to leverage Azure Cognitive Services from various experiences in Synapse.

This quickstart covers:

- Create a Cognitive Services resource like Text Analytics or Anomaly Detector.
- Store an authentication key to Cognitive Services resources as secrets in Azure Key Vault, and configure access for an Azure Synapse Analytics workspace.
- Create an Azure Key Vault linked service in your Azure Synapse Analytics workspace.
- Create an Azure Cognitive Services linked service in your Azure Synapse Analytics workspace.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Azure Data Lake Storage Gen2 file system that you work with.

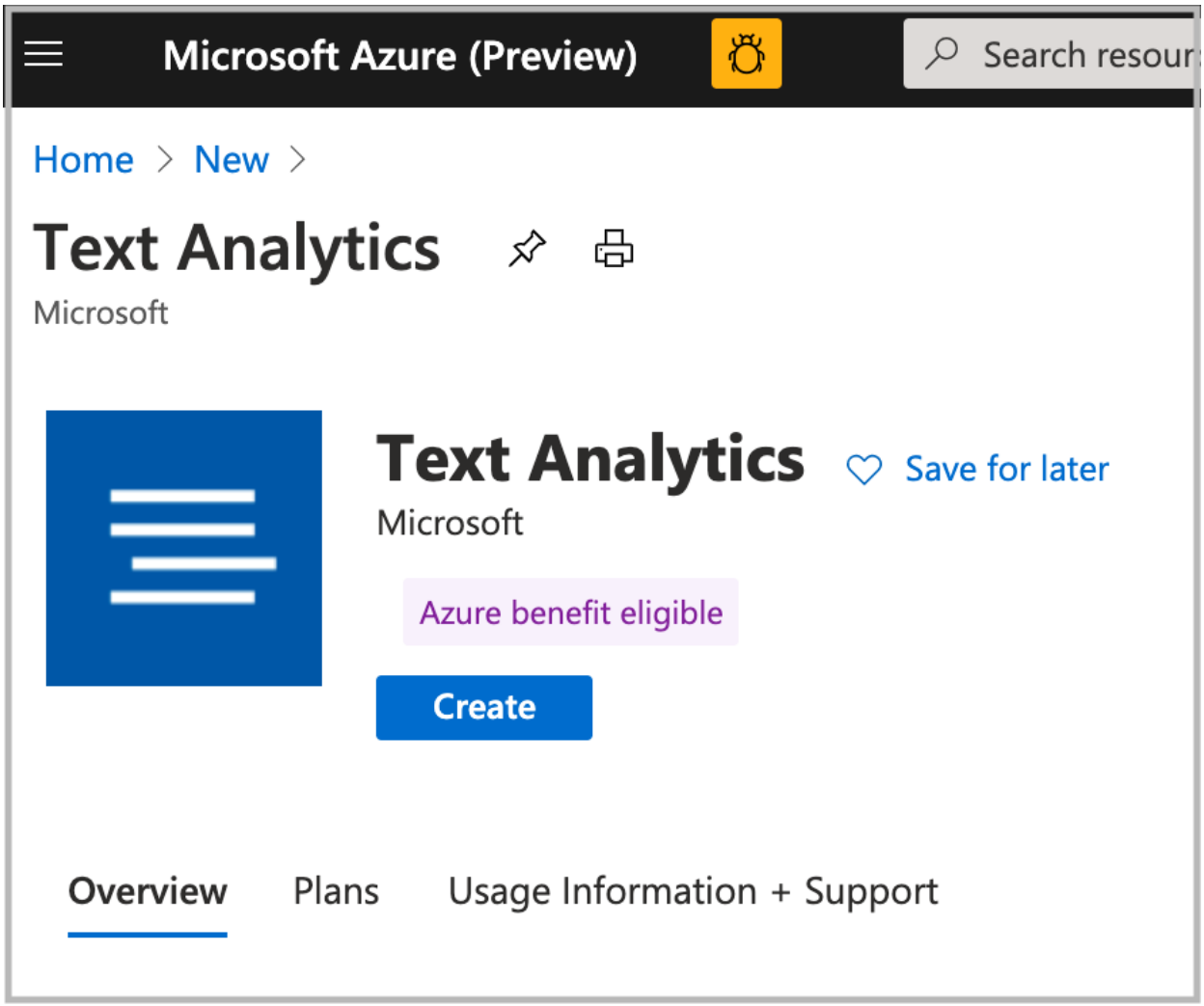
Sign in to the Azure portal

Sign in to the Azure portal.

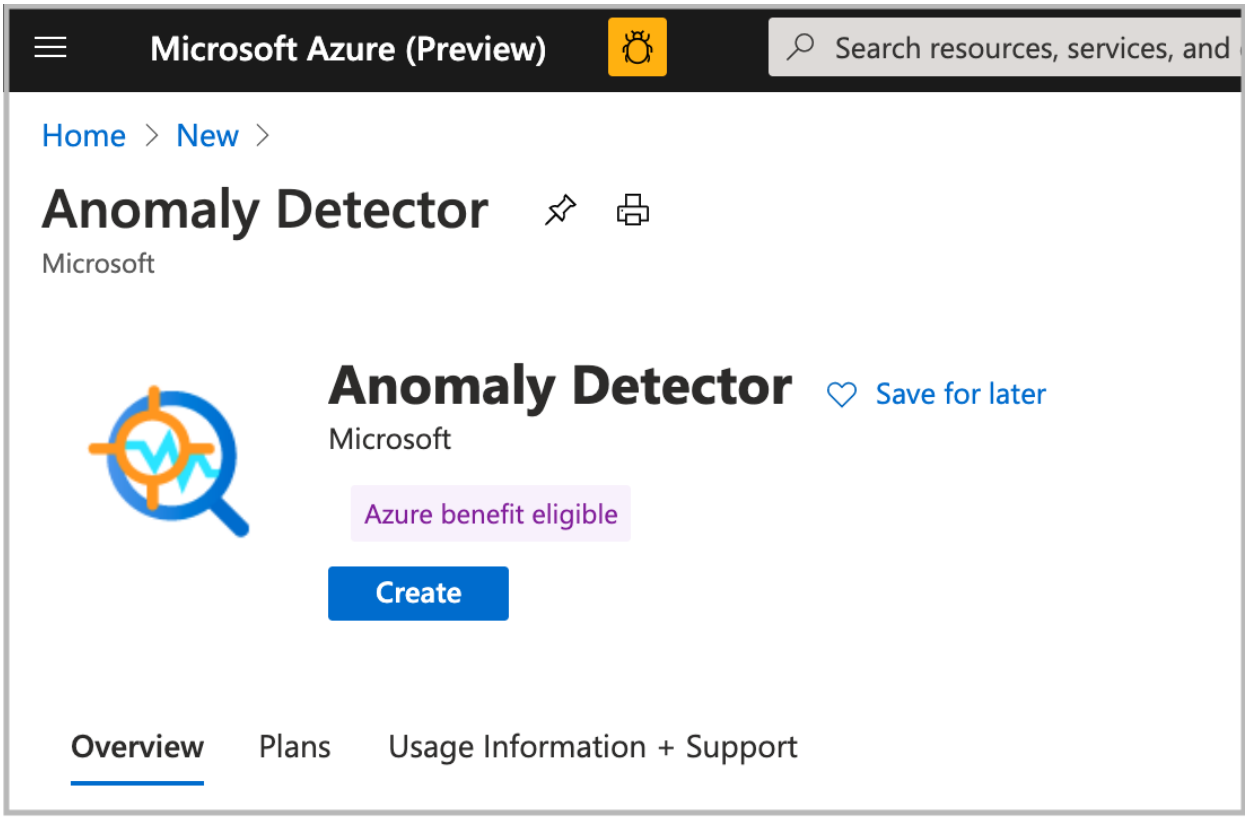
Create a Cognitive Services resource

Azure Cognitive Services includes many types of services. Follow services are examples used in the Azure Synapse tutorials.

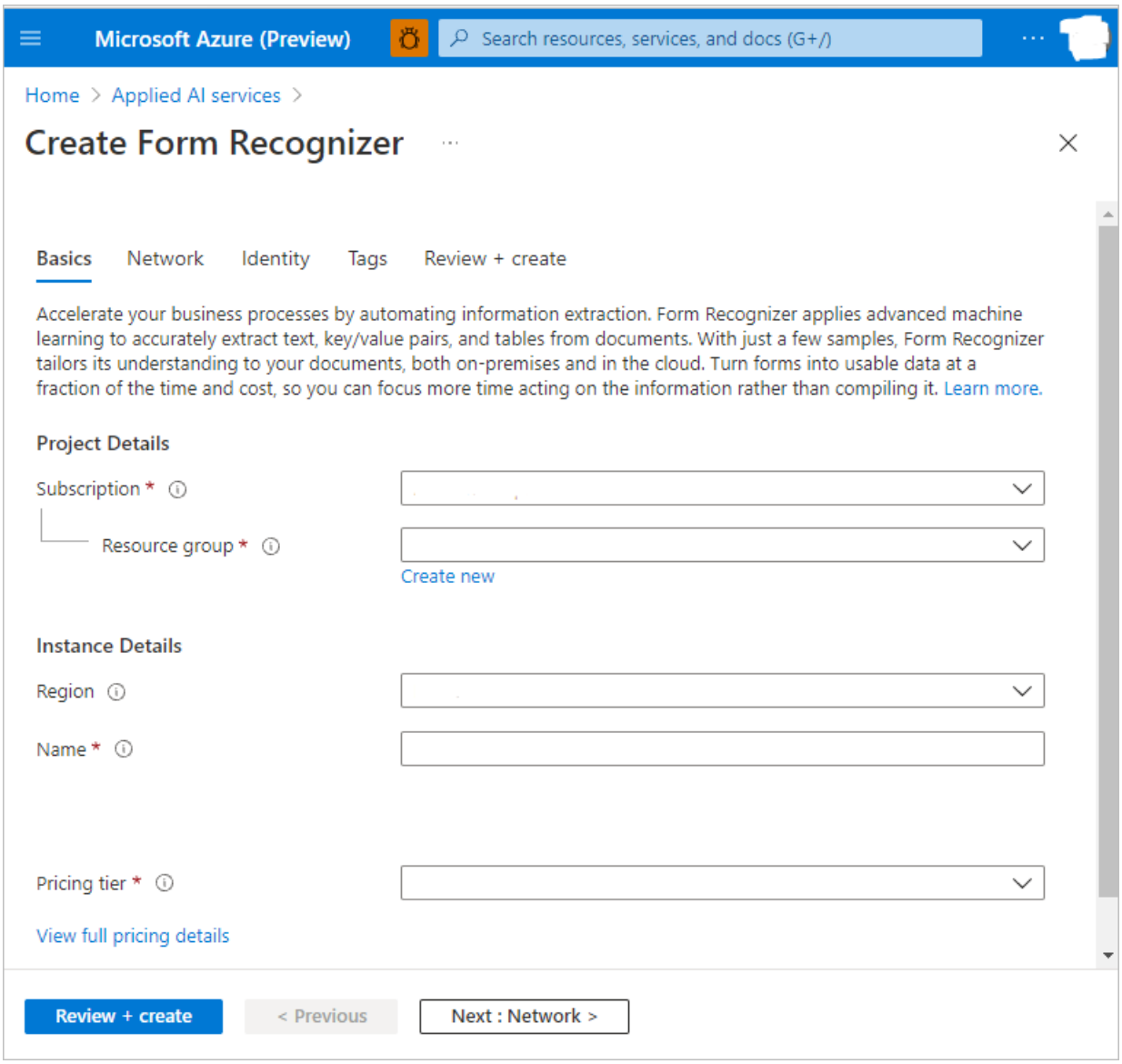
You can create a Text Analytics resource in the Azure portal:



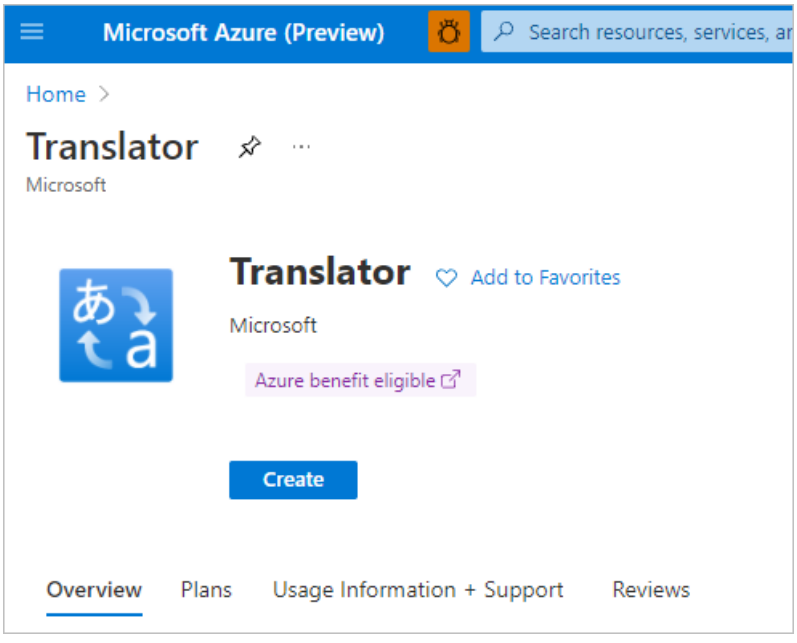
You can create an Anomaly Detector resource in the Azure portal:



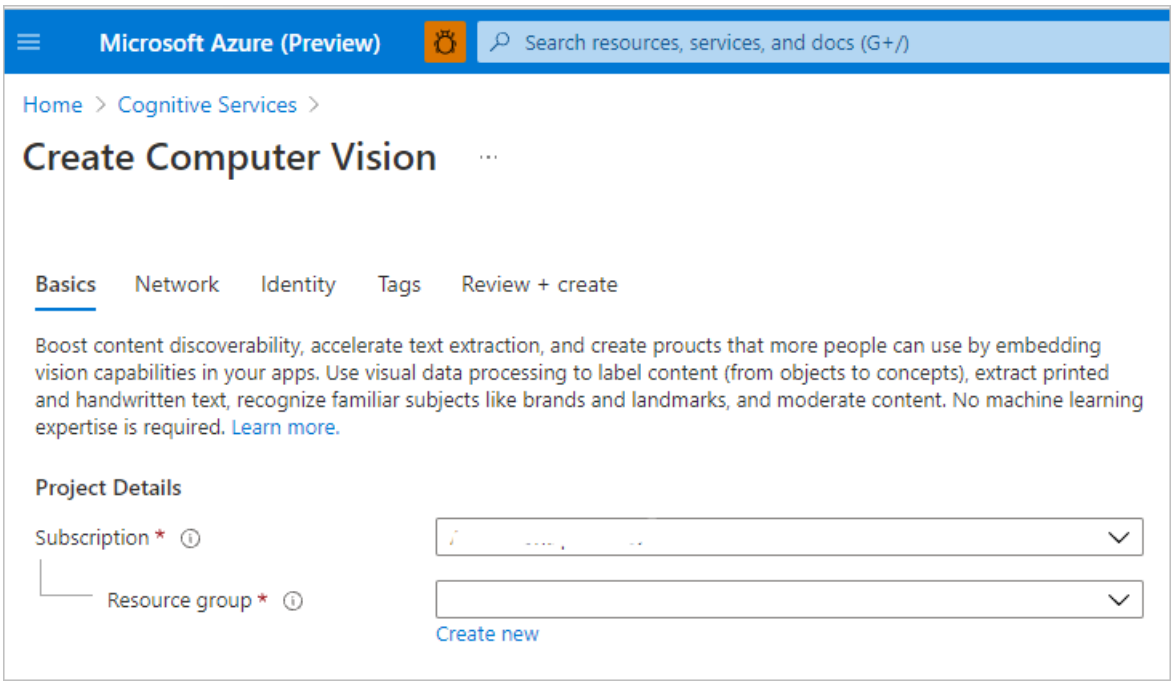
You can create an Form Recognizer resource in the Azure portal:



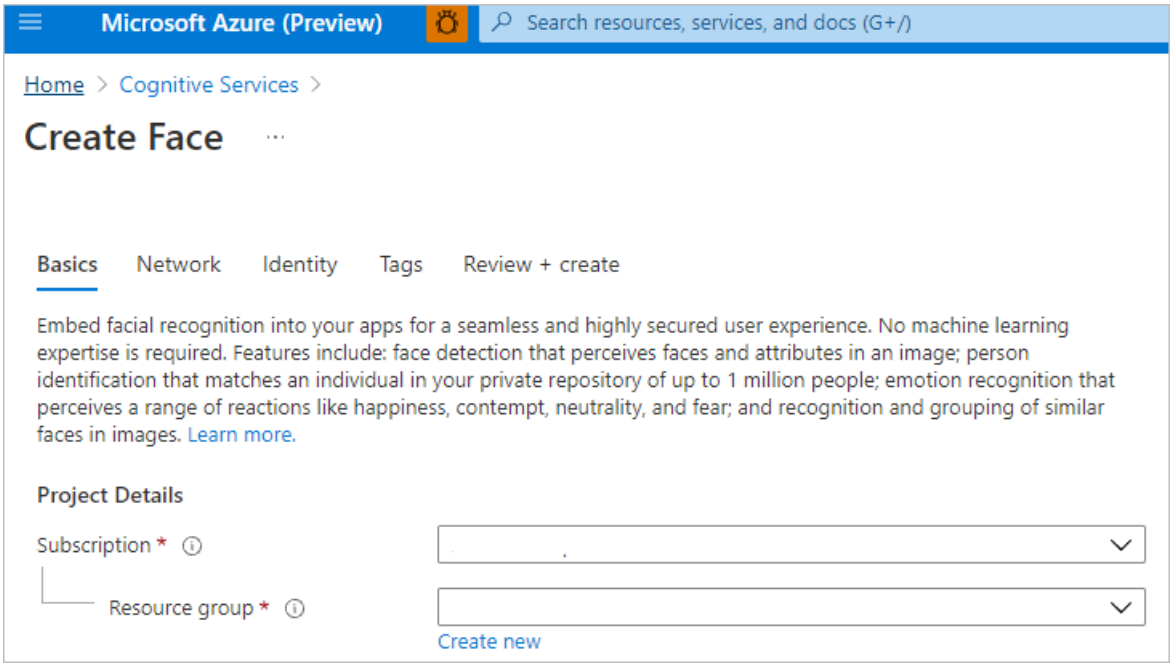
You can create a Translator resource in the Azure portal:



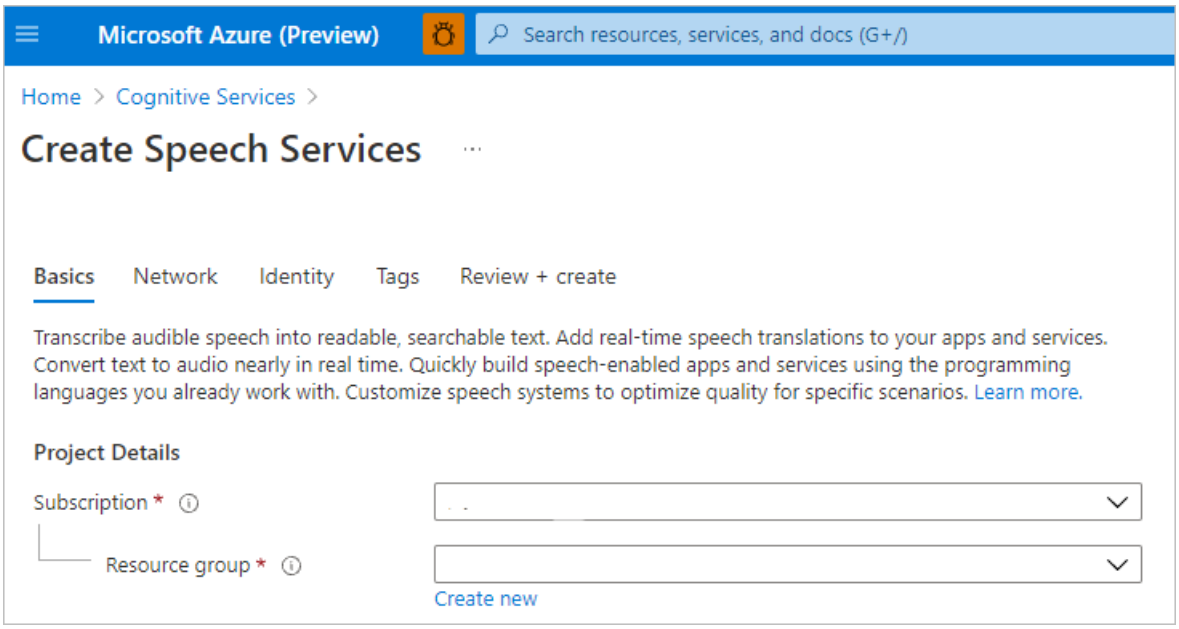
You can create a Computer Vision resource in the Azure portal:



You can create a Face resource in the Azure portal:



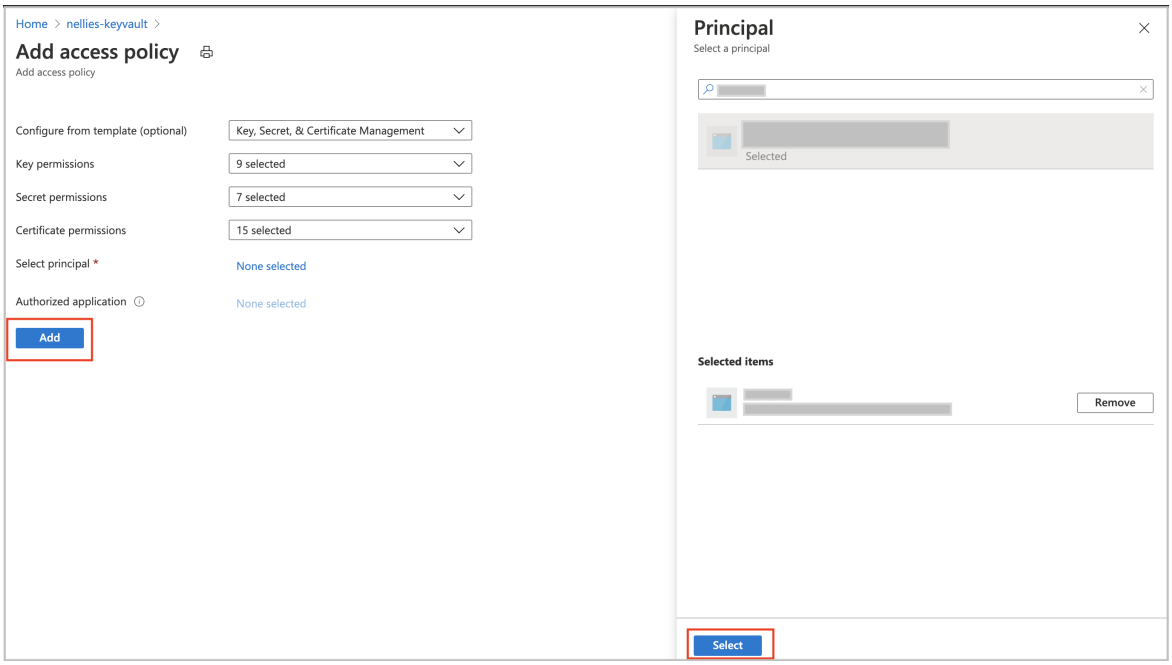
You can create a Speech resource in the Azure portal:



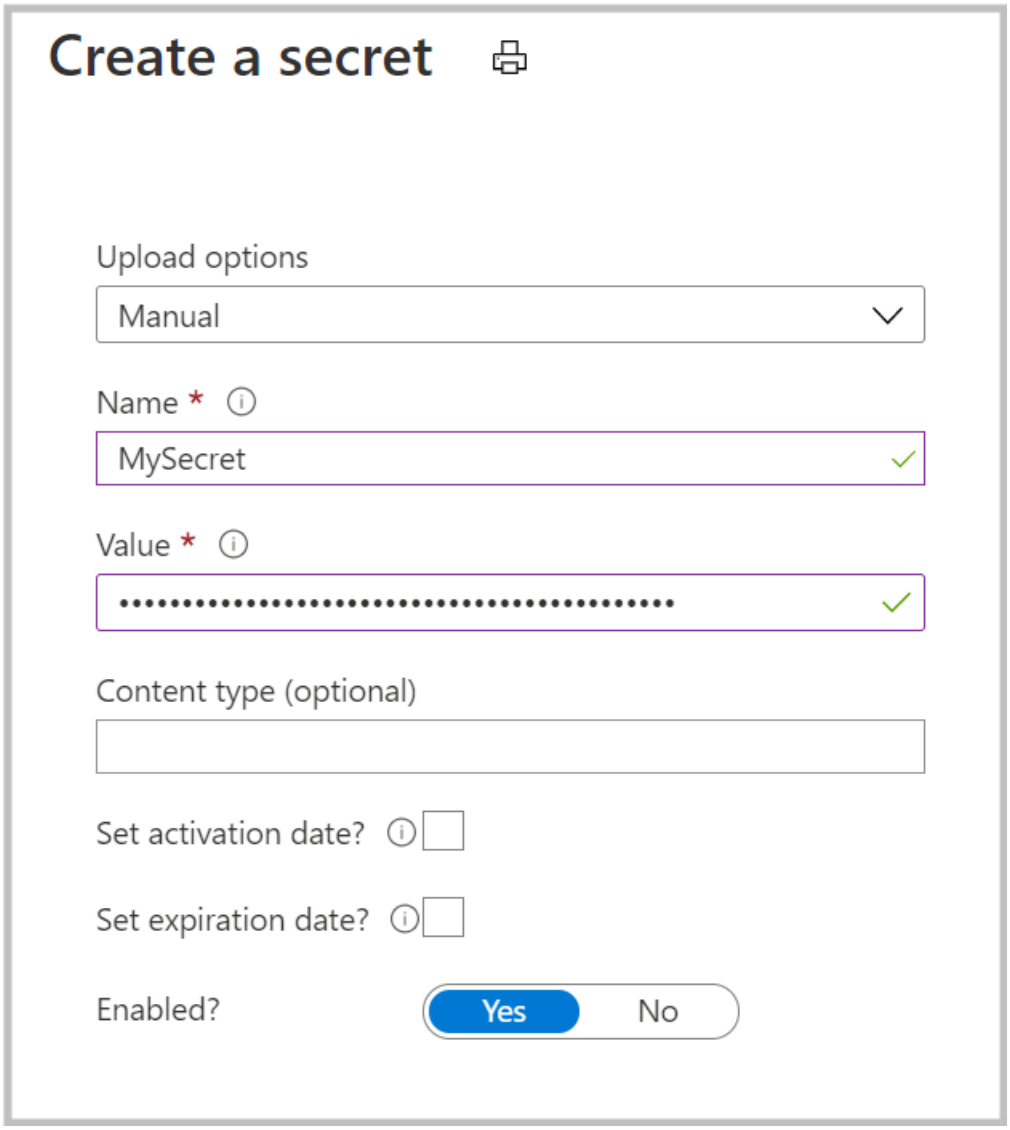
Create a key vault and configure secrets and access

1. Create a key vault in the Azure portal.
2. Go to **Key Vault** > **Access policies**, and grant the Azure Synapse workspace MSI permissions to read secrets from Azure Key Vault.

> **NOTE**
>
> Make sure that the policy changes are saved. This step is easy to miss.



3. Go to your Cognitive Services resource. For example, go to **Anomaly Detector** > **Keys and Endpoint**. Then copy either of the two keys to the clipboard.
4. Go to **Key Vault** > **Secret** to create a new secret. Specify the name of the secret, and then paste the key from the previous step into the **Value** field. Finally, select **Create**.

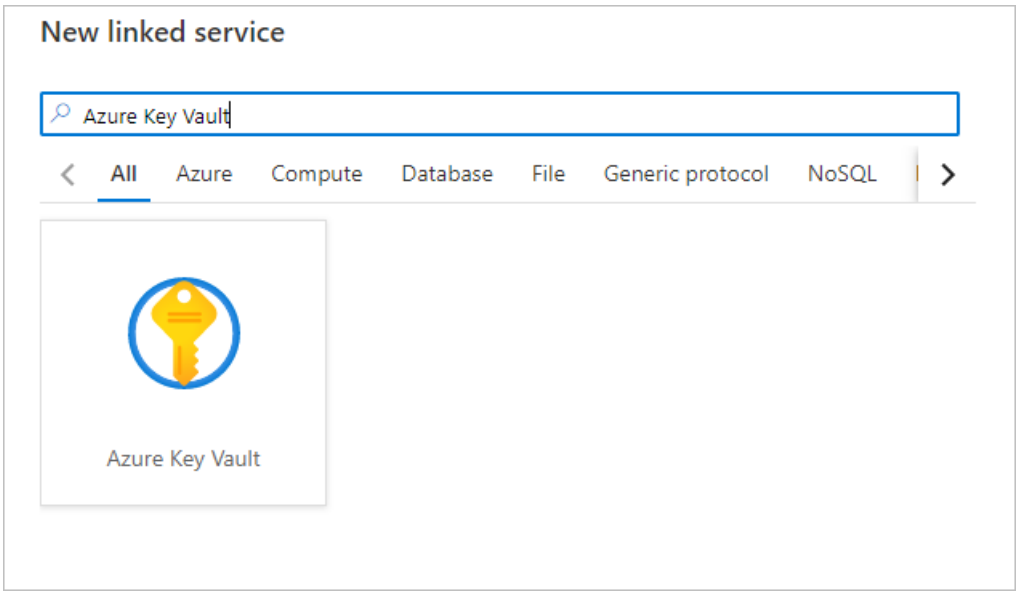


IMPORTANT

Make sure you remember or note down this secret name. You'll use it later when you create the Azure Cognitive Services linked service.

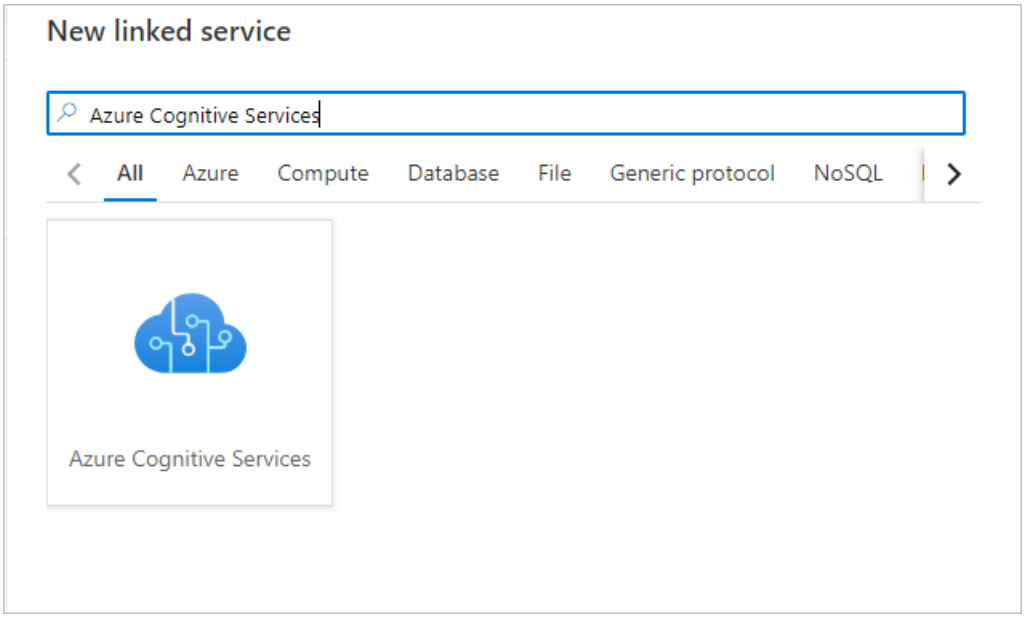
Create an Azure Key Vault linked service in Azure Synapse

1. Open your workspace in Synapse Studio.
2. Go to **Manage** > **Linked Services**. Create an **Azure Key Vault** linked service by pointing to the key vault that you just created.
3. Verify the connection by selecting the **Test connection** button. If the connection is green, select **Create** and then select **Publish all** to save your change.



Create an Azure Cognitive Service linked service in Azure Synapse

1. Open your workspace in Synapse Studio.
2. Go to **Manage** > **Linked Services**. Create an **Azure Cognitive Services** linked service by pointing to the Cognitive Service that you just created.
3. Verify the connection by selecting the **Test connection** button. If the connection is green, select **Create** and then select **Publish all** to save your change.



You're now ready to continue with one of the tutorials for using the Azure Cognitive Services experience in Synapse Studio.

Next steps

- Tutorial: Sentiment analysis with Azure Cognitive Services
- Tutorial: Anomaly detection with Azure Cognitive Services
- Tutorial: Machine learning model scoring in Azure Synapse dedicated SQL Pools.
- Machine Learning capabilities in Azure Synapse Analytics

Train machine learning models

8/19/2022 • 3 minutes to read • Edit Online

Apache Spark in Azure Synapse Analytics enables machine learning with big data, providing the ability to obtain valuable insight from large amounts of structured, unstructured, and fast-moving data. There are several options when training machine learning models using Azure Spark in Azure Synapse Analytics: Apache Spark MLlib, Azure Machine Learning, and various other open-source libraries.

Apache SparkML and MLlib

Apache Spark in Azure Synapse Analytics is one of Microsoft's implementations of Apache Spark in the cloud. It provides a unified, open-source, parallel data processing framework supporting in-memory processing to boost big data analytics. The Spark processing engine is built for speed, ease of use, and sophisticated analytics. Spark's in-memory distributed computation capabilities make it a good choice for the iterative algorithms used in machine learning and graph computations.

There are two scalable machine learning libraries that bring algorithmic modeling capabilities to this distributed environment: MLlib and SparkML. MLlib contains the original API built on top of RDDs. SparkML is a newer package that provides a higher-level API built on top of DataFrames for constructing ML pipelines. SparkML doesn't yet support all of the features of MLlib, but is replacing MLlib as Spark's standard machine learning library.

> **NOTE**
> You can learn more about creating a SparkML model by following this tutorial.

Popular libraries

Every Apache Spark pool in Azure Synapse Analytics comes with a set of pre-loaded and popular machine learning libraries. These libraries provide reusable code that you may want to include in your programs or projects. Some of the relevant machine learning libraries that are included by default include:

- Scikit-learn is one of the most popular single-node machine learning libraries for classical ML algorithms. Scikit-learn supports most of the supervised and unsupervised learning algorithms and can also be used for data-mining and data-analysis.
- XGBoost is a popular machine learning library that contains optimized algorithms for training decision trees and random forests.
- PyTorch & Tensorflow are powerful Python deep learning libraries. Within an Apache Spark pool in Azure Synapse Analytics, you can use these libraries to build single-machine models by setting the number of executors on your pool to zero. Even though Apache Spark is not functional under this configuration, it is a simple and cost-effective way to create single-machine models.

You can learn more about the available libraries and related versions by viewing the published Azure Synapse Analytics runtime.

MMLSpark

The Microsoft Machine Learning library for Apache Spark is MMLSpark. This library is designed to make data scientists more productive on Spark, increase the rate of experimentation, and leverage cutting-edge machine

learning techniques, including deep learning, on large datasets.

MMLSpark provides a layer on top of SparkML's low-level APIs when building scalable ML models, such as indexing strings, coercing data into a layout expected by machine learning algorithms, and assembling feature vectors. The MMLSpark library simplifies these and other common tasks for building models in PySpark.

Automated ML in Azure Machine Learning

Azure Machine Learning is a cloud-based environment that allows you to train, deploy, automate, manage, and track machine learning models. Automated ML in Azure Machine Learning accepts training data and configuration settings and automatically iterates through combinations of different feature normalization/standardization methods, models, and hyperparameter settings to arrive at the best model.

When using automated ML within Azure Synapse Analytics, you can leverage the deep integration between the different services to simplify authentication & model training.

> **NOTE**
> You can learn more about creating an Azure Machine Learning automated ML experiment by following this tutorial.

Azure Cognitive Services

Azure Cognitive Services provides machine learning capabilities to solve general problems such as analyzing text for emotional sentiment or analyzing images to recognize objects or faces. You don't need special machine learning or data science knowledge to use these services. A Cognitive Service provides part or all of the components in a machine learning solution: data, algorithm, and trained model. These services are meant to require general knowledge about your data without needing experience with machine learning or data science. You can leverage these pre-trained Cognitive Services automatically within Azure Synapse Analytics.

Next steps

This article provides an overview of the various options to train machine learning models within Apache Spark pools in Azure Synapse Analytics. You can learn more about model training by following the tutorial below:

- Run Automated ML experiments using Azure Machine Learning and Azure Synapse Analytics: Automated ML Tutorial
- Run SparkML experiments: Apache SparkML Tutorial
- View the default libraries: Azure Synapse Analytics runtime

Deep learning (Preview)

8/19/2022 • 2 minutes to read • Edit Online

Apache Spark in Azure Synapse Analytics enables machine learning with big data, providing the ability to obtain valuable insight from large amounts of structured, unstructured, and fast-moving data. There are several options when training machine learning models using Azure Spark in Azure Synapse Analytics: Apache Spark MLlib, Azure Machine Learning, and various other open-source libraries.

GPU-enabled Apache Spark pools

To simplify the process for creating and managing pools, Azure Synapse takes care of pre-installing low-level libraries and setting up all the complex networking requirements between compute nodes. This integration allows users to get started with GPU- accelerated pools within just a few minutes. To learn more about how to create a GPU-accelerated pool, you can visit the quickstart on how to create a GPU-accelerated pool.

NOTE

- GPU-accelerated pools can be created in workspaces located in East US, Australia East, and North Europe.
- GPU-accelerated pools are only available with the Apache Spark 3.1 and 3.2 runtime.
- You might need to request a limit increase in order to create GPU-enabled clusters.

GPU ML Environment

Azure Synapse Analytics provides built-in support for deep learning infrastructure. The Azure Synapse Analytics runtimes for Apache Spark 3 include support for the most common deep learning libraries like TensorFlow and PyTorch. The Azure Synapse runtime also includes supporting libraries like Petastorm and Horovod which are commonly used for distributed training.

Tensorflow

TensorFlow is an open source machine learning framework for all developers. It is used for implementing machine learning and deep learning applications.

For more information about Tensorflow, you can visit the Tensorflow API documentation.

PyTorch

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs.

For more information about PyTorch, you can visit the PyTorch documentation.

Horovod

Horovod is a distributed deep learning training framework for TensorFlow, Keras, and PyTorch. Horovod was developed to make distributed deep learning fast and easy to use. With this framework, an existing training script can be scaled up to run on hundreds of GPUs in just a few lines of code. In addition, Horovod can run on top of Apache Spark, making it possible to unify data processing and model training into a single pipeline.

To learn more about how to run distributed training jobs in Azure Synapse Analytics, you can visit the following tutorials: - Tutorial: Distributed training with Horovod and PyTorch - Tutorial: Distributed training with Horovod and Tensorflow

For more information about Horovod, you can visit the Horovod documentation,

Petastorm

Petastorm is an open source data access library which enables single-node or distributed training of deep learning models. This library enables training directly from datasets in Apache Parquet format and datasets that have already been loaded as an Apache Spark DataFrame. Petastorm supports popular training frameworks such as Tensorflow and PyTorch.

For more information about Petastorm, you can visit the Petastorm GitHub page or the Petastorm API documentation.

Next steps

This article provides an overview of the various options to train machine learning models within Apache Spark pools in Azure Synapse Analytics. You can learn more about model training by following the tutorial below:

- Run SparkML experiments: Apache SparkML Tutorial
- View libraries within the Apache Spark 3 runtime: Apache Spark 3 Runtime
- Accelerate ETL workloads with RAPIDS: Apache Spark Rapids

Analyze data with Apache Spark

8/19/2022 • 5 minutes to read • Edit Online

In this tutorial, you'll learn how to perform exploratory data analysis by using Azure Open Datasets and Apache Spark. You can then visualize the results in a Synapse Studio notebook in Azure Synapse Analytics.

In particular, we'll analyze the New York City (NYC) Taxi dataset. The data is available through Azure Open Datasets. This subset of the dataset contains information about yellow taxi trips: information about each trip, the start and end time and locations, the cost, and other interesting attributes.

Before you begin

Create an Apache Spark Pool by following the Create an Apache Spark pool tutorial.

Download and prepare the data

1. Create a notebook by using the PySpark kernel. For instructions, see Create a notebook.

   > **NOTE**
   > Because of the PySpark kernel, you don't need to create any contexts explicitly. The Spark context is automatically created for you when you run the first code cell.

2. In this tutorial, we'll use several different libraries to help us visualize the dataset. To do this analysis, import the following libraries:

   ```
   import matplotlib.pyplot as plt
   import seaborn as sns
   import pandas as pd
   ```

3. Because the raw data is in a Parquet format, you can use the Spark context to pull the file into memory as a DataFrame directly. Create a Spark DataFrame by retrieving the data via the Open Datasets API. Here, we use the Spark DataFrame *schema on read* properties to infer the datatypes and schema.

   ```
   from azureml.opendatasets import NycTlcYellow
   from datetime import datetime
   from dateutil import parser

   end_date = parser.parse('2018-06-06')
   start_date = parser.parse('2018-05-01')
   nyc_tlc = NycTlcYellow(start_date=start_date, end_date=end_date)
   df = nyc_tlc.to_spark_dataframe()
   ```

4. After the data is read, we'll want to do some initial filtering to clean the dataset. We might remove unneeded columns and add columns that extract important information. In addition, we'll filter out anomalies within the dataset.

```
# Filter the dataset
from pyspark.sql.functions import *

filtered_df = df.select('vendorID', 'passengerCount', 'tripDistance','paymentType', 'fareAmount',
'tipAmount'\
                                , date_format('tpepPickupDateTime', 'hh').alias('hour_of_day')\
                                , dayofweek('tpepPickupDateTime').alias('day_of_week')\
                                , dayofmonth(col('tpepPickupDateTime')).alias('day_of_month'))\
                            .filter((df.passengerCount > 0)\
                                & (df.tipAmount >= 0)\
                                & (df.fareAmount >= 1) & (df.fareAmount <= 250)\
                                & (df.tripDistance > 0) & (df.tripDistance <= 200))

filtered_df.createOrReplaceTempView("taxi_dataset")
```

Analyze data

As a data analyst, you have a wide range of tools available to help you extract insights from the data. In this part of the tutorial, we'll walk through a few useful tools available within Azure Synapse Analytics notebooks. In this analysis, we want to understand the factors that yield higher taxi tips for our selected period.

Apache Spark SQL Magic

First, we'll perform exploratory data analysis by Apache Spark SQL and magic commands with the Azure Synapse notebook. After we have our query, we'll visualize the results by using the built-in `chart options` capability.

1. Within your notebook, create a new cell and copy the following code. By using this query, we want to understand how the average tip amounts have changed over the period we've selected. This query will also help us identify other useful insights, including the minimum/maximum tip amount per day and the average fare amount.

   ```
   %%sql
   SELECT
       day_of_month
       , MIN(tipAmount) AS minTipAmount
       , MAX(tipAmount) AS maxTipAmount
       , AVG(tipAmount) AS avgTipAmount
       , AVG(fareAmount) as fareAmount
   FROM taxi_dataset
   GROUP BY day_of_month
   ORDER BY day_of_month ASC
   ```

2. After our query finishes running, we can visualize the results by switching to the chart view. This example creates a line chart by specifying the `day_of_month` field as the key and `avgTipAmount` as the value. After you've made the selections, select **Apply** to refresh your chart.

Visualize data

In addition to the built-in notebook charting options, you can use popular open-source libraries to create your own visualizations. In the following examples, we'll use Seaborn and Matplotlib. These are commonly used Python libraries for data visualization.

NOTE

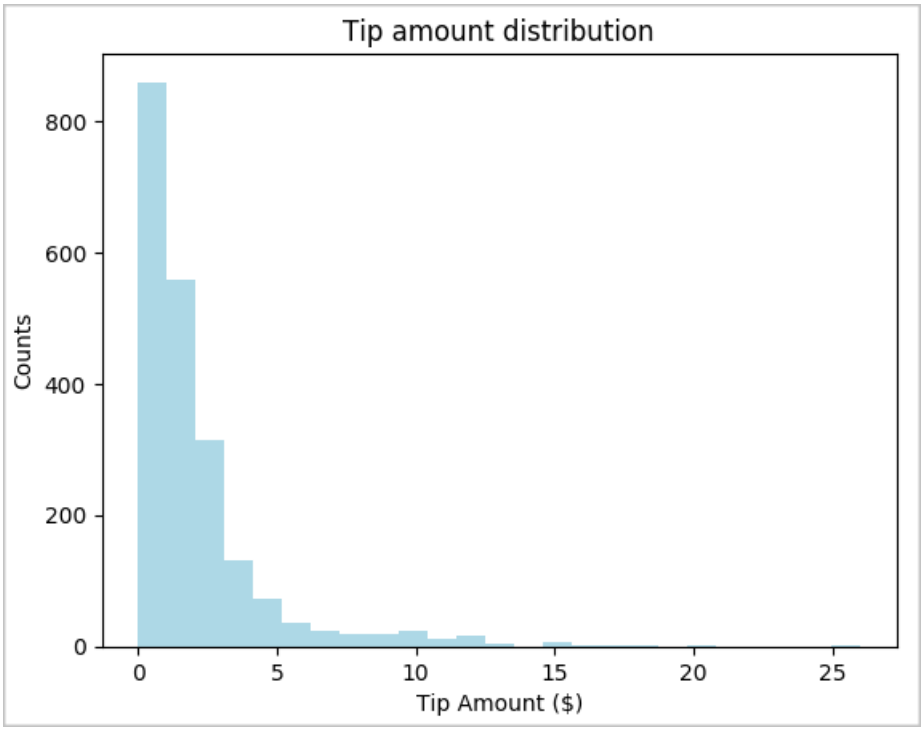
By default, every Apache Spark pool in Azure Synapse Analytics contains a set of commonly used and default libraries. You can view the full list of libraries in the Azure Synapse runtime documentation. In addition, to make third-party or locally built code available to your applications, you can install a library onto one of your Spark pools.

1. To make development easier and less expensive, we'll downsample the dataset. We'll use the built-in Apache Spark sampling capability. In addition, both Seaborn and Matplotlib require a Pandas DataFrame or NumPy array. To get a Pandas DataFrame, use the `toPandas()` command to convert the DataFrame.

```
# To make development easier, faster, and less expensive, downsample for now
sampled_taxi_df = filtered_df.sample(True, 0.001, seed=1234)

# The charting package needs a Pandas DataFrame or NumPy array to do the conversion
sampled_taxi_pd_df = sampled_taxi_df.toPandas()
```

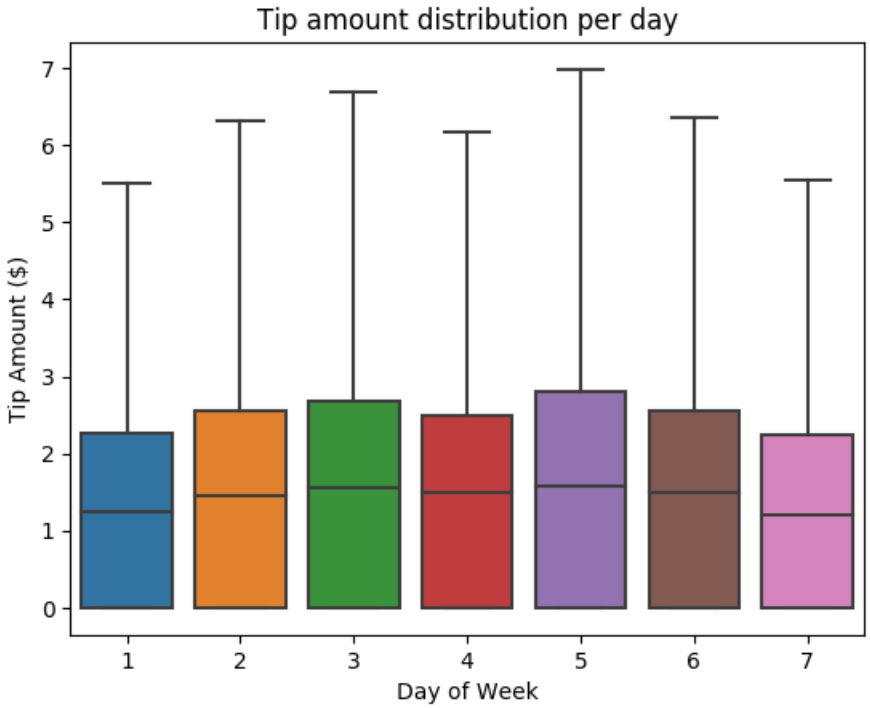
2. We want to understand the distribution of tips in our dataset. We'll use Matplotlib to create a histogram that shows the distribution of tip amount and count. Based on the distribution, we can see that tips are skewed toward amounts less than or equal to $10.

```
# Look at a histogram of tips by count by using Matplotlib

ax1 = sampled_taxi_pd_df['tipAmount'].plot(kind='hist', bins=25, facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()
```



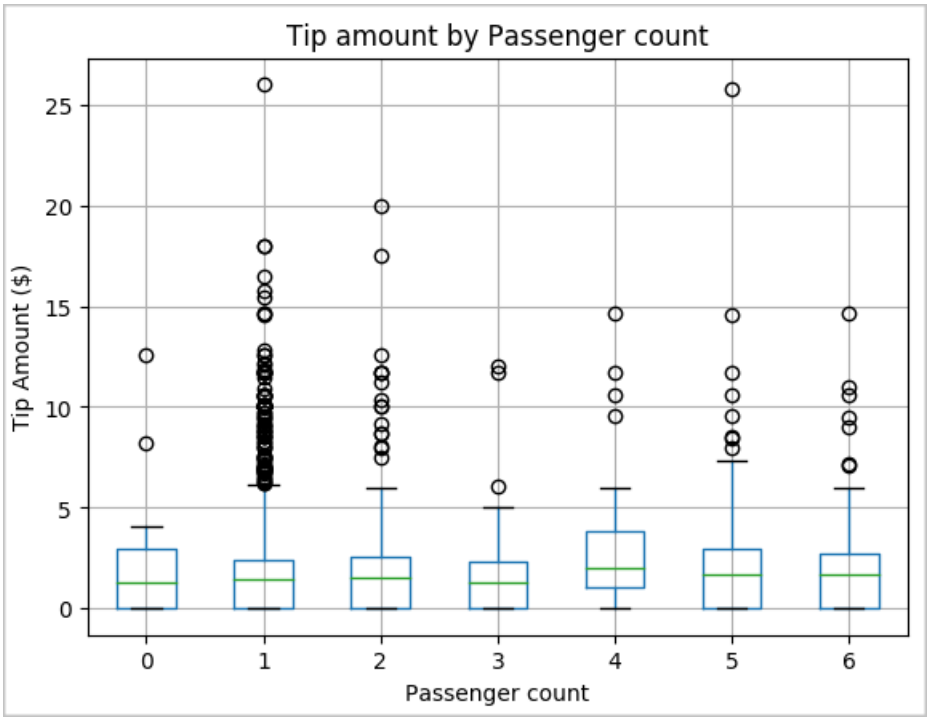
3. Next, we want to understand the relationship between the tips for a given trip and the day of the week. Use Seaborn to create a box plot that summarizes the trends for each day of the week.

```
# View the distribution of tips by day of week using Seaborn
ax = sns.boxplot(x="day_of_week", y="tipAmount",data=sampled_taxi_pd_df, showfliers = False)
ax.set_title('Tip amount distribution per day')
ax.set_xlabel('Day of Week')
ax.set_ylabel('Tip Amount ($)')
plt.show()
```



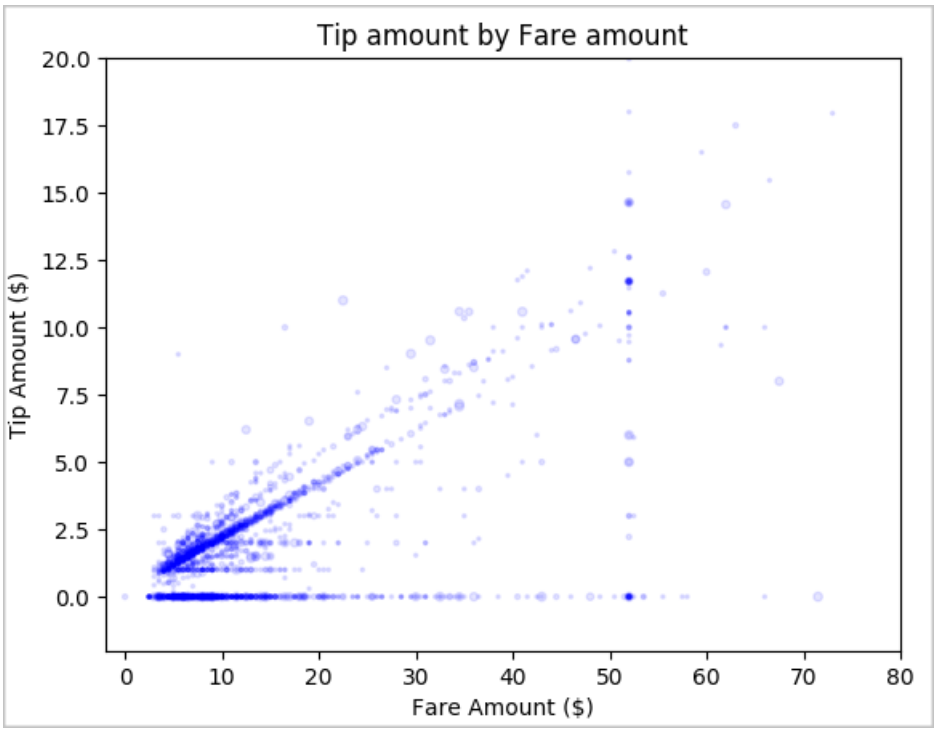
4. Another hypothesis of ours might be that there's a positive relationship between the number of passengers and the total taxi tip amount. To verify this relationship, run the following code to generate a box plot that illustrates the distribution of tips for each passenger count.

```
# How many passengers tipped by various amounts
ax2 = sampled_taxi_pd_df.boxplot(column=['tipAmount'], by=['passengerCount'])
ax2.set_title('Tip amount by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
ax2.set_ylim(0,30)
plt.suptitle('')
plt.show()
```



5. Last, we want to understand the relationship between the fare amount and the tip amount. Based on the results, we can see that there are several observations where people don't tip. However, we also see a positive relationship between the overall fare and tip amounts.

```
# Look at the relationship between fare and tip amounts

ax = sampled_taxi_pd_df.plot(kind='scatter', x= 'fareAmount', y = 'tipAmount', c='blue', alpha =
0.10, s=2.5*(sampled_taxi_pd_df['passengerCount']))
ax.set_title('Tip amount by Fare amount')
ax.set_xlabel('Fare Amount ($)')
ax.set_ylabel('Tip Amount ($)')
plt.axis([-2, 80, -2, 20])
plt.suptitle('')
plt.show()
```



Shut down the Spark instance

After you finish running the application, shut down the notebook to release the resources. Either close the tab or select **End Session** from the status panel at the bottom of the notebook.

See also

- Overview: Apache Spark on Azure Synapse Analytics
- Build a machine learning model with Apache SparkML

Next steps

- Azure Synapse Analytics
- Apache Spark official documentation

Tutorial: Build a machine learning app with Apache Spark MLlib and Azure Synapse Analytics

8/19/2022 • 8 minutes to read • Edit Online

In this article, you'll learn how to use Apache Spark MLlib to create a machine learning application that does simple predictive analysis on an Azure open dataset. Spark provides built-in machine learning libraries. This example uses *classification* through logistic regression.

SparkML and MLlib are core Spark libraries that provide many utilities that are useful for machine learning tasks, including utilities that are suitable for:

- Classification
- Regression
- Clustering
- Topic modeling
- Singular value decomposition (SVD) and principal component analysis (PCA)
- Hypothesis testing and calculating sample statistics

Understand classification and logistic regression

Classification, a popular machine learning task, is the process of sorting input data into categories. It's the job of a classification algorithm to figure out how to assign *labels* to input data that you provide. For example, you can think of a machine learning algorithm that accepts stock information as input and divide the stock into two categories: stocks that you should sell and stocks that you should keep.

Logistic regression is an algorithm that you can use for classification. Spark's logistic regression API is useful for *binary classification*, or classifying input data into one of two groups. For more information about logistic regression, see Wikipedia.

In summary, the process of logistic regression produces a *logistic function* that you can use to predict the probability that an input vector belongs in one group or the other.

Predictive analysis example on NYC taxi data

In this example, you use Spark to perform some predictive analysis on taxi-trip tip data from New York. The data is available through Azure Open Datasets. This subset of the dataset contains information about yellow taxi trips, including information about each trip, the start and end time and locations, the cost, and other interesting attributes.

> **IMPORTANT**
> There might be additional charges for pulling this data from its storage location.

In the following steps, you develop a model to predict whether a particular trip includes a tip or not.

Create an Apache Spark machine learning model

1. Create a notebook by using the PySpark kernel. For instructions, see Create a notebook.

2. Import the types required for this application. Copy and paste the following code into an empty cell, and

then press Shift+Enter. Or run the cell by using the blue play icon to the left of the code.

```
import matplotlib.pyplot as plt
from datetime import datetime
from dateutil import parser
from pyspark.sql.functions import unix_timestamp, date_format, col, when
from pyspark.ml import Pipeline
from pyspark.ml import PipelineModel
from pyspark.ml.feature import RFormula
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorIndexer
from pyspark.ml.classification import LogisticRegression
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

Because of the PySpark kernel, you don't need to create any contexts explicitly. The Spark context is automatically created for you when you run the first code cell.

Construct the input DataFrame

Because the raw data is in a Parquet format, you can use the Spark context to pull the file into memory as a DataFrame directly. Although the code in the following steps uses the default options, it's possible to force mapping of data types and other schema attributes if needed.

1. Run the following lines to create a Spark DataFrame by pasting the code into a new cell. This step retrieves the data via the Open Datasets API. Pulling all of this data generates about 1.5 billion rows.

   Depending on the size of your serverless Apache Spark pool, the raw data might be too large or take too much time to operate on. You can filter this data down to something smaller. The following code example uses `start_date` and `end_date` to apply a filter that returns a single month of data.

   ```
   from azureml.opendatasets import NycTlcYellow

   end_date = parser.parse('2018-06-06')
   start_date = parser.parse('2018-05-01')
   nyc_tlc = NycTlcYellow(start_date=start_date, end_date=end_date)
   filtered_df = nyc_tlc.to_spark_dataframe()
   ```

2. The downside to simple filtering is that, from a statistical perspective, it might introduce bias into the data. Another approach is to use the sampling built into Spark.

   The following code reduces the dataset to about 2,000 rows, if it's applied after the preceding code. You can use this sampling step instead of the simple filter or in conjunction with the simple filter.

   ```
   # To make development easier, faster, and less expensive, downsample for now
   sampled_taxi_df = filtered_df.sample(True, 0.001, seed=1234)
   ```

3. It's now possible to look at the data to see what was read. It's normally better to review data with a subset rather than the full set, depending on the size of the dataset.

   The following code offers two ways to view the data. The first way is basic. The second way provides a much richer grid experience, along with the capability to visualize the data graphically.

   ```
   #sampled_taxi_df.show(5)
   display(sampled_taxi_df)
   ```

4. Depending on the size of the generated dataset, and your need to experiment or run the notebook many

times, you might want to cache the dataset locally in the workspace. There are three ways to perform explicit caching:

- Save the DataFrame locally as a file.
- Save the DataFrame as a temporary table or view.
- Save the DataFrame as a permanent table.

The first two of these approaches are included in the following code examples.

Creating a temporary table or view provides different access paths to the data, but it lasts only for the duration of the Spark instance session.

```
sampled_taxi_df.createOrReplaceTempView("nytaxi")
```

Prepare the data

The data in its raw form is often not suitable for passing directly to a model. You must perform a series of actions on the data to get it into a state where the model can consume it.

In the following code, you perform four classes of operations:

- The removal of outliers or incorrect values through filtering.
- The removal of columns, which are not needed.
- The creation of new columns derived from the raw data to make the model work more effectively. This operation is sometimes called featurization.
- Labeling. Because you're undertaking binary classification (will there be a tip or not on a given trip), there's a need to convert the tip amount into a 0 or 1 value.

```
taxi_df = sampled_taxi_df.select('totalAmount', 'fareAmount', 'tipAmount', 'paymentType', 'rateCodeId',
'passengerCount'\
                                , 'tripDistance', 'tpepPickupDateTime', 'tpepDropoffDateTime'\
                                , date_format('tpepPickupDateTime', 'hh').alias('pickupHour')\
                                , date_format('tpepPickupDateTime', 'EEEE').alias('weekdayString')\
                                , (unix_timestamp(col('tpepDropoffDateTime')) -
unix_timestamp(col('tpepPickupDateTime'))).alias('tripTimeSecs')\
                                , (when(col('tipAmount') > 0, 1).otherwise(0)).alias('tipped')
                                )\
                        .filter((sampled_taxi_df.passengerCount > 0) & (sampled_taxi_df.passengerCount < 8)\
                                & (sampled_taxi_df.tipAmount >= 0) & (sampled_taxi_df.tipAmount <= 25)\
                                & (sampled_taxi_df.fareAmount >= 1) & (sampled_taxi_df.fareAmount <= 250)\
                                & (sampled_taxi_df.tipAmount < sampled_taxi_df.fareAmount)\
                                & (sampled_taxi_df.tripDistance > 0) & (sampled_taxi_df.tripDistance <=
100)\
                                & (sampled_taxi_df.rateCodeId <= 5)
                                & (sampled_taxi_df.paymentType.isin({"1", "2"}))
                                )
```

You then make a second pass over the data to add the final features.

```
taxi_featurised_df = taxi_df.select('totalAmount', 'fareAmount', 'tipAmount', 'paymentType',
'passengerCount'\
                                                , 'tripDistance', 'weekdayString',
'pickupHour','tripTimeSecs','tipped'\
                                                , when((taxi_df.pickupHour <= 6) | (taxi_df.pickupHour >=
20),"Night")\
                                                .when((taxi_df.pickupHour >= 7) & (taxi_df.pickupHour <=
10), "AMRush")\
                                                .when((taxi_df.pickupHour >= 11) & (taxi_df.pickupHour <=
15), "Afternoon")\
                                                .when((taxi_df.pickupHour >= 16) & (taxi_df.pickupHour <=
19), "PMRush")\
                                                .otherwise(0).alias('trafficTimeBins')
                                              )\
                                       .filter((taxi_df.tripTimeSecs >= 30) & (taxi_df.tripTimeSecs <=
7200))
```

Create a logistic regression model

The final task is to convert the labeled data into a format that can be analyzed through logistic regression. The input to a logistic regression algorithm needs to be a set of *label/feature vector pairs*, where the *feature vector* is a vector of numbers that represent the input point.

So, you need to convert the categorical columns into numbers. Specifically, you need to convert the `trafficTimeBins` and `weekdayString` columns into integer representations. There are multiple approaches to performing the conversion. The following example takes the `OneHotEncoder` approach, which is common.

```
# Because the sample uses an algorithm that works only with numeric features, convert them so they can be
consumed
sI1 = StringIndexer(inputCol="trafficTimeBins", outputCol="trafficTimeBinsIndex")
en1 = OneHotEncoder(dropLast=False, inputCol="trafficTimeBinsIndex", outputCol="trafficTimeBinsVec")
sI2 = StringIndexer(inputCol="weekdayString", outputCol="weekdayIndex")
en2 = OneHotEncoder(dropLast=False, inputCol="weekdayIndex", outputCol="weekdayVec")

# Create a new DataFrame that has had the encodings applied
encoded_final_df = Pipeline(stages=[sI1, en1, sI2,
en2]).fit(taxi_featurised_df).transform(taxi_featurised_df)
```

This action results in a new DataFrame with all columns in the right format to train a model.

Train a logistic regression model

The first task is to split the dataset into a training set and a testing or validation set. The split here is arbitrary. Experiment with different split settings to see if they affect the model.

```
# Decide on the split between training and testing data from the DataFrame
trainingFraction = 0.7
testingFraction = (1-trainingFraction)
seed = 1234

# Split the DataFrame into test and training DataFrames
train_data_df, test_data_df = encoded_final_df.randomSplit([trainingFraction, testingFraction], seed=seed)
```

Now that there are two DataFrames, the next task is to create the model formula and run it against the training DataFrame. Then you can validate against the testing DataFrame. Experiment with different versions of the model formula to see the impact of different combinations.

> **NOTE**
>
> To save the model, assign the *Storage Blob Data Contributor* role to the Azure SQL Database server resource scope. For detailed steps, see Assign Azure roles using the Azure portal. Only members with owner privileges can perform this step.

```
## Create a new logistic regression object for the model
logReg = LogisticRegression(maxIter=10, regParam=0.3, labelCol = 'tipped')

## The formula for the model
classFormula = RFormula(formula="tipped ~ pickupHour + weekdayVec + passengerCount + tripTimeSecs +
tripDistance + fareAmount + paymentType+ trafficTimeBinsVec")

## Undertake training and create a logistic regression model
lrModel = Pipeline(stages=[classFormula, logReg]).fit(train_data_df)

## Saving the model is optional, but it's another form of inter-session cache
datestamp = datetime.now().strftime('%m-%d-%Y-%s')
fileName = "lrModel_" + datestamp
logRegDirfilename = fileName
lrModel.save(logRegDirfilename)

## Predict tip 1/0 (yes/no) on the test dataset; evaluation using area under ROC
predictions = lrModel.transform(test_data_df)
predictionAndLabels = predictions.select("label","prediction").rdd
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)
```

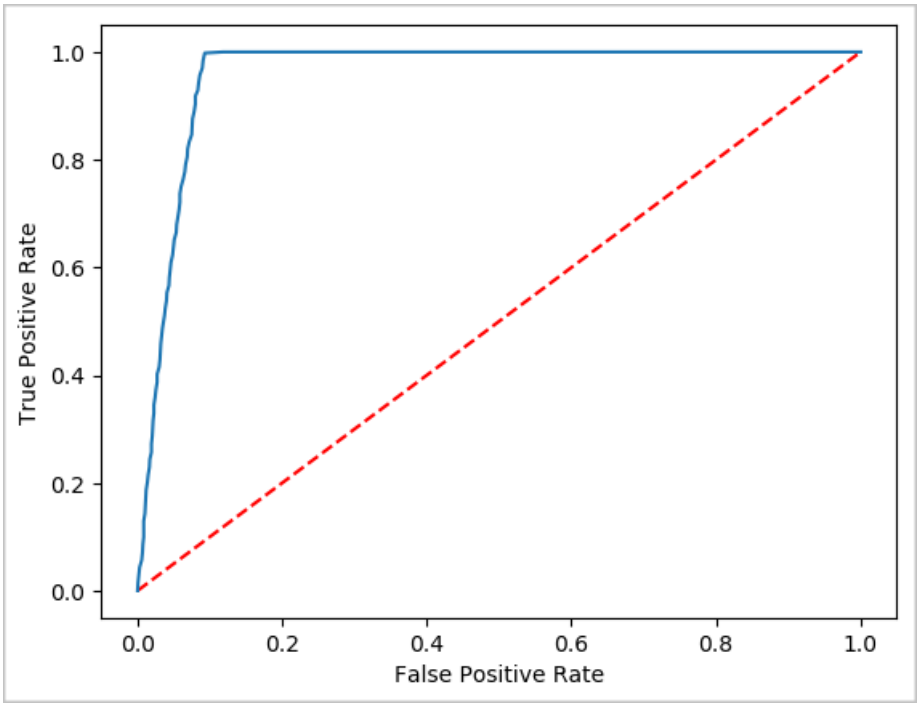
The output from this cell is:

```
Area under ROC = 0.9779470729751403
```

Create a visual representation of the prediction

You can now construct a final visualization to help you reason about the results of this test. An ROC curve is one way to review the result.

```
## Plot the ROC curve; no need for pandas, because this uses the modelSummary object
modelSummary = lrModel.stages[-1].summary

plt.plot([0, 1], [0, 1], 'r--')
plt.plot(modelSummary.roc.select('FPR').collect(),
         modelSummary.roc.select('TPR').collect())
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



Shut down the Spark instance

After you finish running the application, shut down the notebook to release the resources by closing the tab. Or select **End Session** from the status panel at the bottom of the notebook.

See also

- Overview: Apache Spark on Azure Synapse Analytics

Next steps

- .NET for Apache Spark documentation
- Azure Synapse Analytics
- Apache Spark official documentation

> **NOTE**
> Some of the official Apache Spark documentation relies on using the Spark console, which is not available on Apache Spark in Azure Synapse Analytics. Use the notebook or IntelliJ experiences instead.

Tutorial: Train a machine learning model without code

8/19/2022 • 5 minutes to read • Edit Online

You can enrich your data in Spark tables with new machine learning models that you train by using automated machine learning. In Azure Synapse Analytics, you can select a Spark table in the workspace to use as a training dataset for building machine learning models, and you can do this in a code-free experience.

In this tutorial, you learn how to train machine learning models by using a code-free experience in Synapse Studio. Synapse Studio is a feature of Azure Synapse Analytics.

You'll use automated machine learning in Azure Machine Learning, instead of coding the experience manually. The type of model that you train depends on the problem you're trying to solve. For this tutorial, you'll use a regression model to predict taxi fares from the New York City taxi dataset.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- An Azure Synapse Analytics workspace. Ensure that it has an Azure Data Lake Storage Gen2 storage account configured as the default storage. For the Data Lake Storage Gen2 file system that you work with, ensure that you're the *Storage Blob Data Contributor*.
- An Apache Spark pool (version 2.4) in your Azure Synapse Analytics workspace. For details, see Quickstart: Create a serverless Apache Spark pool using Synapse Studio.
- An Azure Machine Learning linked service in your Azure Synapse Analytics workspace. For details, see Quickstart: Create a new Azure Machine Learning linked service in Azure Synapse Analytics.

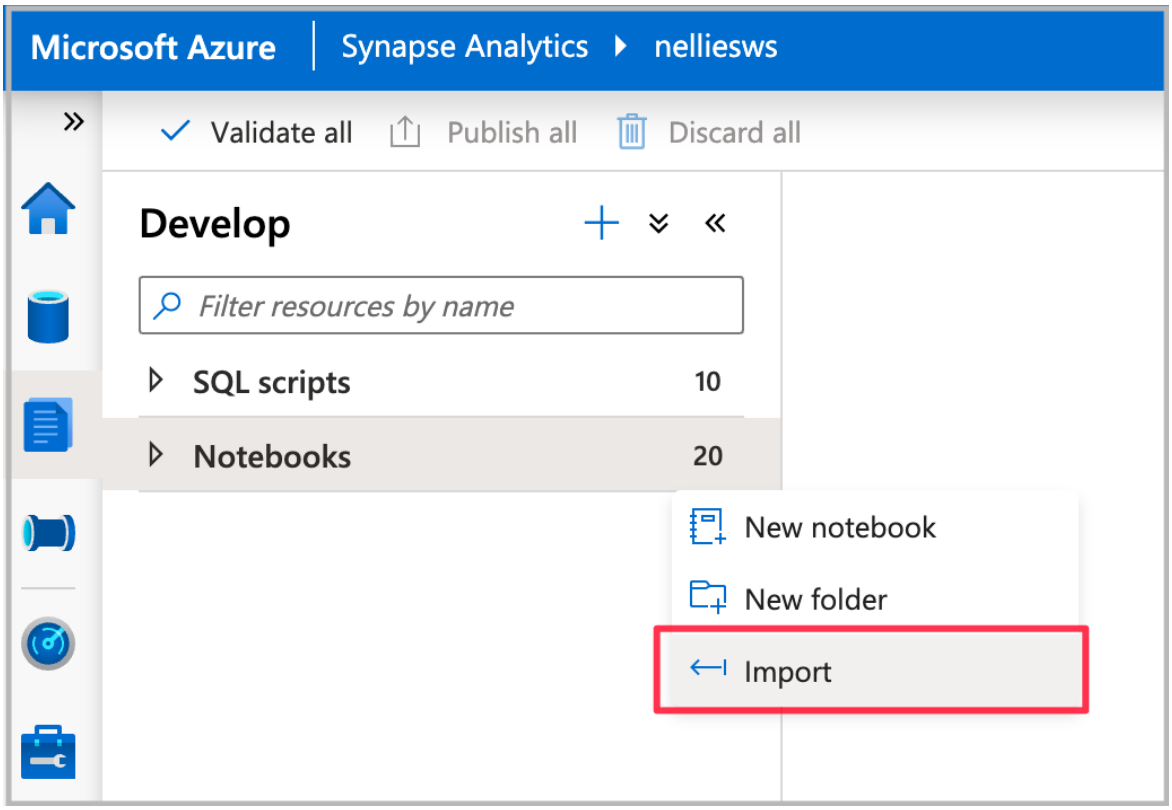
Sign in to the Azure portal

Sign in to the Azure portal.

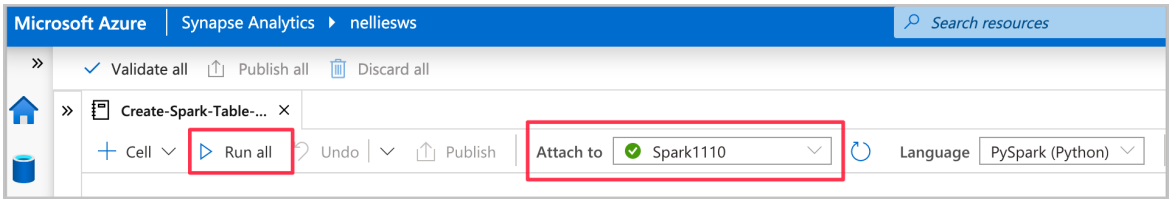
Create a Spark table for the training dataset

For this tutorial, you need a Spark table. The following notebook creates one:

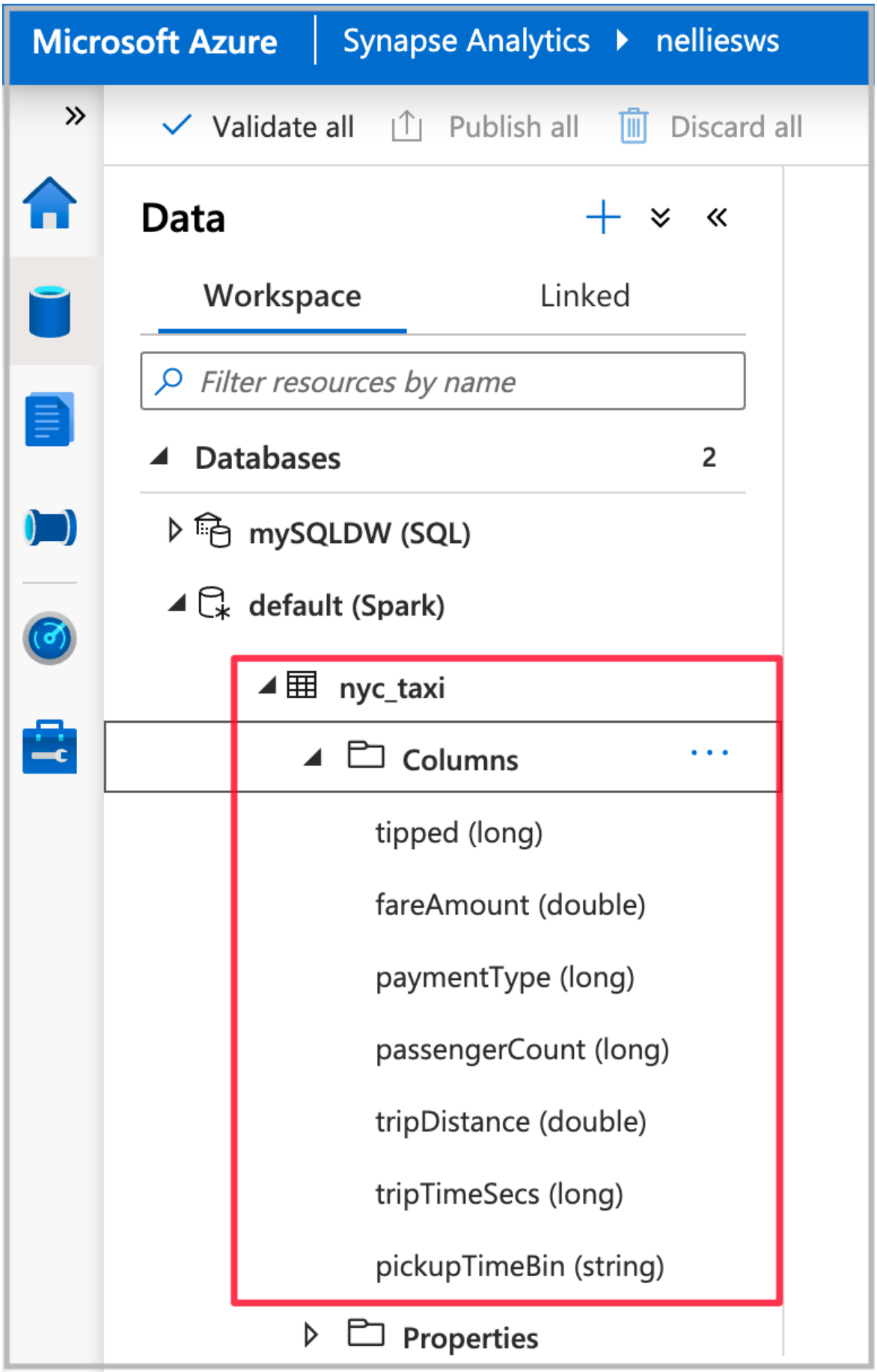
1. Download the notebook Create-Spark-Table-NYCTaxi- Data.ipynb.
2. Import the notebook to Synapse Studio.



3. Select the Spark pool that you want to use, and then select **Run all**. This step gets New York taxi data from the open dataset and saves the data to your default Spark database.

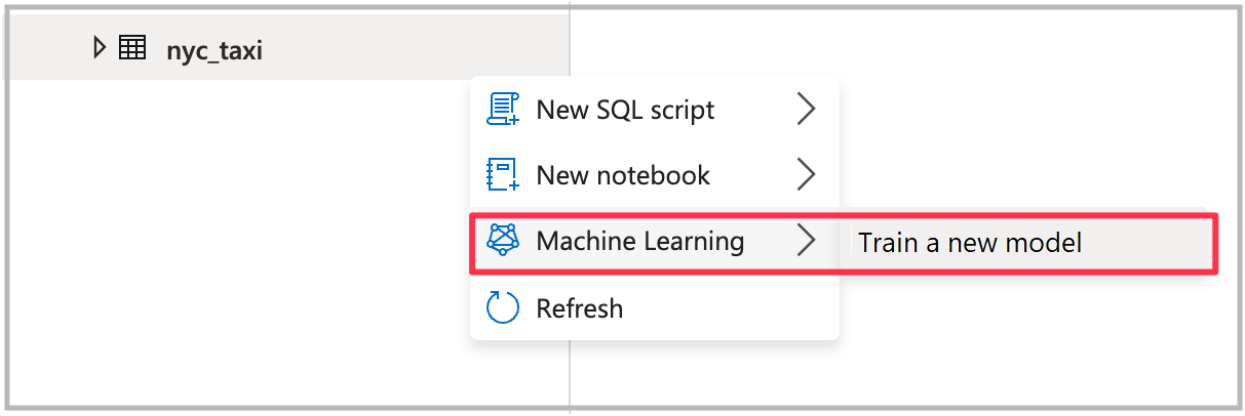


4. After the notebook run has completed, you see a new Spark table under the default Spark database. From **Data**, find the table named **nyc_taxi**.



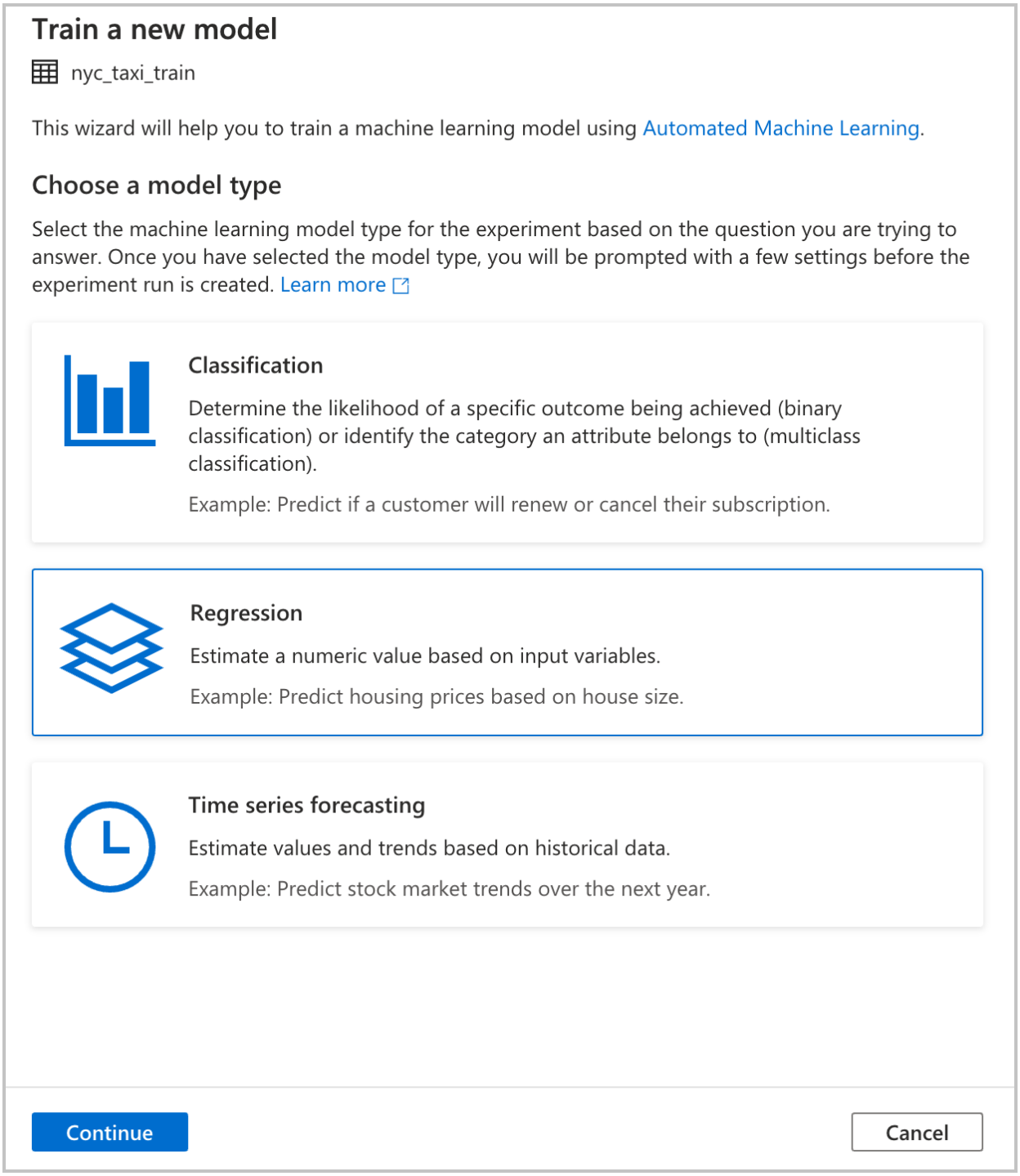
Open the automated machine learning wizard

To open the wizard, right-click the Spark table that you created in the previous step. Then select **Machine Learning** > **Train a new model**.



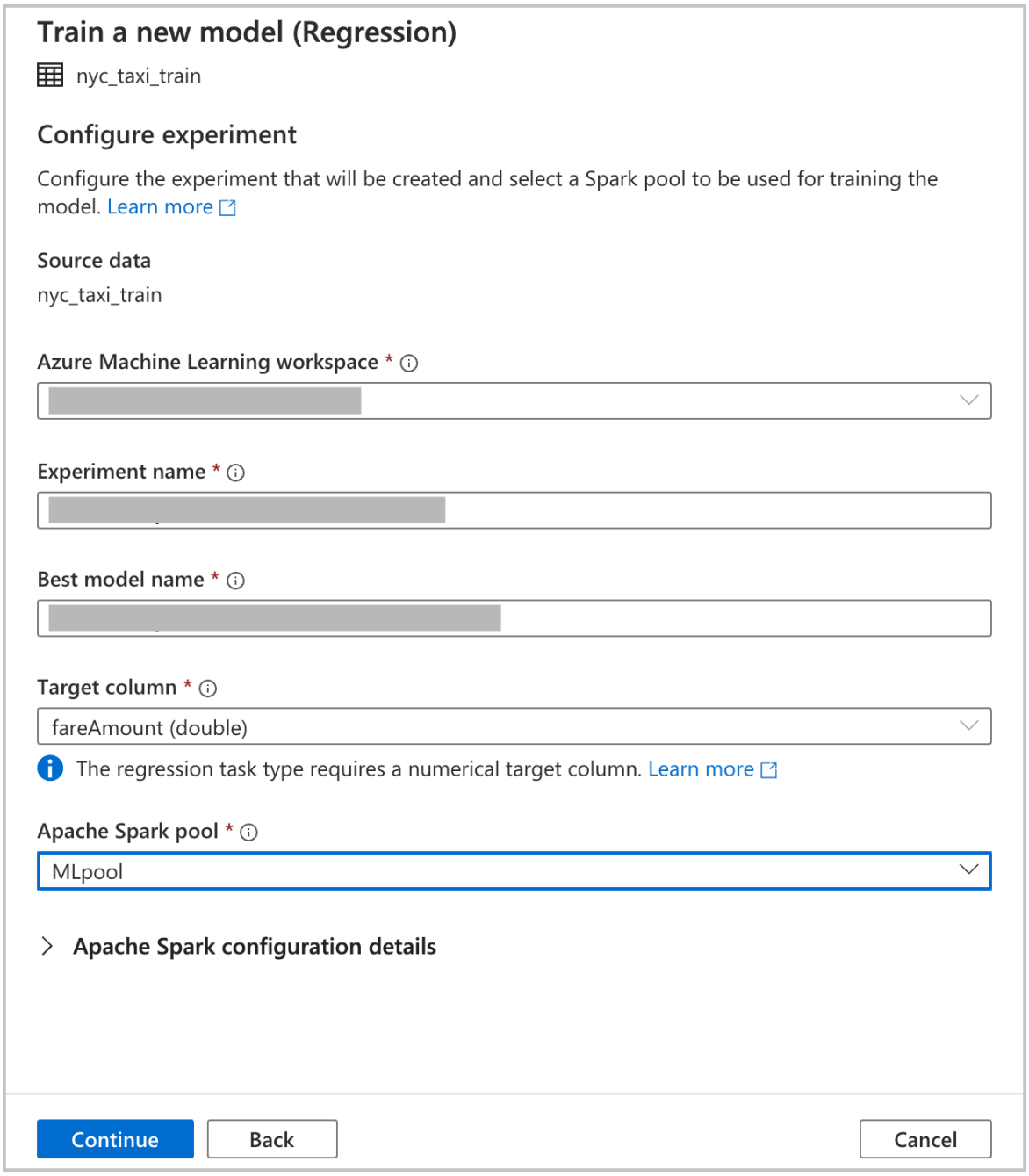
Choose a model type

Select the machine learning model type for the experiment, based on the question you're trying to answer. Because the value you're trying to predict is numeric (taxi fares), select **Regression** here. Then select **Continue**.



Configure the experiment

1. Provide configuration details for creating an automated machine learning experiment run in Azure Machine Learning. This run trains multiple models. The best model from a successful run is registered in the Azure Machine Learning model registry.



- **Azure Machine Learning workspace**: An Azure Machine Learning workspace is required for creating an automated machine learning experiment run. You also need to link your Azure Synapse Analytics workspace with the Azure Machine Learning workspace by using a linked service. After you've fulfilled all the prerequisites, you can specify the Azure Machine Learning workspace that you want to use for this automated run.
- **Experiment name**: Specify the experiment name. When you submit an automated machine learning run, you provide an experiment name. Information for the run is stored under that experiment in the Azure Machine Learning workspace. This experience creates a new experiment by default and generates a proposed name, but you can also provide the name of an existing experiment.
- **Best model name**: Specify the name of the best model from the automated run. The best model

is given this name and saved in the Azure Machine Learning model registry automatically after this run. An automated machine learning run creates many machine learning models. Based on the primary metric that you select in a later step, those models can be compared and the best model can be selected.

- **Target column**: This is what the model will be trained to predict. Choose the column in the dataset that contains the data you want to predict. For this tutorial, select the numeric column `fareAmount` as the target column.
- **Spark pool**: Specify the Spark pool that you want to use for the automated experiment run. The computations are run on the pool that you specify.
- **Spark configuration details**: In addition to the Spark pool, you have the option to provide session configuration details.

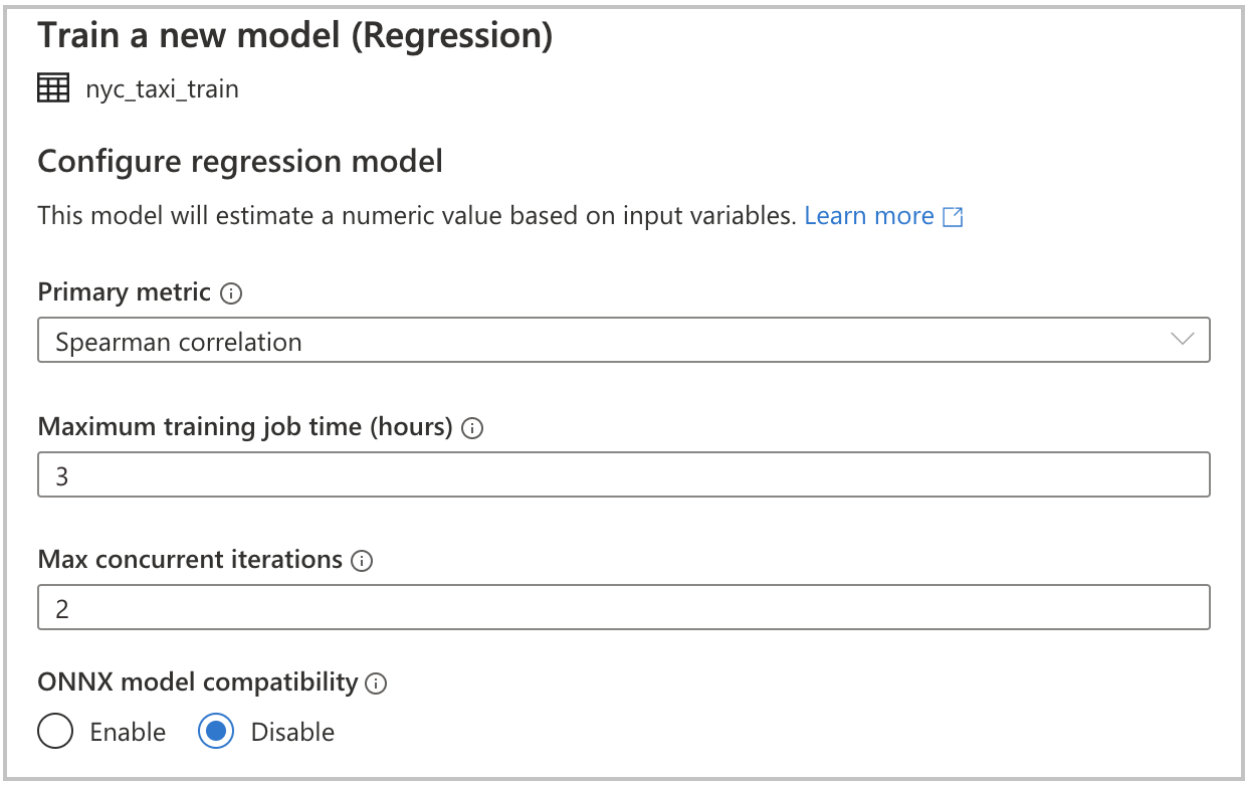
2. Select **Continue**.

Configure the model

Because you selected **Regression** as your model type in the previous section, the following configurations are available (these are also available for the **Classification** model type):

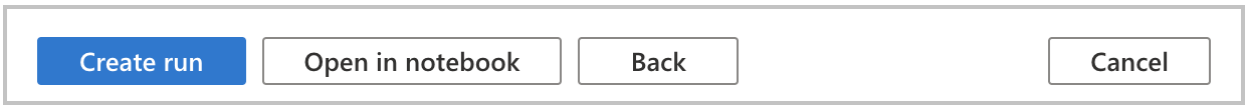
- **Primary metric**: Enter the metric that measures how well the model is doing. You use this metric to compare different models created in the automated run and determine which model performed best.
- **Training job time (hours)**: Specify the maximum amount of time, in hours, for an experiment to run and train models. Note that you can also provide values less than 1 (for example, **0.5**).
- **Max concurrent iterations**: Choose the maximum number of iterations that run in parallel.
- **ONNX model compatibility**: If you enable this option, the models trained by automated machine learning are converted to the ONNX format. This is particularly relevant if you want to use the model for scoring in Azure Synapse Analytics SQL pools.

These settings all have a default value that you can customize.



Start a run

After all the required configurations are done, you can start your automated run. You can choose to create a run directly by selecting **Create run** - this starts the run without code. Alternatively, if you prefer code, you can select **Open in notebook** - this opens a notebook containing the code that creates the run so you can view the code and start the run yourself.



> **NOTE**
> If you selected **Time series forecasting** as your model type in the previous section, you must make additional configurations. Forecasting also doesn't support ONNX model compatibility.

Create a run directly

To start your automated machine learning run directly, select **Create Run**. You see a notification that indicates the run is starting. Then you see another notification that indicates success. You can also check the status in Azure Machine Learning by selecting the link in the notification.

Create a run with a notebook

To generate a notebook, select **Open In Notebook**. This gives you an opportunity to add settings or otherwise modify the code for your automated machine learning run. When you're ready to run the code, select **Run all**.

Monitor the run

After you've successfully submitted the run, you see a link to the experiment run in the Azure Machine Learning workspace in the notebook output. Select the link to monitor your automated run in Azure Machine Learning.

Next steps

- Tutorial: Machine learning model scoring wizard (preview) for dedicated SQL pools
- Quickstart: Create a new Azure Machine Learning linked service in Azure Synapse Analytics
- Machine learning capabilities in Azure Synapse Analytics

Tutorial: Train a model in Python with automated machine learning

8/19/2022 • 10 minutes to read • Edit Online

Azure Machine Learning is a cloud-based environment that allows you to train, deploy, automate, manage, and track machine learning models.

In this tutorial, you use automated machine learning in Azure Machine Learning to create a regression model to predict taxi fare prices. This process arrives at the best model by accepting training data and configuration settings, and automatically iterating through combinations of different methods, models, and hyperparameter settings.

In this tutorial, you learn how to:

- Download the data by using Apache Spark and Azure Open Datasets.
- Transform and clean data by using Apache Spark DataFrames.
- Train a regression model in automated machine learning.
- Calculate model accuracy.

Before you begin

- Create a serverless Apache Spark pool by following the Create a serverless Apache Spark pool quickstart.
- Complete the Azure Machine Learning workspace setup tutorial if you don't have an existing Azure Machine Learning workspace.

Understand regression models

Regression models predict numerical output values based on independent predictors. In regression, the objective is to help establish the relationship among those independent predictor variables by estimating how one variable affects the others.

Example based on New York City taxi data

In this example, you use Spark to perform some analysis on taxi-trip tip data from New York City (NYC). The data is available through Azure Open Datasets. This subset of the dataset contains information about yellow taxi trips, including information about each trip, the start and end time and locations, and the cost.

> **IMPORTANT**
> There might be additional charges for pulling this data from its storage location. In the following steps, you develop a model to predict NYC taxi fare prices.

Download and prepare the data

Here's how:

1. Create a notebook by using the PySpark kernel. For instructions, see Create a notebook.

> **NOTE**
>
> Because of the PySpark kernel, you don't need to create any contexts explicitly. The Spark context is automatically created for you when you run the first code cell.

2. Because the raw data is in a Parquet format, you can use the Spark context to pull the file directly into memory as a DataFrame. Create a Spark DataFrame by retrieving the data via the Open Datasets API. Here, you use the Spark DataFrame `schema on read` properties to infer the datatypes and schema.

```
blob_account_name = "azureopendatastorage"
blob_container_name = "nyctlc"
blob_relative_path = "yellow"
blob_sas_token = r""

# Allow Spark to read from the blob remotely
wasbs_path = 'wasbs://%s@%s.blob.core.windows.net/%s' % (blob_container_name, blob_account_name,
blob_relative_path)
spark.conf.set('fs.azure.sas.%s.%s.blob.core.windows.net' % (blob_container_name,
blob_account_name),blob_sas_token)

# Spark read parquet; note that it won't load any data yet
df = spark.read.parquet(wasbs_path)
```

3. Depending on the size of your Spark pool, the raw data might be too large or take too much time to operate on. You can filter this data down to something smaller, like a month of data, by using the `start_date` and `end_date` filters. After you filter a DataFrame, you also run the `describe()` function on the new DataFrame to see summary statistics for each field.

   Based on the summary statistics, you can see that there are some irregularities in the data. For example, the statistics show that the minimum trip distance is less than 0. You need to filter out these irregular data points.

```
# Create an ingestion filter
start_date = '2015-01-01 00:00:00'
end_date = '2015-12-31 00:00:00'

filtered_df = df.filter('tpepPickupDateTime > "' + start_date + '" and tpepPickupDateTime< "' +
end_date + '"')

filtered_df.describe().show()
```

4. Generate features from the dataset by selecting a set of columns and creating various time-based features from the pickup `datetime` field. Filter out the outliers that were identified from the earlier step, and then remove the last few columns because they're unnecessary for training.

```
from datetime import datetime
from pyspark.sql.functions import *

# To make development easier, faster, and less expensive, downsample for now
sampled_taxi_df = filtered_df.sample(True, 0.001, seed=1234)

taxi_df = sampled_taxi_df.select('vendorID', 'passengerCount', 'tripDistance',  'startLon',
'startLat', 'endLon' \
                                 , 'endLat', 'paymentType', 'fareAmount', 'tipAmount'\
                                 , column('puMonth').alias('month_num') \
                                 , date_format('tpepPickupDateTime', 'hh').alias('hour_of_day')\
                                 , date_format('tpepPickupDateTime', 'EEEE').alias('day_of_week')\
                                 , dayofmonth(col('tpepPickupDateTime')).alias('day_of_month')
                                 ,(unix_timestamp(col('tpepDropoffDateTime')) -
unix_timestamp(col('tpepPickupDateTime'))).alias('trip_time'))\
                          .filter((sampled_taxi_df.passengerCount > 0) &
(sampled_taxi_df.passengerCount < 8)\
                                 & (sampled_taxi_df.tipAmount >= 0)\
                                 & (sampled_taxi_df.fareAmount >= 1) & (sampled_taxi_df.fareAmount <=
250)\
                                 & (sampled_taxi_df.tipAmount < sampled_taxi_df.fareAmount)\
                                 & (sampled_taxi_df.tripDistance > 0) & (sampled_taxi_df.tripDistance
<= 200)\
                                 & (sampled_taxi_df.rateCodeId <= 5)\
                                 & (sampled_taxi_df.paymentType.isin({"1", "2"})))
taxi_df.show(10)
```

As you can see, this will create a new DataFrame with additional columns for the day of the month, pickup hour, weekday, and total trip time.

Generate test and validation datasets

After you have your final dataset, you can split the data into training and test sets by using the `random_ split` function in Spark. By using the provided weights, this function randomly splits the data into the training dataset for model training and the validation dataset for testing.

```
# Random split dataset using Spark; convert Spark to pandas
training_data, validation_data = taxi_df.randomSplit([0.8,0.2], 223)
```

This step ensures that the data points to test the finished model haven't been used to train the model.

Connect to an Azure Machine Learning workspace

In Azure Machine Learning, a workspace is a class that accepts your Azure subscription and resource information. It also creates a cloud resource to monitor and track your model runs. In this step, you create a workspace object from the existing Azure Machine Learning workspace.

```
from azureml.core import Workspace

# Enter your subscription id, resource group, and workspace name.
subscription_id = "<enter your subscription ID>" #you should be owner or contributor
resource_group = "<enter your resource group>" #you should be owner or contributor
workspace_name = "<enter your workspace name>" #your workspace name

ws = Workspace(workspace_name = workspace_name,
               subscription_id = subscription_id,
               resource_group = resource_group)
```

Convert a DataFrame to an Azure Machine Learning dataset

To submit a remote experiment, convert your dataset into an Azure Machine Learning `TabularDatset` instance. TabularDataset represents data in a tabular format by parsing the provided files.

The following code gets the existing workspace and the default Azure Machine Learning datastore. It then passes the datastore and file locations to the path parameter to create a new `TabularDataset` instance.

```
import pandas
from azureml.core import Dataset

# Get the Azure Machine Learning default datastore
datastore = ws.get_default_datastore()
training_pd = training_data.toPandas().to_csv('training_pd.csv', index=False)

# Convert into an Azure Machine Learning tabular dataset
datastore.upload_files(files = ['training_pd.csv'],
                       target_path = 'train-dataset/tabular/',
                       overwrite = True,
                       show_progress = True)
dataset_training = Dataset.Tabular.from_delimited_files(path = [(datastore, 'train-
dataset/tabular/training_pd.csv')])
```

```
Uploading an estimated of 1 files
Uploading training_pd.csv
Uploaded training_pd.csv, 1 files out of an estimated total of 1
Uploaded 1 files
```

Submit an automated experiment

The following sections walk you through the process of submitting an automated machine learning experiment.

Define training settings

1. To submit an experiment, you need to define the experiment parameter and model settings for training. For the full list of settings, see Configure automated machine learning experiments in Python.

```
import logging

automl_settings = {
    "iteration_timeout_minutes": 10,
    "experiment_timeout_minutes": 30,
    "enable_early_stopping": True,
    "primary_metric": 'r2_score',
    "featurization": 'auto',
    "verbosity": logging.INFO,
    "n_cross_validations": 2}
```

2. Pass the defined training settings as a `kwargs` parameter to an `AutoMLConfig` object. Because you're using Spark, you must also pass the Spark context, which is automatically accessible by the `sc` variable. Additionally, you specify the training data and the type of model, which is regression in this case.

```
from azureml.train.automl import AutoMLConfig

automl_config = AutoMLConfig(task='regression',
                             debug_log='automated_ml_errors.log',
                             training_data = dataset_training,
                             spark_context = sc,
                             model_explainability = False,
                             label_column_name ="fareAmount",**automl_settings)
```

> **NOTE**
> Automated machine learning pre-processing steps become part of the underlying model. These steps include feature normalization, handling missing data, and converting text to numeric. When you're using the model for predictions, the same pre-processing steps applied during training are applied to your input data automatically.

Train the automatic regression model

Next, you create an experiment object in your Azure Machine Learning workspace. An experiment acts as a container for your individual runs.

```
from azureml.core.experiment import Experiment

# Start an experiment in Azure Machine Learning
experiment = Experiment(ws, "aml-synapse-regression")
tags = {"Synapse": "regression"}
local_run = experiment.submit(automl_config, show_output=True, tags = tags)

# Use the get_details function to retrieve the detailed output for the run.
run_details = local_run.get_details()
```

When the experiment has finished, the output returns details about the completed iterations. For each iteration, you see the model type, the run duration, and the training accuracy. The `BEST` field tracks the best-running training score based on your metric type.

```
*********************************************************************************************
DATA GUARDRAILS:

TYPE:         Missing feature values imputation
STATUS:       PASSED
DESCRIPTION:  No feature missing values were detected in the training data.
              Learn more about missing value imputation: https://aka.ms/AutomatedMLFeaturization

*********************************************************************************************

TYPE:         High cardinality feature detection
STATUS:       PASSED
DESCRIPTION:  Your inputs were analyzed, and no high cardinality features were detected.
              Learn more about high cardinality feature handling: https://aka.ms/AutomatedMLFeaturization

*********************************************************************************************

*********************************************************************************************
ITERATION: The iteration being evaluated.
PIPELINE: A summary description of the pipeline being evaluated.
DURATION: Time taken for the current iteration.
METRIC: The result of computing score on the fitted pipeline.
BEST: The best observed score thus far.
*********************************************************************************************

 ITERATION   PIPELINE                                       DURATION      METRIC      BEST
         0   MaxAbsScaler LightGBM                          0:01:06       0.9602    0.9602
         1   MaxAbsScaler LightGBM                          0:01:45       0.7953    0.9602
         2   MaxAbsScaler RandomForest                      0:10:21       0.9155    0.9602
         3   StandardScalerWrapper ElasticNet               0:02:11       0.9009    0.9602
         4   MaxAbsScaler ExtremeRandomTrees                0:01:45       0.8929    0.9602
         5   MaxAbsScaler LightGBM                          0:01:05       0.8226    0.9602
         6   MaxAbsScaler ExtremeRandomTrees                0:07:44       0.9087    0.9602
         7   MaxAbsScaler RandomForest                      0:01:07       0.8733    0.9602
         8   MaxAbsScaler ElasticNet                        0:01:01       0.9005    0.9602
         9   MaxAbsScaler ExtremeRandomTrees                0:00:43          nan    0.9602
        10    VotingEnsemble                                0:04:36       0.9600    0.9602
        11    StackEnsemble                                 0:02:52       0.9600    0.9602
```

> **NOTE**
>
> After you submit the automated machine learning experiment, it runs various iterations and model types. This run typically takes 60 to 90 minutes.

Retrieve the best model

To select the best model from your iterations, use the `get_output` function to return the best run and fitted model. The following code retrieves the best run and fitted model for any logged metric or a particular iteration.

```
# Get best model
best_run, fitted_model = local_run.get_output()
```

Test model accuracy

1. To test the model accuracy, use the best model to run taxi fare predictions on the test dataset. The `predict` function uses the best model and predicts the values of `y` (fare amount) from the validation dataset.

   ```
   # Test best model accuracy
   validation_data_pd = validation_data.toPandas()
   y_test = validation_data_pd.pop("fareAmount").to_frame()
   y_predict = fitted_model.predict(validation_data_pd)
   ```

2. The root-mean-square error is a frequently used measure of the differences between sample values predicted by a model and the values observed. You calculate the root-mean-square error of the results by comparing the `y_test` DataFrame to the values predicted by the model.

   The function `mean_squared_error` takes two arrays and calculates the average squared error between

them. You then take the square root of the result. This metric indicates roughly how far the taxi fare predictions are from the actual fare values.

```
from sklearn.metrics import mean_squared_error
from math import sqrt

# Calculate root-mean-square error
y_actual = y_test.values.flatten().tolist()
rmse = sqrt(mean_squared_error(y_actual, y_predict))

print("Root Mean Square Error:")
print(rmse)
```

```
Root Mean Square Error:
2.309997102577151
```

The root-mean-square error is a good measure of how accurately the model predicts the response. From the results, you see that the model is fairly good at predicting taxi fares from the dataset's features, typically within $2.00.

3. Run the following code to calculate the mean-absolute-percent error. This metric expresses accuracy as a percentage of the error. It does this by calculating an absolute difference between each predicted and actual value and then summing all the differences. Then, it expresses that sum as a percentage of the total of the actual values.

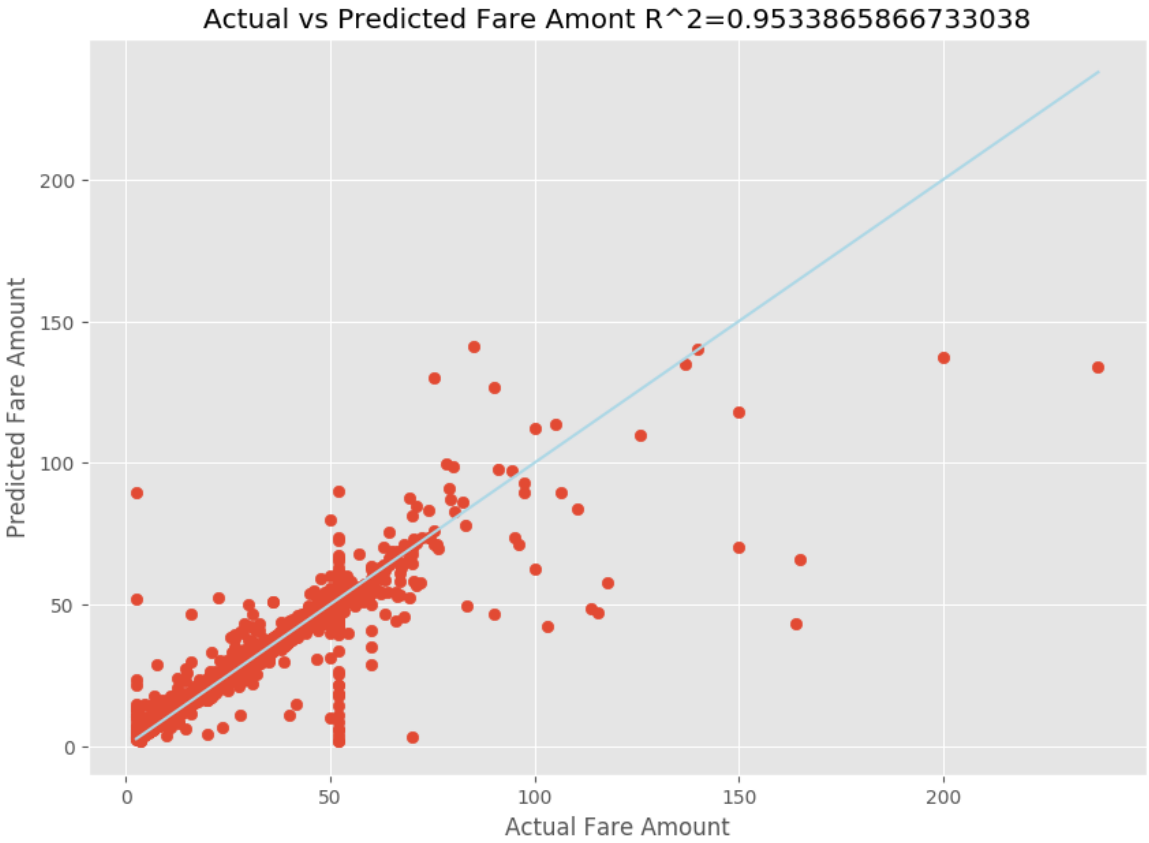
```
# Calculate mean-absolute-percent error and model accuracy
sum_actuals = sum_errors = 0

for actual_val, predict_val in zip(y_actual, y_predict):
    abs_error = actual_val - predict_val
    if abs_error < 0:
        abs_error = abs_error * -1

    sum_errors = sum_errors + abs_error
    sum_actuals = sum_actuals + actual_val

mean_abs_percent_error = sum_errors / sum_actuals

print("Model MAPE:")
print(mean_abs_percent_error)
print()
print("Model Accuracy:")
print(1 - mean_abs_percent_error)
```

```
Model MAPE:
0.03655071038487368

Model Accuracy:
0.9634492896151263
```

From the two prediction accuracy metrics, you see that the model is fairly good at predicting taxi fares from the dataset's features.

4. After fitting a linear regression model, you now need to determine how well the model fits the data. To do this, you plot the actual fare values against the predicted output. In addition, you calculate the R-squared measure to understand how close the data is to the fitted regression line.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score

# Calculate the R2 score by using the predicted and actual fare prices
y_test_actual = y_test["fareAmount"]
r2 = r2_score(y_test_actual, y_predict)

# Plot the actual versus predicted fare amount values
plt.style.use('ggplot')
plt.figure(figsize=(10, 7))
plt.scatter(y_test_actual,y_predict)
plt.plot([np.min(y_test_actual), np.max(y_test_actual)], [np.min(y_test_actual),
np.max(y_test_actual)], color='lightblue')
plt.xlabel("Actual Fare Amount")
plt.ylabel("Predicted Fare Amount")
plt.title("Actual vs Predicted Fare Amount R^2={}".format(r2))
plt.show()
```



From the results, you can see that the R-squared measure accounts for 95 percent of the variance. This is also validated by the actual plot versus the observed plot. The more variance that the regression model accounts for, the closer the data points will fall to the fitted regression line.

Register the model to Azure Machine Learning

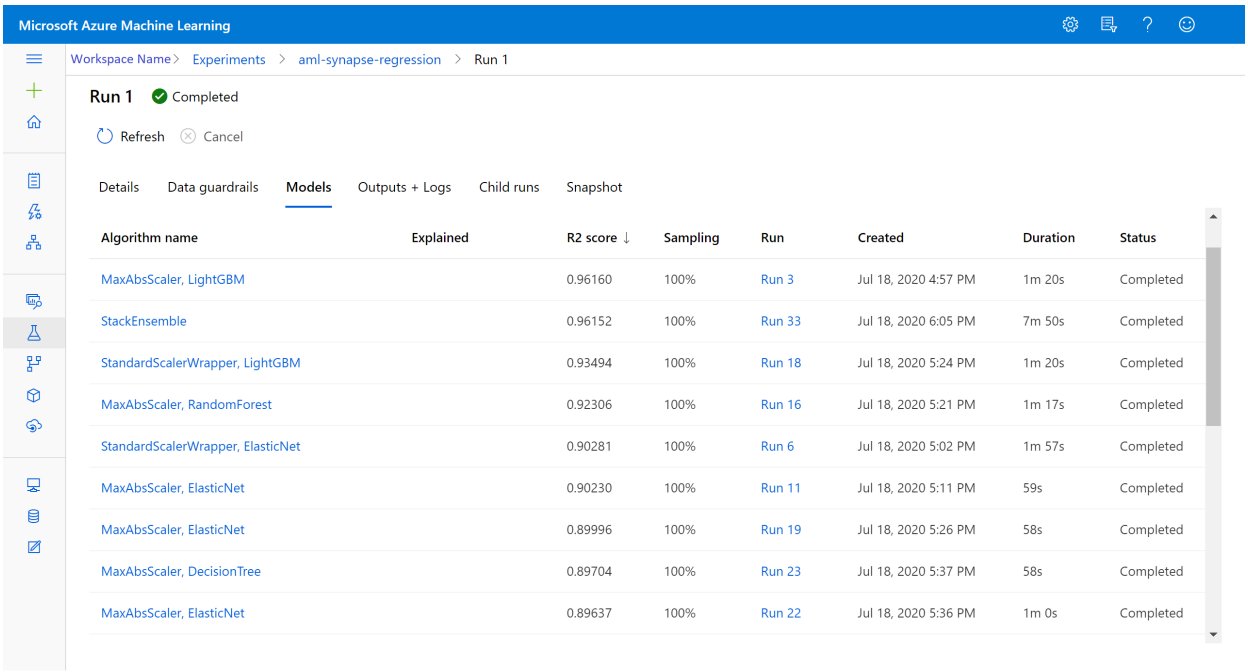
After you've validated your best model, you can register it to Azure Machine Learning. Then, you can download or deploy the registered model and receive all the files that you registered.

```
description = 'My automated ML model'
model_path='outputs/model.pkl'
model = best_run.register_model(model_name = 'NYCYellowTaxiModel', model_path = model_path, description =
description)
print(model.name, model.version)
```

```
NYCYellowTaxiModel 1
```

View results in Azure Machine Learning

You can also access the results of the iterations by going to the experiment in your Azure Machine Learning workspace. Here, you can get additional details on the status of your run, attempted models, and other model metrics.



Next steps

- Azure Synapse Analytics
- Tutorial: Build a machine learning app with Apache Spark MLlib and Azure Synapse Analytics

Tutorial: Distributed Training with Horovod Estimator and PyTorch (Preview)

8/19/2022 • 5 minutes to read • Edit Online

Horovod is a distributed training framework for libraries like TensorFlow and PyTorch. With Horovod, users can scale up an existing training script to run on hundreds of GPUs in just a few lines of code.

Within Azure Synapse Analytics, users can quickly get started with Horovod using the default Apache Spark 3 runtime.For Spark ML pipeline applications using PyTorch, users can use the horovod.spark estimator API. This notebook uses an Apache Spark dataframe to perform distributed training of a distributed neural network (DNN) model on MNIST dataset. This tutorial leverages PyTorch and the Horovod Estimator to run the training process.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Create a GPU-enabled Apache Spark pool in your Azure Synapse Analytics workspace. For details, see Create a GPU-enabled Apache Spark pool in Azure Synapse. For this tutorial, we suggest using the GPU-Large cluster size with 3 nodes.

Configure the Apache Spark session

At the start of the session, we will need to configure a few Apache Spark settings. In most cases, we only needs to set the numExecutors and spark.rapids.memory.gpu.reserve. For very large models, users may also need to configure the `spark.kryoserializer.buffer.max` setting. For Tensorflow models, users will need to set the `spark.executorEnv.TF_FORCE_GPU_ALLOW_GROWTH` to be true.

In the example below, you can see how the Spark configurations can be passed with the `%%configure` command. The detailed meaning of each parameter is explained in the Apache Spark configuration documentation. The values provided below are the suggested, best practice values for Azure Synapse GPU-large pools.

```
%%configure -f
{
    "driverMemory": "30g",
    "driverCores": 4,
    "executorMemory": "60g",
    "executorCores": 12,
    "numExecutors": 3,
    "conf":{
        "spark.rapids.memory.gpu.reserve": "10g",
        "spark.executorEnv.TF_FORCE_GPU_ALLOW_GROWTH": "true",
        "spark.kryoserializer.buffer.max": "2000m"
   }
}
```

For this tutorial, we will use the following configurations:

```
%%configure -f
{
    "numExecutors": 3,
    "conf":{
        "spark.rapids.memory.gpu.reserve": "10g"
   }
}
```

NOTE

When training with Horovod, users should set the Spark configuration for `numExecutors` to be less or equal to the number of nodes.

Import dependencies

In this tutorial, we will leverage PySpark to read and process the dataset. We will then use PyTorch and Horovod to build the distributed neural network (DNN) model and run the training process. To get started, we will need to import the following dependencies:

```
# base libs
import sys
import uuid

# numpy
import numpy as np

# pyspark related
import pyspark
import pyspark.sql.types as T
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf

# pytorch related
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# horovod related
import horovod.spark.torch as hvd
from horovod.spark.common.backend import SparkBackend
from horovod.spark.common.store import Store

# azure related
from azure.synapse.ml.horovodutils import AdlsStore
```

Connect to alternative storage account

We will need the Azure Data Lake Storage (ADLS) account for storing intermediate and model data. If you are using an alternative storage account, be sure to set up the linked service to automatically authenticate and read from the account. In addition, you will need to modify the following properties below: `remote_url`, `account_name`, and `linked_service_name`.

```
num_proc = 3  # equal to numExecutors
batch_size = 128
epochs = 3
lr_single_node = 0.01  # learning rate for single node code

uuid_str = str(uuid.uuid4())  # with uuid, each run will use a new directory
work_dir = '/tmp/' + uuid_str

# create adls store for model training, use your own adls account info
remote_url = "<<ABFS path to storage account>>"
account_name = "<<name of storage account>>"
linked_service_name = "<<name of linked service>>"
sas_token = TokenLibrary.getConnectionString(linked_service_name)
adls_store_path = remote_url + work_dir

store = AdlsStore.create(adls_store_path,
                         storage_options={
                             'account_name': account_name,
                             'sas_token': sas_token
                         },
                         save_runs=True)

print(adls_store_path)
```

Prepare dataset

Next, we will prepare the dataset for training. In this tutorial, we will use the MNIST dataset from Azure Open Datasets.

```
# Initialize SparkSession
spark = SparkSession.builder.getOrCreate()

# Download MNIST dataset from Azure Open Datasets
from azureml.opendatasets import MNIST

mnist = MNIST.get_tabular_dataset()
mnist_df = mnist.to_pandas_dataframe()
mnist_df.info()

# Preprocess dataset
mnist_df['features'] = mnist_df.iloc[:, :784].values.tolist()
mnist_df.drop(mnist_df.iloc[:, :784], inplace=True, axis=1)
mnist_df.head()
```

Process data with Apache Spark

Now, we will create an Apache Spark dataframe. This dataframe will be used with the `HorovodEstimator` for training.

```
# Create Spark DataFrame for training
df = spark.createDataFrame(mnist_df)

# repartition DataFrame for training
train_df = df.repartition(num_proc)

# Train/test split
train_df, test_df = train_df.randomSplit([0.9, 0.1])

# show the dataset
train_df.show()
train_df.count()
```

Define DNN model

Once we have finished processing our dataset, we can now define our PyTorch model. The same code could also be used to train a single-node PyTorch model.

```
# Define the PyTorch model without any Horovod-specific parameters
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = x.float()
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)


model = Net()
optimizer = optim.SGD(model.parameters(),
                      lr=lr_single_node * num_proc,
                      momentum=0.5)  # notice the lr is scaled up
loss = nn.NLLLoss()
```

Train model

Now, we can train a Horovod Spark estimator on top of our Apache Spark dataframe.

```
# Train a Horovod Spark Estimator on the DataFrame
backend = SparkBackend(num_proc=num_proc,
                       stdout=sys.stdout,
                       stderr=sys.stderr,
                       prefix_output_with_timestamp=True)

torch_estimator = hvd.TorchEstimator(
    backend=backend,
    store=store,
    partitions_per_process=1,  # important for GPU training
    model=model,
    optimizer=optimizer,
    loss=lambda input, target: loss(input, target.long()),
    input_shapes=[[-1, 1, 28, 28]],
    feature_cols=['features'],
    label_cols=['label'],
    batch_size=batch_size,
    epochs=epochs,
    validation=0.1,
    verbose=2)

torch_model = torch_estimator.fit(train_df).setOutputCols(['label_prob'])
```

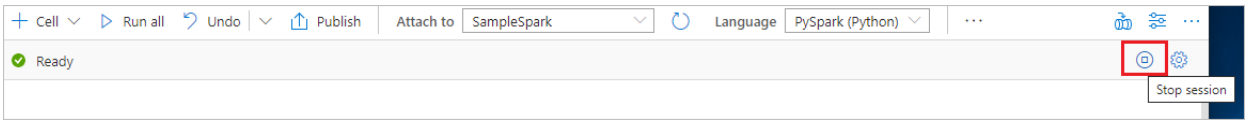
Evaluate trained model

Once the training process has finished, we can then evaluate the model on the test dataset.

```
# Evaluate the model on the held-out test DataFrame
pred_df = torch_model.transform(test_df)

argmax = udf(lambda v: float(np.argmax(v)), returnType=T.DoubleType())
pred_df = pred_df.withColumn('label_pred', argmax(pred_df.label_prob))
evaluator = MulticlassClassificationEvaluator(predictionCol='label_pred',
                                              labelCol='label',
                                              metricName='accuracy')

print('Test accuracy:', evaluator.evaluate(pred_df))
```

Clean up resources

To ensure the Spark instance is shut down, end any connected sessions(notebooks). The pool shuts down when the **idle time** specified in the Apache Spark pool is reached. You can also select **stop session** from the status bar at the upper right of the notebook.



Next steps

- Check out Synapse sample notebooks
- Learn more about GPU-enabled Apache Spark pools

Tutorial: Distributed Training with Horovod Runner and Tensorflow (Preview)

8/19/2022 • 6 minutes to read • Edit Online

Horovod is a distributed training framework for libraries like TensorFlow and PyTorch. With Horovod, users can scale up an existing training script to run on hundreds of GPUs in just a few lines of code.

Within Azure Synapse Analytics, users can quickly get started with Horovod using the default Apache Spark 3 runtime.For Spark ML pipeline applications using Tensorflow, users can use `HorovodRunner`. This notebook uses an Apache Spark dataframe to perform distributed training of a distributed neural network (DNN) model on MNIST dataset. This tutorial leverages Tensorflow and the `HorovodRunner` to run the training process.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Create a GPU-enabled Apache Spark pool in your Azure Synapse Analytics workspace. For details, see Create a GPU-enabled Apache Spark pool in Azure Synapse. For this tutorial, we suggest using the GPU-Large cluster size with 3 nodes.

Configure the Apache Spark session

At the start of the session, we will need to configure a few Apache Spark settings. In most cases, we only needs to set the `numExecutors` and `spark.rapids.memory.gpu.reserve`. For very large models, users may also need to configure the `spark.kryoserializer.buffer.max` setting. For Tensorflow models, users will need to set the `spark.executorEnv.TF_FORCE_GPU_ALLOW_GROWTH` to be true.

In the example below, you can see how the Spark configurations can be passed with the `%%configure` command. The detailed meaning of each parameter is explained in the Apache Spark configuration documentation. The values provided below are the suggested, best practice values for Azure Synapse GPU-large pools.

```
%%configure -f
{
    "driverMemory": "30g",
    "driverCores": 4,
    "executorMemory": "60g",
    "executorCores": 12,
    "numExecutors": 3,
    "conf":{
        "spark.rapids.memory.gpu.reserve": "10g",
        "spark.executorEnv.TF_FORCE_GPU_ALLOW_GROWTH": "true",
        "spark.kryoserializer.buffer.max": "2000m"
   }
}
```

For this tutorial, we will use the following configurations:

```
%%configure -f
{
    "numExecutors": 3,
    "conf":{
        "spark.rapids.memory.gpu.reserve": "10g",
        "spark.executorEnv.TF_FORCE_GPU_ALLOW_GROWTH": "true"
   }
}
```

NOTE

When training with Horovod, users should set the Spark configuration for `numExecutors` to be less or equal to the number of nodes.

Setup primary storage account

We will need the Azure Data Lake Storage (ADLS) account for storing intermediate and model data. If you are using an alternative storage account, be sure to set up the linked service to automatically authenticate and read from the account.

In this example, we will read from the primary Azure Synapse Analytics storage account. To do this, you will need to modify the following properties below: `remote_url`.

```
# Specify training parameters
num_proc = 3  # equal to numExecutors
batch_size = 128
epochs = 3
lr_single_node = 0.1  # learning rate for single node code

# configure adls store remote url
remote_url = "<<abfss path to storage account>>
```

Prepare dataset

Next, we will prepare the dataset for training. In this tutorial, we will use the MNIST dataset from Azure Open Datasets.

```
def get_dataset(rank=0, size=1):
    # import dependency libs
    from azureml.opendatasets import MNIST
    from sklearn.preprocessing import OneHotEncoder
    import numpy as np

    # Download MNIST dataset from Azure Open Datasets
    mnist = MNIST.get_tabular_dataset()
    mnist_df = mnist.to_pandas_dataframe()

    # Preprocess dataset
    mnist_df['features'] = mnist_df.iloc[:, :784].values.tolist()
    mnist_df.drop(mnist_df.iloc[:, :784], inplace=True, axis=1)

    x = np.array(mnist_df['features'].values.tolist())
    y = np.array(mnist_df['label'].values.tolist()).reshape(-1, 1)

    enc = OneHotEncoder()
    enc.fit(y)
    y = enc.transform(y).toarray()

    (x_train, y_train), (x_test, y_test) = (x[:60000], y[:60000]), (x[60000:],
                                                                    y[60000:])

    # Prepare dataset for distributed training
    x_train = x_train[rank::size]
    y_train = y_train[rank::size]
    x_test = x_test[rank::size]
    y_test = y_test[rank::size]

    # Reshape and Normalize data for model input
    x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
    x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    x_train /= 255.0
    x_test /= 255.0

    return (x_train, y_train), (x_test, y_test)
```

Define DNN model

Once we have finished processing our dataset, we can now define our Tensorflow model. The same code could also be used to train a single-node Tensorflow model.

```
# Define the TensorFlow model without any Horovod-specific parameters
def get_model():
    from tensorflow.keras import models
    from tensorflow.keras import layers

    model = models.Sequential()
    model.add(
        layers.Conv2D(32,
                      kernel_size=(3, 3),
                      activation='relu',
                      input_shape=(28, 28, 1)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(layers.Dropout(0.25))
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(10, activation='softmax'))
    return model
```

Define a training function for a single node

First, we will train our Tensorflow model on the driver node of the Apache Spark pool. Once we have finished the training process, we will evaluate the model and print the loss and accuracy scores.

```
def train(learning_rate=0.1):
    import tensorflow as tf
    from tensorflow import keras

    gpus = tf.config.experimental.list_physical_devices('GPU')
    for gpu in gpus:
        tf.config.experimental.set_memory_growth(gpu, True)

    # Prepare dataset
    (x_train, y_train), (x_test, y_test) = get_dataset()

    # Initialize model
    model = get_model()

    # Specify the optimizer (Adadelta in this example)
    optimizer = keras.optimizers.Adadelta(learning_rate=learning_rate)

    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    model.fit(x_train,
              y_train,
              batch_size=batch_size,
              epochs=epochs,
              verbose=2,
              validation_data=(x_test, y_test))
    return model

# Run the training process on the driver
model = train(learning_rate=lr_single_node)

# Evaluate the single node, trained model
_, (x_test, y_test) = get_dataset()
loss, accuracy = model.evaluate(x_test, y_test, batch_size=128)
print("loss:", loss)
print("accuracy:", accuracy)
```

Migrate to HorovodRunner for distributed training

Next, we will take a look at how the same code could be re-run using `HorovodRunner` for distributed training.

Define training function

To do this, we will first define a training function for `HorovodRunner`.

```
# Define training function for Horovod runner
def train_hvd(learning_rate=0.1):
    # Import base libs
    import tempfile
    import os
    import shutil
    import atexit

    # Import tensorflow modules to each worker
    import tensorflow as tf
    from tensorflow import keras
    import horovod.tensorflow.keras as hvd

    # Initialize Horovod
    hvd.init()

    # Pin GPU to be used to process local rank (one GPU per process)
```

```
    # These steps are skipped on a CPU cluster
    gpus = tf.config.experimental.list_physical_devices('GPU')
    for gpu in gpus:
        tf.config.experimental.set_memory_growth(gpu, True)
    if gpus:
        tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()],
                                                   'GPU')

    # Call the get_dataset function you created, this time with the Horovod rank and size
    (x_train, y_train), (x_test, y_test) = get_dataset(hvd.rank(), hvd.size())

    # Initialize model with random weights
    model = get_model()

    # Adjust learning rate based on number of GPUs
    optimizer = keras.optimizers.Adadelta(learning_rate=learning_rate *
                                          hvd.size())

    # Use the Horovod Distributed Optimizer
    optimizer = hvd.DistributedOptimizer(optimizer)

    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    # Create a callback to broadcast the initial variable states from rank 0 to all other processes.
    # This is required to ensure consistent initialization of all workers when training is started with
random weights or restored from a checkpoint.
    callbacks = [
        hvd.callbacks.BroadcastGlobalVariablesCallback(0),
    ]

    # Model checkpoint location.
    ckpt_dir = tempfile.mkdtemp()
    ckpt_file = os.path.join(ckpt_dir, 'checkpoint.h5')
    atexit.register(lambda: shutil.rmtree(ckpt_dir))

    # Save checkpoints only on worker 0 to prevent conflicts between workers
    if hvd.rank() == 0:
        callbacks.append(
            keras.callbacks.ModelCheckpoint(ckpt_file,
                                            monitor='val_loss',
                                            mode='min',
                                            save_best_only=True))

    model.fit(x_train,
              y_train,
              batch_size=batch_size,
              callbacks=callbacks,
              epochs=epochs,
              verbose=2,
              validation_data=(x_test, y_test))

    # Return model bytes only on worker 0
    if hvd.rank() == 0:
        with open(ckpt_file, 'rb') as f:
            return f.read()
```

Run training

Once we have defined the model, we will run the training process.

```
# Run training
import os
import sys
import horovod.spark


best_model_bytes = \
    horovod.spark.run(train_hvd, args=(lr_single_node, ), num_proc=num_proc,
                      env=os.environ.copy(),
                      stdout=sys.stdout, stderr=sys.stderr, verbose=2,
                      prefix_output_with_timestamp=True)[0]
```

Save checkpoints to ADLS storage

The code below shows how to save the checkpoints to the Azure Data Lake Storage (ADLS) account.

```
import tempfile
import fsspec
import os

local_ckpt_dir = tempfile.mkdtemp()
local_ckpt_file = os.path.join(local_ckpt_dir, 'mnist-ckpt.h5')
adls_ckpt_file = remote_url + local_ckpt_file

with open(local_ckpt_file, 'wb') as f:
    f.write(best_model_bytes)

## Upload local file to ADLS
fs = fsspec.filesystem('abfss')
fs.upload(local_ckpt_file, adls_ckpt_file)

print(adls_ckpt_file)
```

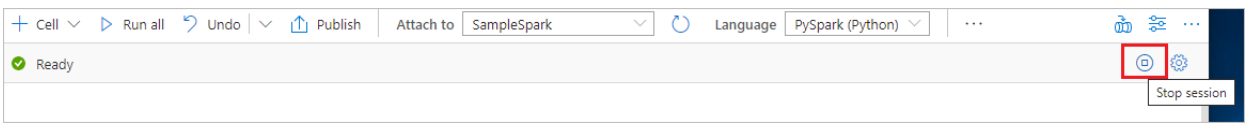
Evaluate Horovod trained model

Once we have finished training our model, we can then take a look at the loss and accuracy for the final model.

```
import tensorflow as tf

hvd_model = tf.keras.models.load_model(local_ckpt_file)

_, (x_test, y_test) = get_dataset()
loss, accuracy = hvd_model.evaluate(x_test, y_test, batch_size=128)
print("loaded model loss and accuracy:", loss, accuracy)
```

Clean up resources

To ensure the Spark instance is shut down, end any connected sessions(notebooks). The pool shuts down when the **idle time** specified in the Apache Spark pool is reached. You can also select **stop session** from the status bar at the upper right of the notebook.



Next steps

- Check out Synapse sample notebooks
- Learn more about GPU-enabled Apache Spark pools

Load data with Petastorm (Preview)

8/19/2022 • 6 minutes to read • Edit Online

Petastorm is an open source data access library which enables single-node or distributed training of deep learning models. This library enables training directly from datasets in Apache Parquet format and datasets that have already been loaded as an Apache Spark DataFrame. Petastorm supports popular training frameworks such as Tensorflow and PyTorch.

For more information about Petastorm, you can visit the Petastorm GitHub page or the Petastorm API documentation.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Create a GPU-enabled Apache Spark pool in your Azure Synapse Analytics workspace. For details, see Create a GPU-enabled Apache Spark pool in Azure Synapse. For this tutorial, we suggest using the GPU-Large cluster size with 3 nodes.

Configure the Apache Spark session

At the start of the session, we will need to configure a few Apache Spark settings. In most cases, we only needs to set the `numExecutors` and `spark.rapids.memory.gpu.reserve`. In the example below, you can see how the Spark configurations can be passed with the `%%configure` command. The detailed meaning of each parameter is explained in the Apache Spark configuration documentation.

```
%%configure -f
{
    "numExecutors": 3,
    "conf":{
        "spark.rapids.memory.gpu.reserve": "10g"
   }
}
```

Petastorm write APIs

A dataset created using Petastorm is stored in an Apache Parquet format. On top of a Parquet schema, Petastorm also stores higher-level schema information that makes multidimensional arrays into a native part of a Petastorm dataset.

In the sample below, we create a dataset using PySpark. We write the dataset to an Azure Data Lake Storage Gen2 account.

```
import numpy as np
from pyspark.sql import SparkSession
from pyspark.sql.types import IntegerType

from petastorm.codecs import ScalarCodec, CompressedImageCodec, NdarrayCodec
from petastorm.etl.dataset_metadata import materialize_dataset
from petastorm.unischema import dict_to_spark_row, Unischema, UnischemaField

# The schema defines how the dataset schema looks like
HelloWorldSchema = Unischema('HelloWorldSchema', [
    UnischemaField('id', np.int32, (), ScalarCodec(IntegerType()), False),
    UnischemaField('image1', np.uint8, (128, 256, 3), CompressedImageCodec('png'), False),
    UnischemaField('array_4d', np.uint8, (None, 128, 30, None), NdarrayCodec(), False),
])


def row_generator(x):
    """Returns a single entry in the generated dataset. Return a bunch of random values as an example."""
    return {'id': x,
            'image1': np.random.randint(0, 255, dtype=np.uint8, size=(128, 256, 3)),
            'array_4d': np.random.randint(0, 255, dtype=np.uint8, size=(4, 128, 30, 3))}


def generate_petastorm_dataset(output_url):
    rowgroup_size_mb = 256

    spark = SparkSession.builder.config('spark.driver.memory', '2g').master('local[2]').getOrCreate()
    sc = spark.sparkContext

    # Wrap dataset materialization portion. Will take care of setting up spark environment variables as
    # well as save petastorm specific metadata
    rows_count = 10
    with materialize_dataset(spark, output_url, HelloWorldSchema, rowgroup_size_mb):

        rows_rdd = sc.parallelize(range(rows_count))\
            .map(row_generator)\
            .map(lambda x: dict_to_spark_row(HelloWorldSchema, x))

        spark.createDataFrame(rows_rdd, HelloWorldSchema.as_spark_schema()) \
            .coalesce(10) \
            .write \
            .mode('overwrite') \
            .parquet(output_url)


output_url = 'abfs://container_name@storage_account_url/data_dir' #use your own adls account info
generate_petastorm_dataset(output_url)
```

Petastorm read APIs

Read dataset from a primary storage account

The `petastorm.reader.Reader` class is the main entry point for user code that accesses the data from an ML framework such as Tensorflow or Pytorch. You can read a dataset using the `petastorm.reader.Reader` class and the `petastorm.make_reader` factory method.

In the example below, you can see how you can pass an `abfs` URL protocol.

```
from petastorm import make_reader

#on primary storage associated with the workspace, url can be abbreviated with container path for data
directory
with make_reader('abfs://<container_name>/<data directory path>/') as reader:
    for row in reader:
        print(row)
```

Read dataset from secondary storage account

If you are using an alternative storage account, be sure to set up the linked service to automatically authenticate and read from the account. In addition, you will need to modify the following properties below: `remote_url`, `account_name`, and `linked_service_name`.

```
from petastorm import make_reader

# create sas token for storage account access, use your own adls account info
remote_url = "abfs://container_name@storage_account_url"
account_name = "<<adls account name>>"
linked_service_name = '<<linked service name>>'
TokenLibrary = spark._jvm.com.microsoft.azure.synapse.tokenlibrary.TokenLibrary
sas_token = TokenLibrary.getConnectionString(linked_service_name)

with make_reader('{}/data_directory'.format(remote_url), storage_options = {'sas_token' : sas_token}) as
reader:
    for row in reader:
        print(row)
```

Read dataset in batches

In the example below, you can see how you can pass an `abfs` URL protocol to read data in batches. This example uses the `make_batch_reader` class.

```
from petastorm import make_batch_reader

with make_batch_reader('abfs://<container_name>/<data directory path>/', schema_fields=["value1", "value2"])
as reader:
    for schema_view in reader:
        print("Batched read:\nvalue1: {0} value2: {1}".format(schema_view.value1, schema_view.value2))
```

PyTorch API

To read a Petastorm dataset from PyTorch, you can use the adapter `petastorm.pytorch.DataLoader` class. This allows for custom PyTorch collating functions and transforms to be supplied.

In this example, we will show how Petastorm DataLoader can be used to load a Petastorm dataset with the help of make_reader API. This first section creates the definition of a `Net` class and `train` and `test` function.

```
from __future__ import division, print_function

import argparse
import pyarrow
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import transforms

from petastorm import make_reader, TransformSpec
from petastorm.pytorch import DataLoader
```

```
from pyspark.sql.functions import col

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

def train(model, device, train_loader, log_interval, optimizer, epoch):
    model.train()
    for batch_idx, row in enumerate(train_loader):
        data, target = row['image'].to(device), row['digit'].to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), loss.item()))

def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    count = 0
    with torch.no_grad():
        for row in test_loader:
            data, target = row['image'].to(device), row['digit'].to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.max(1, keepdim=True)[1]  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()
            count += data.shape[0]
    test_loss /= count
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, count, 100. * correct / count))

def _transform_row(mnist_row):
    # For this example, the images are stored as simpler ndarray (28,28), but the
    # training network expects 3-dim images, hence the additional lambda transform.
    transform = transforms.Compose([
        transforms.Lambda(lambda nd: nd.reshape(28, 28, 1)),
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
    ])
    # In addition, the petastorm pytorch DataLoader does not distinguish the notion of
    # data or target transform, but that actually gives the user more flexibility
    # to make the desired partial transform, as shown here.
    result_row = {
        'image': transform(mnist_row['image']),
        'digit': mnist_row['digit']
    }

    return result_row
```

In this example, an Azure Data Lake Storage account is used to store intermediate data. To store this data, you must set up a Linked Service to the storage account and retrieve the following pieces of information: `remote_url`, `account_name`, and `linked_service_name`.

```
from petastorm import make_reader

# create sas token for storage account access, use your own adls account info
remote_url = "abfs://container_name@storage_account_url"
account_name = "<account name>"
linked_service_name = '<linked service name>'
TokenLibrary = spark._jvm.com.microsoft.azure.synapse.tokenlibrary.TokenLibrary
sas_token = TokenLibrary.getConnectionString(linked_service_name)

# Read Petastorm dataset and apply custom PyTorch transformation functions

device = torch.device('cpu') #For GPU, it will be torch.device('cuda'). More details:
https://pytorch.org/docs/stable/tensor_attributes.html#torch-device

model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

loop_epochs = 1
reader_epochs = 1

transform = TransformSpec(_transform_row, removed_fields=['idx'])

for epoch in range(1, loop_epochs + 1):
    with DataLoader(make_reader('{}/train'.format(remote_url), num_epochs=reader_epochs,
transform_spec=transform),batch_size=5) as train_loader:
        train(model, device, train_loader, 10, optimizer, epoch)
    with DataLoader(make_reader('{}/test'.format(remote_url), num_epochs=reader_epochs,
transform_spec=transform), batch_size=5) as test_loader:
        test(model, device, test_loader)
```

Next steps

- Check out Synapse sample notebooks
- Learn more about GPU-enabled Apache Spark pools

Tutorial: Machine learning model scoring wizard (preview) for dedicated SQL pools

8/19/2022 • 4 minutes to read • Edit Online

Learn how to easily enrich your data in dedicated SQL pools with predictive machine learning models. The models that your data scientists create are now easily accessible to data professionals for predictive analytics. A data professional in Azure Synapse Analytics can simply select a model from the Azure Machine Learning model registry for deployment in Azure Synapse SQL pools and launch predictions to enrich the data.

In this tutorial, you'll learn how to:

- Train a predictive machine learning model and register the model in the Azure Machine Learning model registry.
- Use the SQL scoring wizard to launch predictions in a dedicated SQL pool.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Dedicated SQL pool in your Azure Synapse Analytics workspace. For details, see Create a dedicated SQL pool.
- Azure Machine Learning linked service in your Azure Synapse Analytics workspace. For details, see Create an Azure Machine Learning linked service in Azure Synapse.

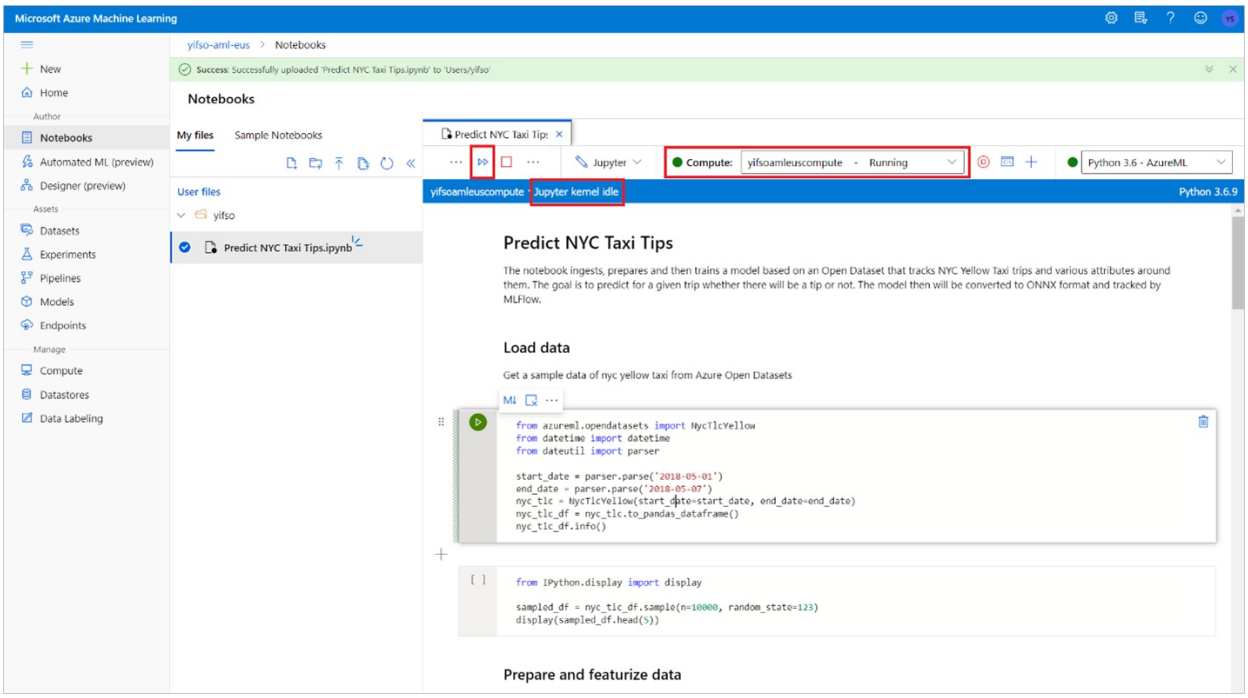
Sign in to the Azure portal

Sign in to the Azure portal.

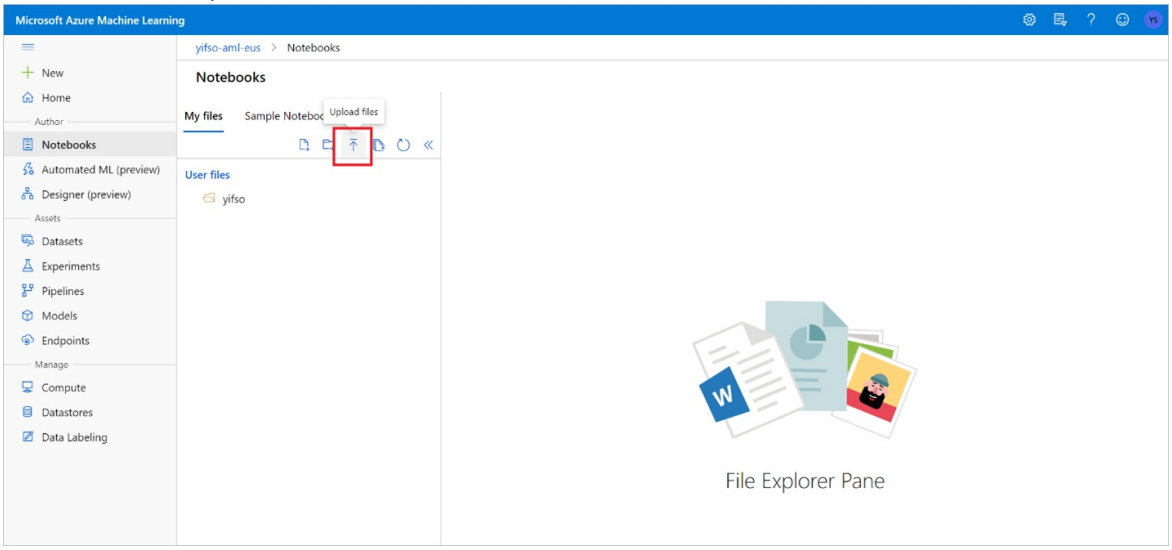
Train a model in Azure Machine Learning

Before you begin, verify that your version of sklearn is 0.20.3.

Before you run all cells in the notebook, check that the compute instance is running.



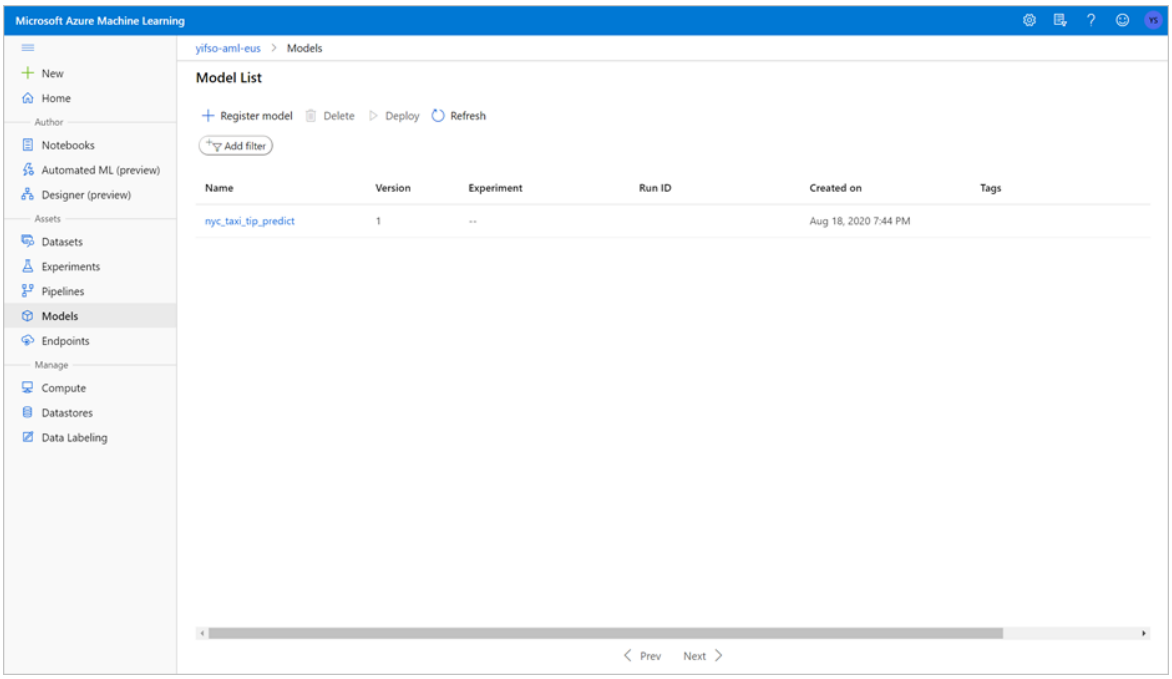
1. Go to your Azure Machine Learning workspace.
2. Download Predict NYC Taxi Tips.ipynb.
3. Open the Azure Machine Learning workspace in Azure Machine Learning Studio.
4. Go to **Notebooks** > **Upload files**. Then select the **Predict NYC Taxi Tips.ipynb** file that you downloaded and upload it.



5. After the notebook is uploaded and opened, select **Run all cells**.

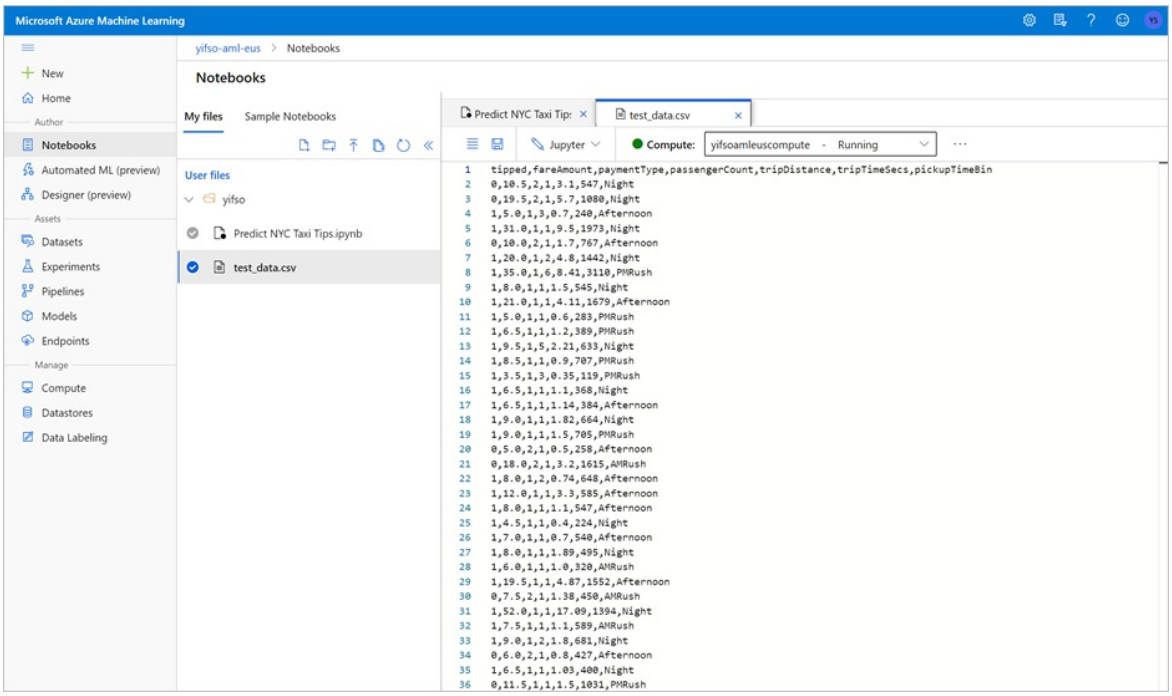
   One of the cells might fail and ask you to authenticate to Azure. Watch for this in the cell outputs, and authenticate in your browser by following the link and entering the code. Then re-run the notebook.

6. The notebook will train an ONNX model and register it with MLflow. Go to **Models** to check that the new model is registered properly.



7. Running the notebook will also export the test data into a CSV file. Download the CSV file to your local system. Later, you'll import the CSV file into a dedicated SQL pool and use the data to test the model.

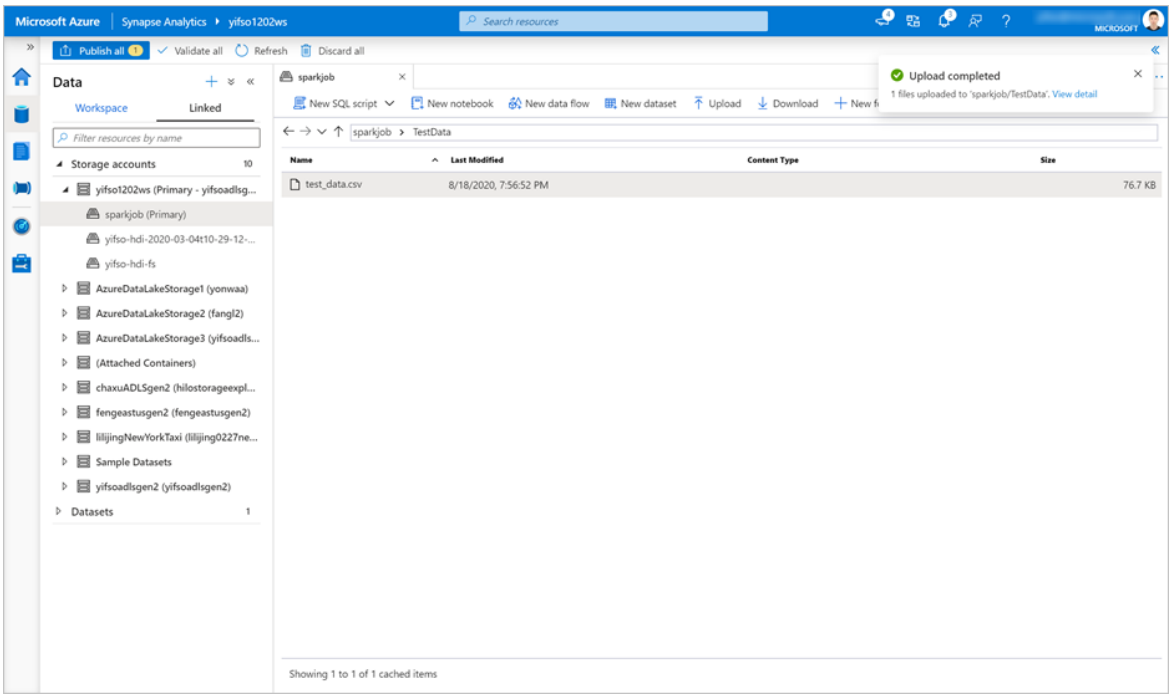
   The CSV file is created in the same folder as your notebook file. Select **Refresh** in File Explorer if you don't see it right away.



Launch predictions with the SQL scoring wizard

1. Open the Azure Synapse workspace with Synapse Studio.

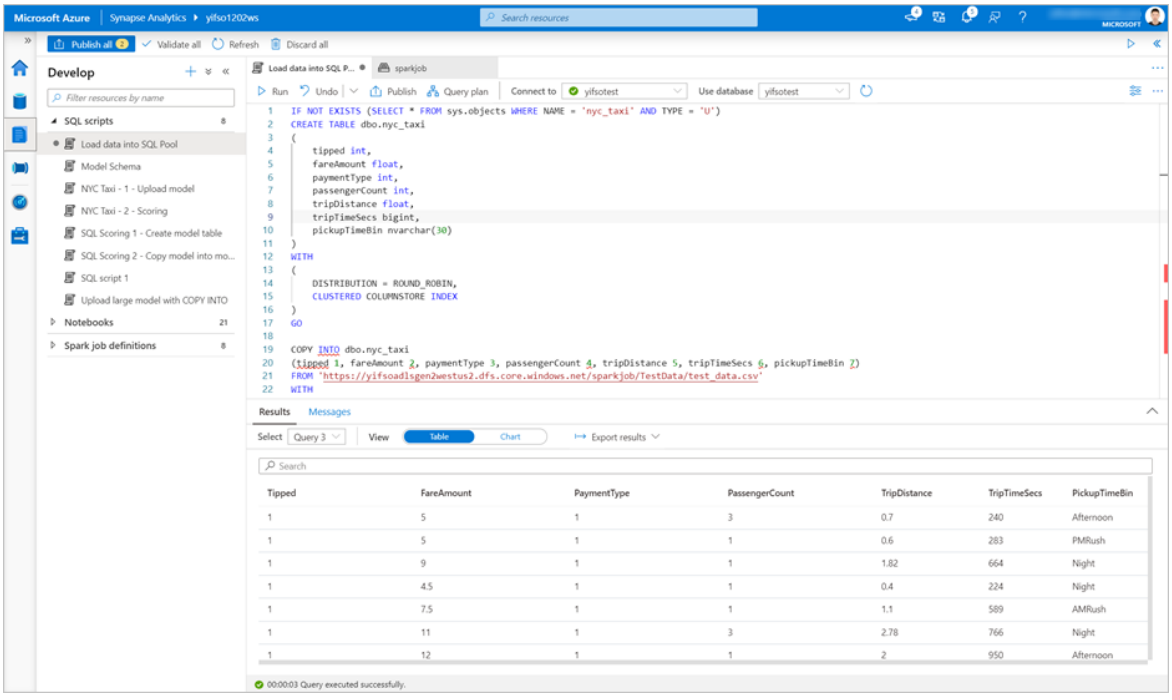
2. Go to **Data** > **Linked** > **Storage Accounts**. Upload `test_data.csv` to the default storage account.



3. Go to **Develop** > **SQL scripts**. Create a new SQL script to load `test_data.csv` into your dedicated SQL pool.

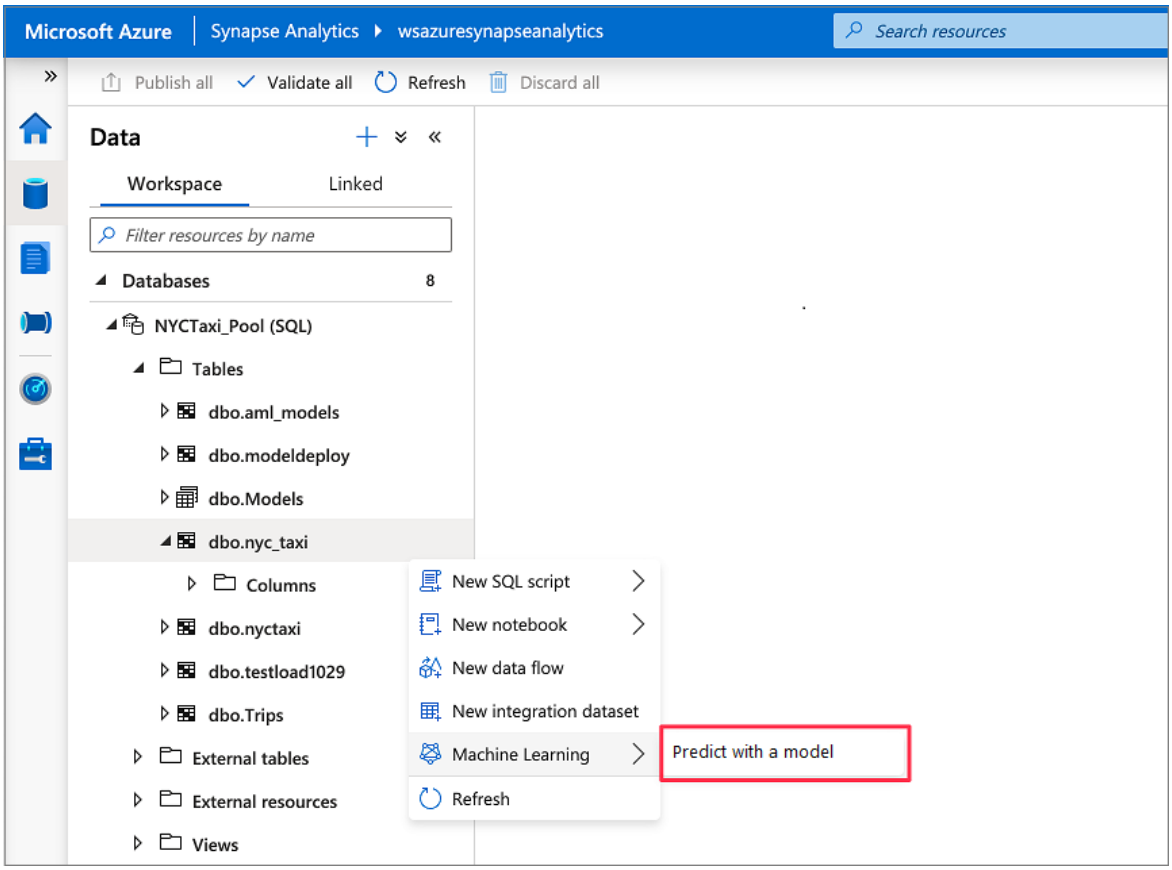
> **NOTE**
>
> Update the file URL in this script before running it.

```
IF NOT EXISTS (SELECT * FROM sys.objects WHERE NAME = 'nyc_taxi' AND TYPE = 'U')
CREATE TABLE dbo.nyc_taxi
(
    tipped int,
    fareAmount float,
    paymentType int,
    passengerCount int,
    tripDistance float,
    tripTimeSecs bigint,
    pickupTimeBin nvarchar(30)
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
GO

COPY INTO dbo.nyc_taxi
(tipped 1, fareAmount 2, paymentType 3, passengerCount 4, tripDistance 5, tripTimeSecs 6,
pickupTimeBin 7)
FROM '<URL to linked storage account>/test_data.csv'
WITH
(
    FILE_TYPE = 'CSV',
    ROWTERMINATOR='0x0A',
    FIELDQUOTE = '"',
    FIELDTERMINATOR = ',',
    FIRSTROW = 2
)
GO

SELECT TOP 100 * FROM nyc_taxi
GO
```



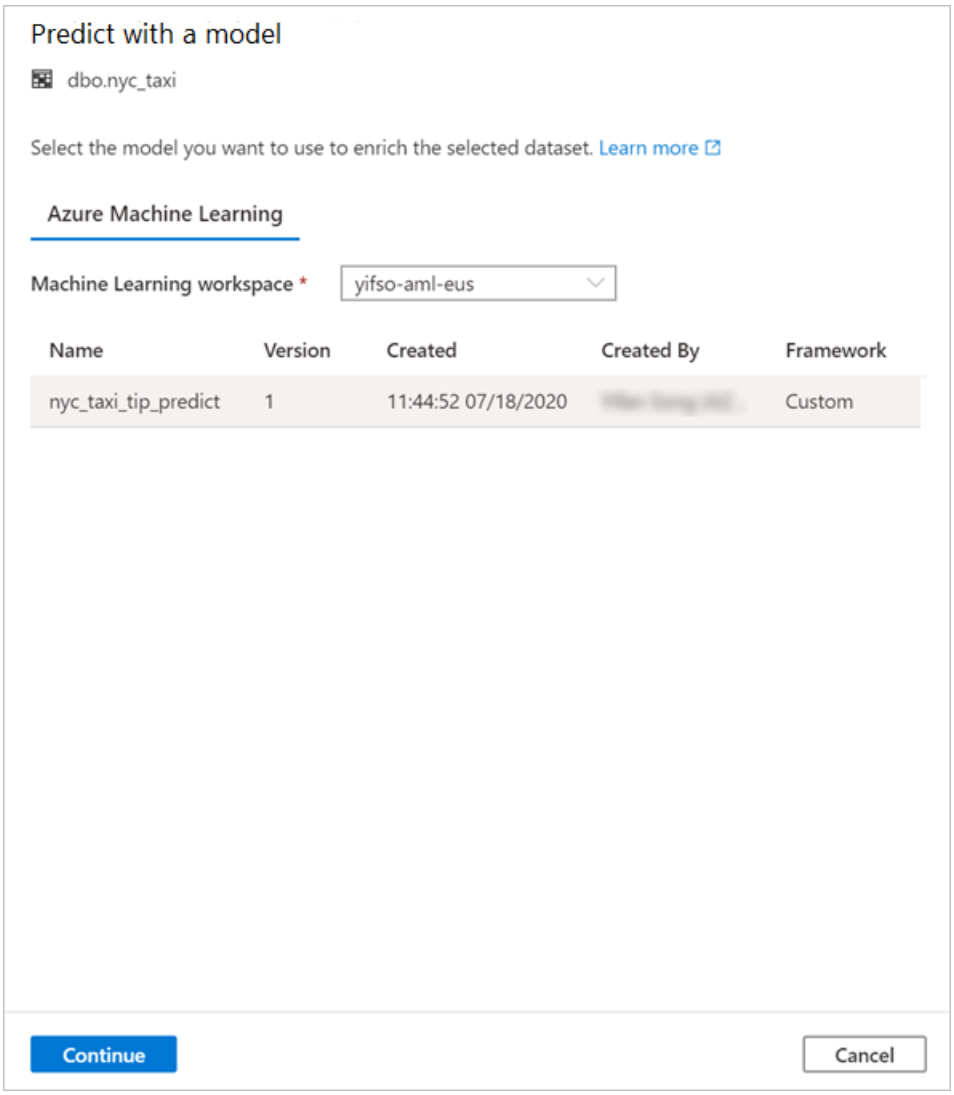
4. Go to **Data** > **Workspace**. Open the SQL scoring wizard by right-clicking the dedicated SQL pool table. Select **Machine Learning** > **Predict with a model**.

> **NOTE**
>
> The machine learning option does not appear unless you have a linked service created for Azure Machine Learning. (See Prerequisites at the beginning of this tutorial.)



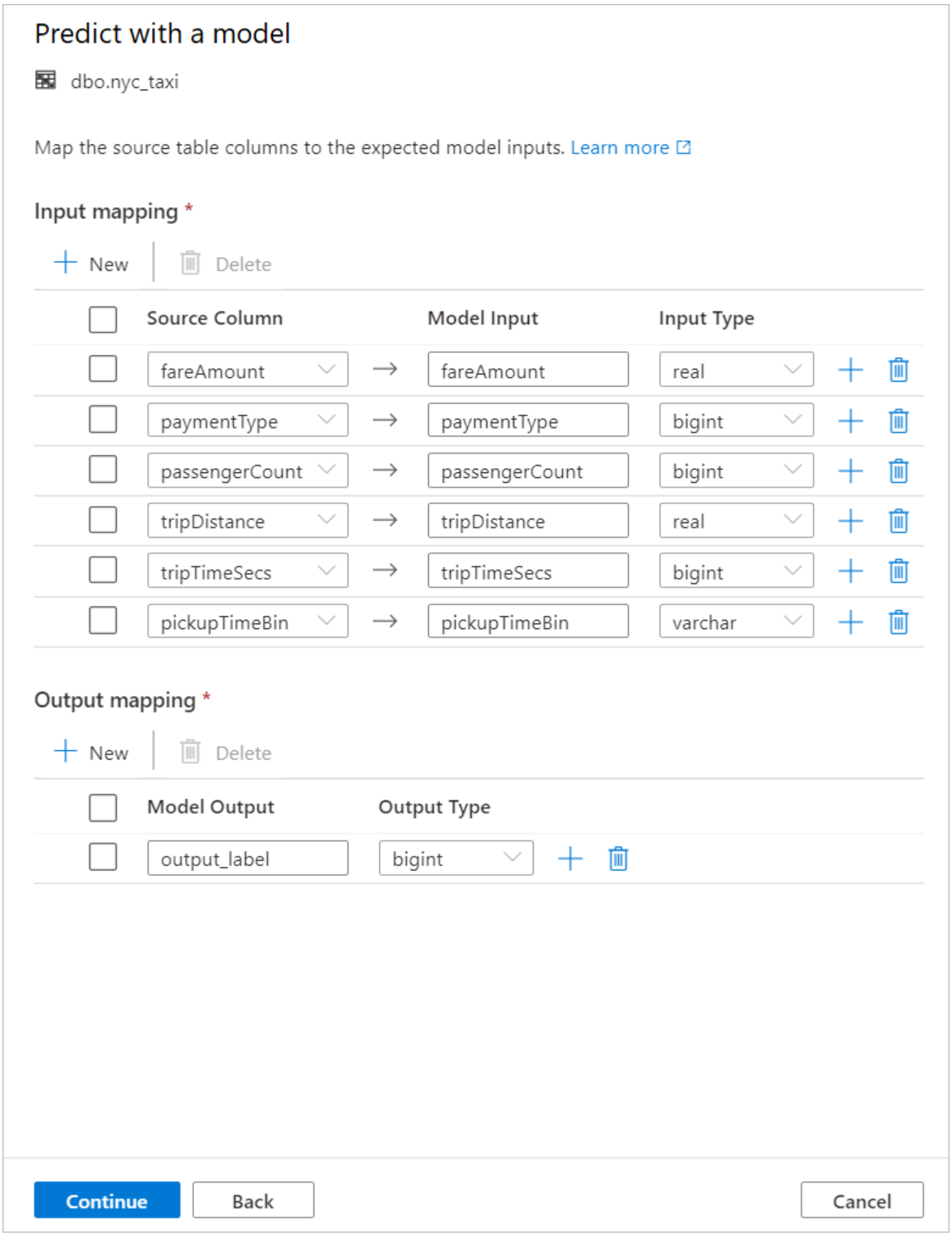
5. Select a linked Azure Machine Learning workspace in the drop-down box. This step loads a list of machine learning models from the model registry of the chosen Azure Machine Learning workspace. Currently, only ONNX models are supported, so this step will display only ONNX models.

6. Select the model that you just trained, and then select **Continue**.



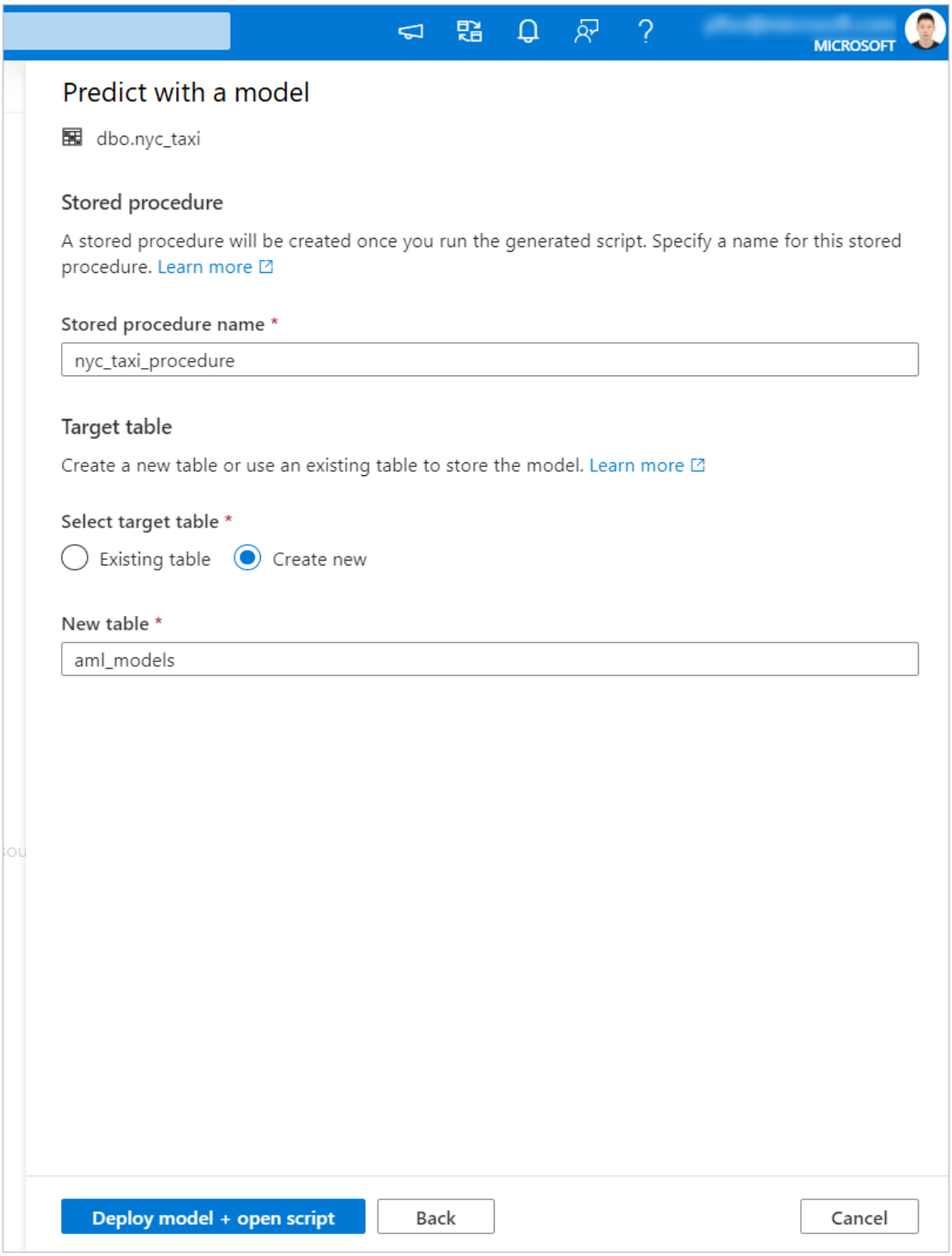
7. Map the table columns to the model inputs and specify the model outputs. If the model is saved in the MLflow format and the model signature is populated, the mapping will be done automatically for you by using a logic based on the similarity of names. The interface also supports manual mapping.

   Select **Continue**.

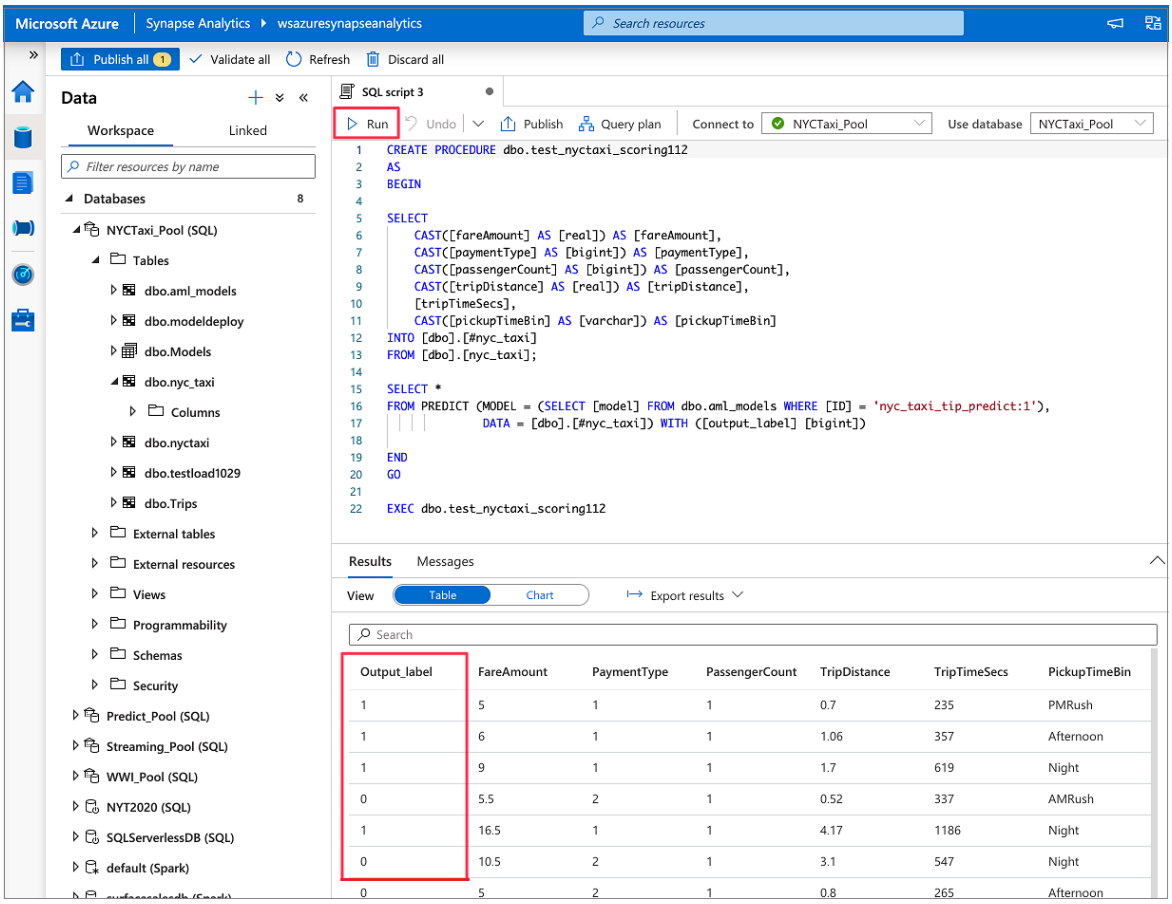


8. The generated T-SQL code is wrapped inside a stored procedure. This is why you need to provide a stored procedure name. The model binary, including metadata (version, description, and other information), will be physically copied from Azure Machine Learning to a dedicated SQL pool table. So you need to specify which table to save the model in.

   You can choose either **Existing table** or **Create new**. When you're done, select **Deploy model + open script** to deploy the model and generate a T-SQL prediction script.



9. After the script is generated, select **Run** to execute the scoring and get predictions.



Next steps

- Quickstart: Create a new Azure Machine Learning linked service in Azure Synapse
- Machine Learning capabilities in Azure Synapse Analytics

Tutorial: Score machine learning models with PREDICT in serverless Apache Spark pools

8/19/2022 • 8 minutes to read • Edit Online

Learn how to use PREDICT functionality in serverless Apache Spark pools in Azure Synapse Analytics for score prediction. You can use a trained model registered in Azure Machine Learning (AML) or in the default Azure Data Lake Storage (ADLS) in your Synapse workspace.

PREDICT in a Synapse PySpark notebook provides you the capability to score machine learning models using the SQL language, user defined functions (UDF), or Transformers. With PREDICT, you can bring your existing machine learning models trained outside Synapse and registered in Azure Data Lake Storage Gen2 or Azure Machine Learning, to score historical data within the secure boundaries of Azure Synapse Analytics. The PREDICT function takes a model and data as inputs. This feature eliminates the step of moving valuable data outside of Synapse for scoring. The goal is to empower model consumers to easily infer machine learning models in Synapse as well as collaborate seamlessly with model producers working with the right framework for their task.

In this tutorial, you'll learn how to:

- Predict scores for data in a serverless Apache Spark pool using machine learning models which are trained outside Synapse and registered in Azure Machine Learning or Azure Data Lake Storage Gen2.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Serverless Apache Spark pool in your Azure Synapse Analytics workspace. For details, see Create a Spark pool in Azure Synapse.
- Azure Machine Learning workspace is needed if you want to train or register model in Azure Machine Learning. For details, see Manage Azure Machine Learning workspaces in the portal or with the Python SDK.
- If your model is registered in Azure Machine Learning then you need a linked service. In Azure Synapse Analytics, a linked service defines your connection information to the service. In this tutorial, you'll add an Azure Synapse Analytics and Azure Machine Learning linked service. To learn more, see Create a new Azure Machine Learning linked service in Synapse.
- The PREDICT functionality requires that you already have a trained model which is either registered in Azure Machine Learning or uploaded in Azure Data Lake Storage Gen2.

NOTE

- PREDICT feature is supported on **Spark3** serverless Apache Spark pool in Azure Synapse Analytics. **Python 3.8** is recommended version for model creation and training.
- PREDICT supports most machine learning models packages in **MLflow** format: **TensorFlow, ONNX, PyTorch, SkLearn and pyfunc** are supported in this preview.
- PREDICT supports **AML and ADLS** model source. Here ADLS account refers to **default Synapse workspace ADLS account**.

Sign in to the Azure portal

Sign in to the Azure portal.

Use PREDICT for MLFLOW packaged models

Make sure all prerequisites are in place before following these steps for using PREDICT.

1. **Import libraries:** Import the following libraries to use PREDICT in spark session.

```
#Import libraries
from pyspark.sql.functions import col, pandas_udf,udf,lit
from azureml.core import Workspace
from azureml.core.authentication import ServicePrincipalAuthentication
import azure.synapse.ml.predict as pcontext
import azure.synapse.ml.predict.utils._logger as synapse_predict_logger
```

2. **Set parameters using variables:** Synapse ADLS data path and model URI need to be set using input variables. You also need to define runtime which is "mlflow" and the data type of model output return. Please note that all data types which are supported in PySpark are supported through PREDICT also.

> **NOTE**
> Before running this script, update it with the URI for ADLS Gen2 data file along with model output return data type and ADLS/AML URI for the model file.

```
#Set input data path
DATA_FILE = "abfss://<filesystemname>@<account name>.dfs.core.windows.net/<file path>"

#Set model URI
    #Set AML URI, if trained model is registered in AML
       AML_MODEL_URI = "<aml model uri>" #In URI ":x" signifies model version in AML. You can
choose which model version you want to run. If ":x" is not provided then by default   latest version
will be picked.

    #Set ADLS URI, if trained model is uploaded in ADLS
       ADLS_MODEL_URI = "abfss://<filesystemname>@<account name>.dfs.core.windows.net/<model
mlflow folder path>"

#Define model return type
RETURN_TYPES = "<data_type>" # for ex: int, float etc. PySpark data types are supported

#Define model runtime. This supports only mlflow
RUNTIME = "mlflow"
```

3. **Ways to authenticate AML workspace:** If the model is stored in the default ADLS account of Synapse workspace, then you do not need any further authentication setup. If the model is registered in Azure Machine Learning, then you can choose either of the following two supported ways of authentication.

> **NOTE**
> Update tenant, client, subscription, resource group, AML workspace and linked service details in this script before running it.

- **Through service principal:** You can use service principal client ID and secret directly to authenticate to AML workspace. Service principal must have "Contributor" access to the AML workspace.

```
    #AML workspace authentication using service principal
    AZURE_TENANT_ID = "<tenant_id>"
    AZURE_CLIENT_ID = "<client_id>"
    AZURE_CLIENT_SECRET = "<client_secret>"

    AML_SUBSCRIPTION_ID = "<subscription_id>"
    AML_RESOURCE_GROUP = "<resource_group_name>"
    AML_WORKSPACE_NAME = "<aml_workspace_name>"

    svc_pr = ServicePrincipalAuthentication(
        tenant_id=AZURE_TENANT_ID,
        service_principal_id=AZURE_CLIENT_ID,
        service_principal_password=AZURE_CLIENT_SECRET
    )

    ws = Workspace(
        workspace_name = AML_WORKSPACE_NAME,
        subscription_id = AML_SUBSCRIPTION_ID,
        resource_group = AML_RESOURCE_GROUP,
        auth=svc_pr
    )
```

- **Through linked service:** You can use linked service to authenticate to AML workspace. Linked service can use "service principal" or Synapse workspace's "Managed Service Identity (MSI)" for authentication. "Service principal" or "Managed Service Identity (MSI)" must have "Contributor" access to the AML workspace.

```
    #AML workspace authentication using linked service
    from notebookutils.mssparkutils import azureML
    ws = azureML.getWorkspace("<linked_service_name>") #   "
<linked_service_name>" is the linked service name, not AML workspace name. Also, linked
service supports MSI and service principal both
```

4. **Enable PREDICT in spark session:** Set the spark configuration `spark.synapse.ml.predict.enabled` to `true` to enable the library.

```
   #Enable SynapseML predict
   spark.conf.set("spark.synapse.ml.predict.enabled","true")
```

5. **Bind model in spark session:** Bind model with required inputs so that the model can be referred in the spark session. Also define alias so that you can use same alias in the PREDICT call.

> **NOTE**
> Update model alias and model uri in this script before running it.

```
   #Bind model within Spark session
    model = pcontext.bind_model(
     return_types=RETURN_TYPES,
     runtime=RUNTIME,
     model_alias="<random_alias_name>", #This alias will be used in PREDICT call to refer  this
model
     model_uri=ADLS_MODEL_URI, #In case of AML, it will be AML_MODEL_URI
     aml_workspace=ws #This is only for AML. In case of ADLS, this parameter can be removed
     ).register()
```

6. **Read data from ADLS:** Read data from ADLS. Create spark dataframe and a view on top of data frame.

> **NOTE**
>
> Update view name in this script before running it.

```
#Read data from ADLS
df = spark.read \
 .format("csv") \
 .option("header", "true") \
 .csv(DATA_FILE,
     inferSchema=True)
df.createOrReplaceTempView('<view_name>')
```

7. **Generate score using PREDICT:** You can call PREDICT three ways, using Spark SQL API, using User define function (UDF), and using Transformer API. Following are examples.

> **NOTE**
>
> Update the model alias name, view name, and comma separated model input column name in this script before running it. Comma separated model input columns are the same as those used while training the model.

```
#Call PREDICT using Spark SQL API

predictions = spark.sql(
               """
                   SELECT PREDICT('<random_alias_name>',
                             <comma_separated_model_input_column_name>) AS predict
                   FROM <view_name>
               """
           ).show()
```

```
#Call PREDICT using user defined function (UDF)

df = df[<comma_separated_model_input_column_name>] # for ex. df["empid","empname"]

df.withColumn("PREDICT",model.udf(lit("<random_alias_name>"),*df.columns)).show()
```

```
#Call PREDICT using Transformer API

columns = [<comma_separated_model_input_column_name>] # for ex. df["empid","empname"]

tranformer = model.create_transformer().setInputCols(columns).setOutputCol("PREDICT")

tranformer.transform(df).show()
```

Sklearn example using PREDICT

1. Import libraries and read the training dataset from ADLS.

```
# Import libraries and read training dataset from ADLS

import fsspec
import pandas
from fsspec.core import split_protocol

adls_account_name = 'xyz' #Provide exact ADLS account name
adls_account_key = 'xyz' #Provide exact ADLS account key

fsspec_handle = fsspec.open('abfs[s]://<container>/<path-to-file>',
account_name=adls_account_name, account_key=adls_account_key)

with fsspec_handle.open() as f:
    train_df = pandas.read_csv(f)
```

2. Train model and generate mlflow artifacts.

```
# Train model and generate mlflow artifacts

import os
import shutil
import mlflow
import json
from mlflow.utils import model_utils
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression


class LinearRegressionModel():
  _ARGS_FILENAME = 'args.json'
  FEATURES_KEY = 'features'
  TARGETS_KEY = 'targets'
  TARGETS_PRED_KEY = 'targets_pred'

  def __init__(self, fit_intercept, nb_input_features=9, nb_output_features=1):
    self.fit_intercept = fit_intercept
    self.nb_input_features = nb_input_features
    self.nb_output_features = nb_output_features

  def get_args(self):
    args = {
        'nb_input_features': self.nb_input_features,
        'nb_output_features': self.nb_output_features,
        'fit_intercept': self.fit_intercept
    }
    return args

  def create_model(self):
    self.model = LinearRegression(fit_intercept=self.fit_intercept)

  def train(self, dataset):

    features = np.stack([sample for sample in iter(
        dataset[LinearRegressionModel.FEATURES_KEY])], axis=0)

    targets = np.stack([sample for sample in iter(
        dataset[LinearRegressionModel.TARGETS_KEY])], axis=0)


    self.model.fit(features, targets)

  def predict(self, dataset):
    features = np.stack([sample for sample in iter(
        dataset[LinearRegressionModel.FEATURES_KEY])], axis=0)
    targets_pred = self.model.predict(features)
```

```
        return targets_pred

    def save(self, path):
        if os.path.exists(path):
            shutil.rmtree(path)

        # save the sklearn model with mlflow
        mlflow.sklearn.save_model(self.model, path)

        # save args
        self._save_args(path)

    def _save_args(self, path):
        args_filename = os.path.join(path, LinearRegressionModel._ARGS_FILENAME)
        with open(args_filename, 'w') as f:
            args = self.get_args()
            json.dump(args, f)


def train(train_df, output_model_path):
    print(f"Start to train LinearRegressionModel.")

    # Initialize input dataset
    dataset = train_df.to_numpy()
    datasets = {}
    datasets['targets'] = dataset[:, -1]
    datasets['features'] = dataset[:, :9]

    # Initialize model class obj
    model_class = LinearRegressionModel(fit_intercept=10)
    with mlflow.start_run(nested=True) as run:
        model_class.create_model()
        model_class.train(datasets)
        model_class.save(output_model_path)
        print(model_class.predict(datasets))


train(train_df, './artifacts/output')
```

3. Store model MLFLOW artifacts in ADLS or register in AML.

```
# Store model MLFLOW artifacts in ADLS

STORAGE_PATH = 'abfs[s]://<container>/<path-to-store-folder>'

protocol, _ = split_protocol(STORAGE_PATH)
print (protocol)

storage_options = {
    'account_name': adls_account_name,
    'account_key': adls_account_key
}
fs = fsspec.filesystem(protocol, **storage_options)
fs.put(
    './artifacts/output',
    STORAGE_PATH,
    recursive=True, overwrite=True)
```

```
    # Register model MLFLOW artifacts in AML

    from azureml.core import Workspace, Model
    from azureml.core.authentication import ServicePrincipalAuthentication

    AZURE_TENANT_ID = "xyz"
    AZURE_CLIENT_ID = "xyz"
    AZURE_CLIENT_SECRET = "xyz"

    AML_SUBSCRIPTION_ID = "xyz"
    AML_RESOURCE_GROUP = "xyz"
    AML_WORKSPACE_NAME = "xyz"

    svc_pr = ServicePrincipalAuthentication(
        tenant_id=AZURE_TENANT_ID,
        service_principal_id=AZURE_CLIENT_ID,
        service_principal_password=AZURE_CLIENT_SECRET
    )

    ws = Workspace(
        workspace_name = AML_WORKSPACE_NAME,
        subscription_id = AML_SUBSCRIPTION_ID,
        resource_group = AML_RESOURCE_GROUP,
        auth=svc_pr
    )

    model = Model.register(
        model_path="./artifacts/output",
        model_name="xyz",
        workspace=ws,
    )
```

4. Set required parameters using variables.

```
    # If using ADLS uploaded model

    import pandas as pd
    from pyspark.sql import SparkSession
    from pyspark.sql.functions import col, pandas_udf,udf,lit
    import azure.synapse.ml.predict as pcontext
    import azure.synapse.ml.predict.utils._logger as synapse_predict_logger

    DATA_FILE = "abfss://xyz@xyz.dfs.core.windows.net/xyz.csv"
    ADLS_MODEL_URI_SKLEARN = "abfss://xyz@xyz.dfs.core.windows.net/mlflow/sklearn/
e2e_linear_regression/"
    RETURN_TYPES = "INT"
    RUNTIME = "mlflow"
```

```
    # If using AML registered model

    from pyspark.sql.functions import col, pandas_udf,udf,lit
    from azureml.core import Workspace
    from azureml.core.authentication import ServicePrincipalAuthentication
    import azure.synapse.ml.predict as pcontext
    import azure.synapse.ml.predict.utils._logger as synapse_predict_logger

    DATA_FILE = "abfss://xyz@xyz.dfs.core.windows.net/xyz.csv"
    AML_MODEL_URI_SKLEARN = "aml://xyz"
    RETURN_TYPES = "INT"
    RUNTIME = "mlflow"
```

5. Enable SynapseML PREDICT functionality in spark session.

```
spark.conf.set("spark.synapse.ml.predict.enabled","true")
```

6. Bind model in spark session.

```
# If using ADLS uploaded model

model = pcontext.bind_model(
 return_types=RETURN_TYPES,
 runtime=RUNTIME,
 model_alias="sklearn_linear_regression",
 model_uri=ADLS_MODEL_URI_SKLEARN,
 ).register()
```

```
# If using AML registered model

model = pcontext.bind_model(
 return_types=RETURN_TYPES,
 runtime=RUNTIME,
 model_alias="sklearn_linear_regression",
 model_uri=AML_MODEL_URI_SKLEARN,
 aml_workspace=ws
 ).register()
```

7. Load test data from ADLS.

```
# Load data from ADLS

df = spark.read \
    .format("csv") \
    .option("header", "true") \
    .csv(DATA_FILE,
        inferSchema=True)
df = df.select(df.columns[:9])
df.createOrReplaceTempView('data')
df.show(10)
```

8. Call PREDICT to generate the score.

```
# Call PREDICT

predictions = spark.sql(
                    """
                        SELECT PREDICT('sklearn_linear_regression', *) AS predict FROM data
                    """
                ).show()
```

Next steps

- Machine Learning capabilities in Azure Synapse Analytics

Cognitive Services in Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

Using pretrained models from Azure Cognitive Services, you can enrich your data with artificial intelligence (AI) in Azure Synapse Analytics.

Azure Cognitive Services are cloud-based services that add cognitive intelligence to your data even if you don't have AI or data science skills. There are a few ways you can use these services with your data in Synapse Analytics:

- The Cognitive Services wizard in Synapse Analytics generates PySpark code in a Synapse notebook that connects to a cognitive service using data in a Spark table. Then, using pretrained machine learning models, the service does the work for you to add AI to your data. Check out Sentiment analysis wizard and Anomaly detection wizard for more details.
- Synapse Machine Learning (SynapseML) allows you to build powerful and highly scalable predictive and analytical models from various Spark data sources. Synapse Spark provide built-in SynapseML libraries including Cognitive Services on Spark. Check out Tutorials for more details.
- Starting from the PySpark code generated by the wizard, or the example SynapseML code provided in the tutorial, you can write your own code to use other cognitive services with your data. See What are Azure Cognitive Services? for more information about available services.

Get started

The tutorial, Pre-requisites for using Cognitive Services in Azure Synapse, walks you through a couple steps you need to perform before using Cognitive Services in Synapse Analytics.

Tutorials

The following tutorials provide complete examples of using Cognitive Services in Synapse Analytics.

- Sentiment analysis with Cognitive Services - Using an example data set of customer comments, you build a Spark table with a column that indicates the sentiment of the comments in each row.
- Anomaly detection with Cognitive Services - Using an example data set of time series data, you build a Spark table with a column that indicates whether the data in each row is an anomaly.
- Build machine learning applications using Microsoft Machine Learning for Apache Spark - This tutorial demonstrates how to use SynapseML to access several models from Cognitive Services.
- Form recognizer with Applied AI Service demonstrates how to use Form Recognizer to analyze your forms and documents, extracts text and data on Azure Synapse Analytics.
- Text Analytics with Cognitive Service shows how to use Text Analytics to analyze unstructured text on Azure Synapse Analytics.
- Translator with Cognitive Service shows how to use Translator to build intelligent, multi-language solutions on Azure Synapse Analytics
- Computer Vision with Cognitive Service demonstrates how to use Computer Vision to analyze images on Azure Synapse Analytics.

Available Cognitive Services APIs

Bing Image Search

API TYPE	SYNAPSEML APIS	COGNITIVE SERVICE APIS (VERSIONS)	DEP VNET SUPPORT
Bing Image Search	BingImageSearch	Images - Visual Search V7.0	Not Supported

Anomaly Detector

API TYPE	SYNAPSEML APIS	COGNITIVE SERVICE APIS (VERSIONS)	DEP VNET SUPPORT
Detect Last Anomaly	DetectLastAnomaly	Detect Last Point V1.0	Supported
Detect Anomalies	DetectAnomalies	Detect Entire Series V1.0	Supported
Simple DetectAnomalies	SimpleDetectAnomalies	Detect Entire Series V1.0	Supported

Computer vision

API TYPE	SYNAPSEML APIS	COGNITIVE SERVICE APIS (VERSIONS)	DEP VNET SUPPORT
OCR	OCR	Recognize Printed Text V2.0	Supported
Recognize Text	RecognizeText	Recognize Text V2.0	Supported
Read Image	ReadImage	Read V3.1	Supported
Generate Thumbnails	GenerateThumbnails	Generate Thumbnail V2.0	Supported
Analyze Image	AnalyzeImage	Analyze Image V2.0	Supported
Recognize Domain Specific Content	RecognizeDomainSpecificContent	Analyze Image By Domain V2.0	Supported
Tag Image	TagImage	Tag Image V2.0	Supported
Describe Image	DescribeImage	Describe Image V2.0	Supported

Translator

API TYPE	SYNAPSEML APIS	COGNITIVE SERVICE APIS (VERSIONS)	DEP VNET SUPPORT
Translate Text	Translate	Translate V3.0	Not Supported
Transliterate Text	Transliterate	Transliterate V3.0	Not Supported
Detect Language	Detect	Detect V3.0	Not Supported
Break Sentence	BreakSentence	Break Sentence V3.0	Not Supported

API TYPE	SYNAPSEML APIS	COGNITIVE SERVICE APIS (VERSIONS)	DEP VNET SUPPORT
Dictionary lookup (alternate translations)	DictionaryLookup	Dictionary Lookup V3.0	Not Supported
Document Translation	DocumentTranslator	Document Translation V1.0	Not Supported

Face

API TYPE	SYNAPSEML APIS	COGNITIVE SERVICE APIS (VERSIONS)	DEP VNET SUPPORT
Detect Face	DetectFace	Detect With Url V1.0	Supported
Find Similar Face	FindSimilarFace	Find Similar V1.0	Supported
Group Faces	GroupFaces	Group V1.0	Supported
Identify Faces	IdentifyFaces	Identify V1.0	Supported
Verify Faces	VerifyFaces	Verify Face To Face V1.0	Supported

Form Recognizer

API TYPE	SYNAPSEML APIS	COGNITIVE SERVICE APIS (VERSIONS)	DEP VNET SUPPORT
Analyze Layout	AnalyzeLayout	Analyze Layout Async V2.1	Supported
Analyze Receipts	AnalyzeReceipts	Analyze Receipt Async V2.1	Supported
Analyze Business Cards	AnalyzeBusinessCards	Analyze Business Card Async V2.1	Supported
Analyze Invoices	AnalyzeInvoices	Analyze Invoice Async V2.1	Supported
Analyze ID Documents	AnalyzeIDDocuments	identification (ID) document model V2.1	Supported
List Custom Models	ListCustomModels	List Custom Models V2.1	Supported
Get Custom Model	GetCustomModel	Get Custom Models V2.1	Supported
Analyze Custom Model	AnalyzeCustomModel	Analyze With Custom Model V2.1	Supported

Speech-to-text

API TYPE	SYNAPSEML APIS	COGNITIVE SERVICE APIS (VERSIONS)	DEP VNET SUPPORT
Speech To Text	SpeechToText	SpeechToText V1.0	Not Supported

API TYPE	SYNAPSEML APIS	COGNITIVE SERVICE APIS (VERSIONS)	DEP VNET SUPPORT
Speech To Text SDK	SpeechToTextSDK	Using Speech SDK Version 1.14.0	Not Supported

Text Analytics

API TYPE	SYNAPSEML APIS	COGNITIVE SERVICE APIS (VERSIONS)	DEP VNET SUPPORT
Text Sentiment V2	TextSentimentV2	Sentiment V2.0	Supported
Language Detector V2	LanguageDetectorV2	Languages V2.0	Supported
Entity Detector V2	EntityDetectorV2	Entities Linking V2.0	Supported
NER V2	NERV2	Entities Recognition General V2.0	Supported
Key Phrase Extractor V2	KeyPhraseExtractorV2	Key Phrases V2.0	Supported
Text Sentiment	TextSentiment	Sentiment V3.1	Supported
Key Phrase Extractor	KeyPhraseExtractor	Key Phrases V3.1	Supported
PII	PII	Entities Recognition Pii V3.1	Supported
NER	NER	Entities Recognition General V3.1	Supported
Language Detector	LanguageDetector	Languages V3.1	Supported
Entity Detector	EntityDetector	Entities Linking V3.1	Supported

Next steps

- Machine Learning capabilities in Azure Synapse Analytics
- What are Cognitive Services?
- Use a sample notebook from the Synapse Analytics gallery

Tutorial: Sentiment analysis with Cognitive Services

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, you'll learn how to easily enrich your data in Azure Synapse Analytics with Azure Cognitive Services. You'll use the Text Analytics capabilities to perform sentiment analysis.

A user in Azure Synapse can simply select a table that contains a text column to enrich with sentiments. These sentiments can be positive, negative, mixed, or neutral. A probability will also be returned.

This tutorial covers:

- Steps for getting a Spark table dataset that contains a text column for sentiment analysis.
- Using a wizard experience in Azure Synapse to enrich data by using Text Analytics in Cognitive Services.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Spark pool in your Azure Synapse Analytics workspace. For details, see Create a Spark pool in Azure Synapse.
- Pre-configuration steps described in the tutorial Configure Cognitive Services in Azure Synapse.

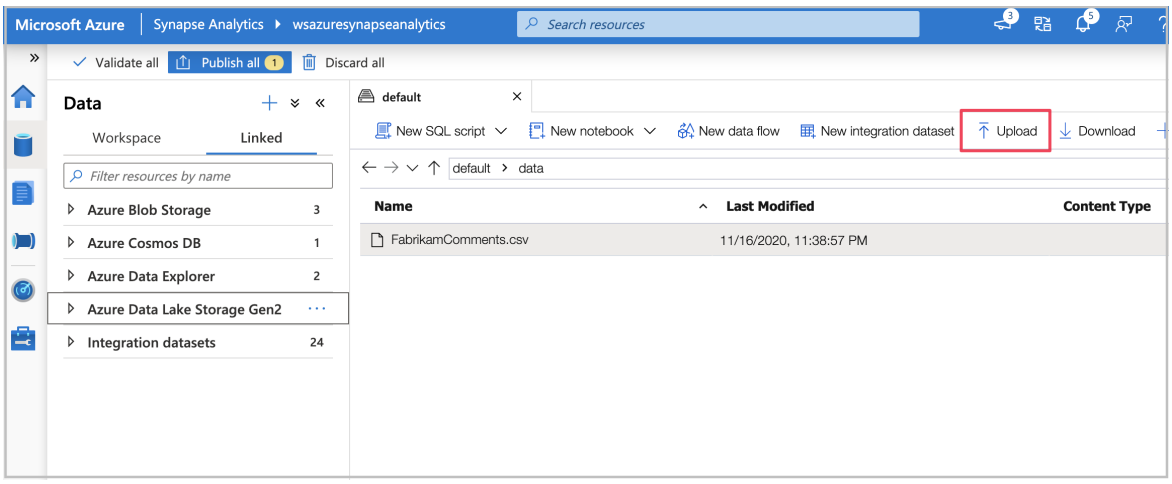
Sign in to the Azure portal

Sign in to the Azure portal.

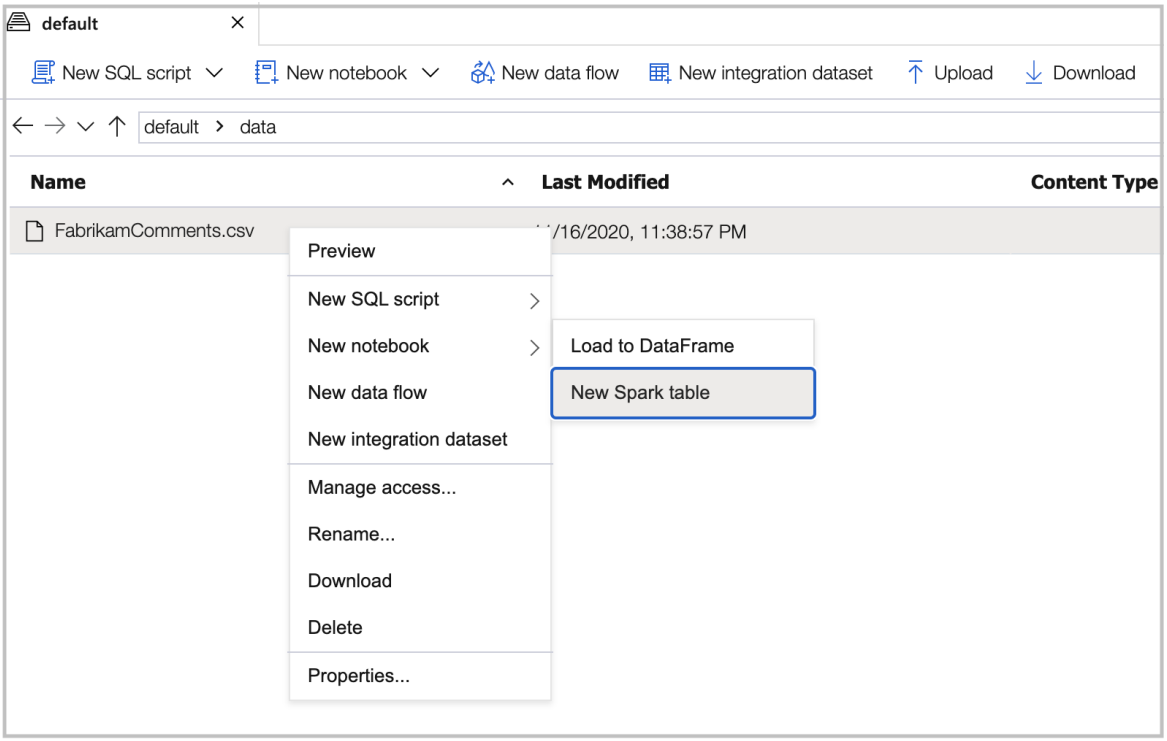
Create a Spark table

You'll need a Spark table for this tutorial.

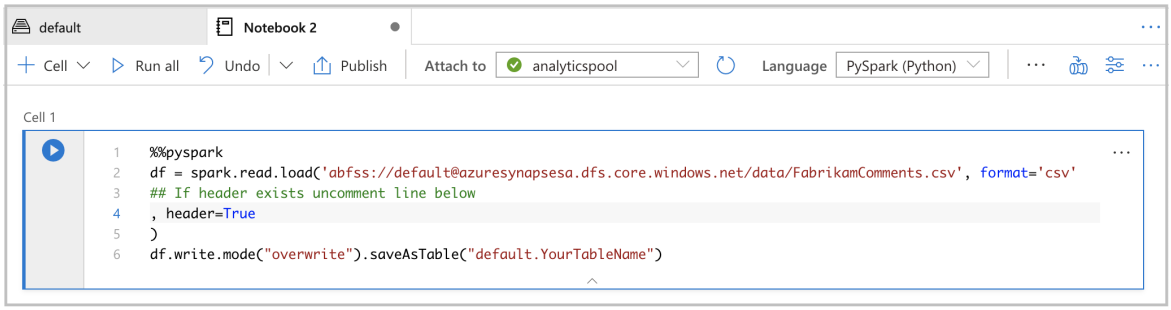
1. Download the FabrikamComments.csv file, which contains a dataset for text analytics.
2. Upload the file to your Azure Synapse storage account in Data Lake Storage Gen2.



3. Create a Spark table from the .csv file by right-clicking the file and selecting **New Notebook** > **Create Spark table**.



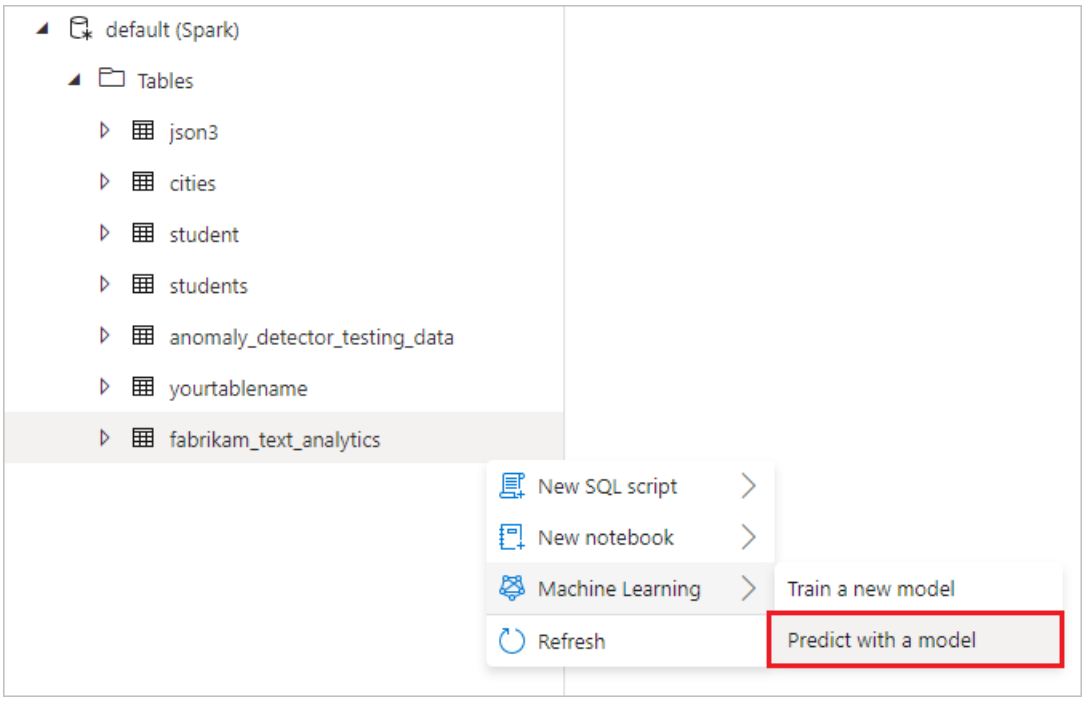
4. Name the table in the code cell and run the notebook on a Spark pool. Remember to set `header=True`.



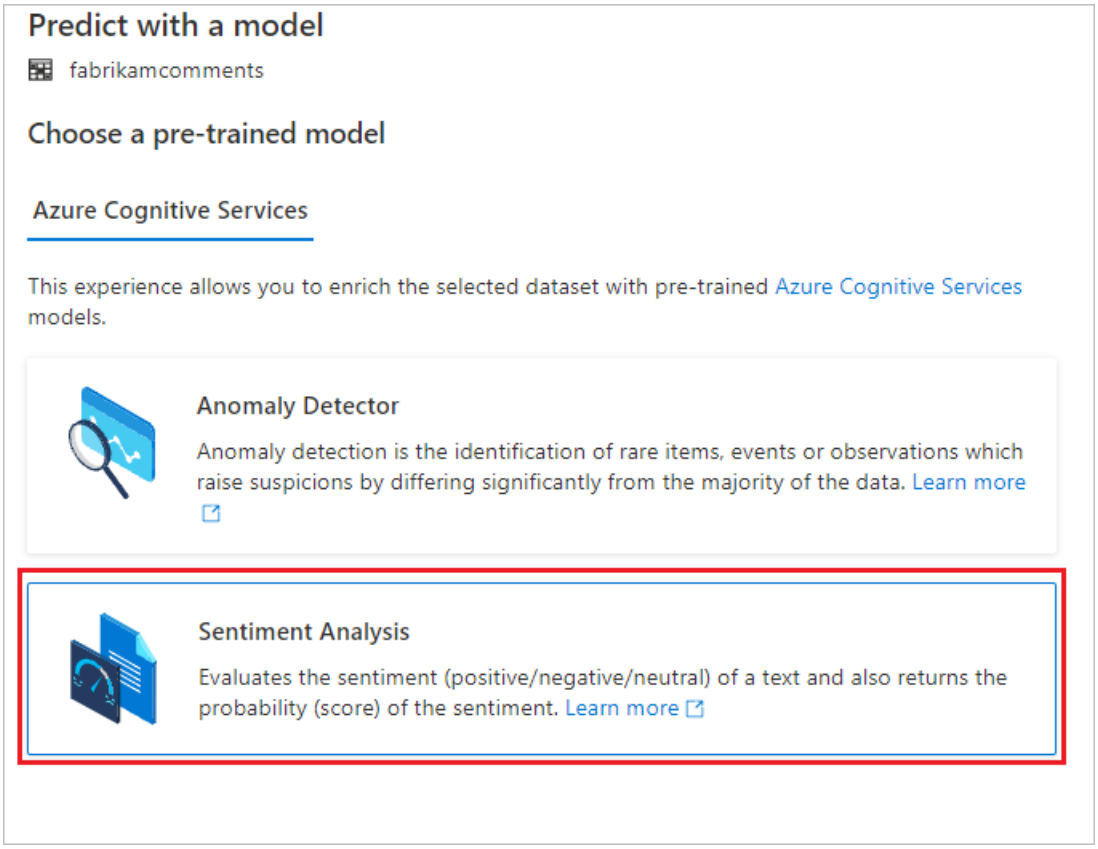

```
%%pyspark
df = spark.read.load('abfss://default@azuresynapsesa.dfs.core.windows.net/data/FabrikamComments.csv',
format='csv'
## Ifa headerexists,uncommentthe linebelow
, header=True
)
df.write.mode("overwrite").saveAsTable("default.YourTableName")
```

Open the Cognitive Services wizard

1. Right-click the Spark table created in the previous procedure. Select **Machine Learning** > **Predict with a model** to open the wizard.



2. A configuration panel appears, and you're asked to select a Cognitive Services model. Select **Sentiment Analysis**.

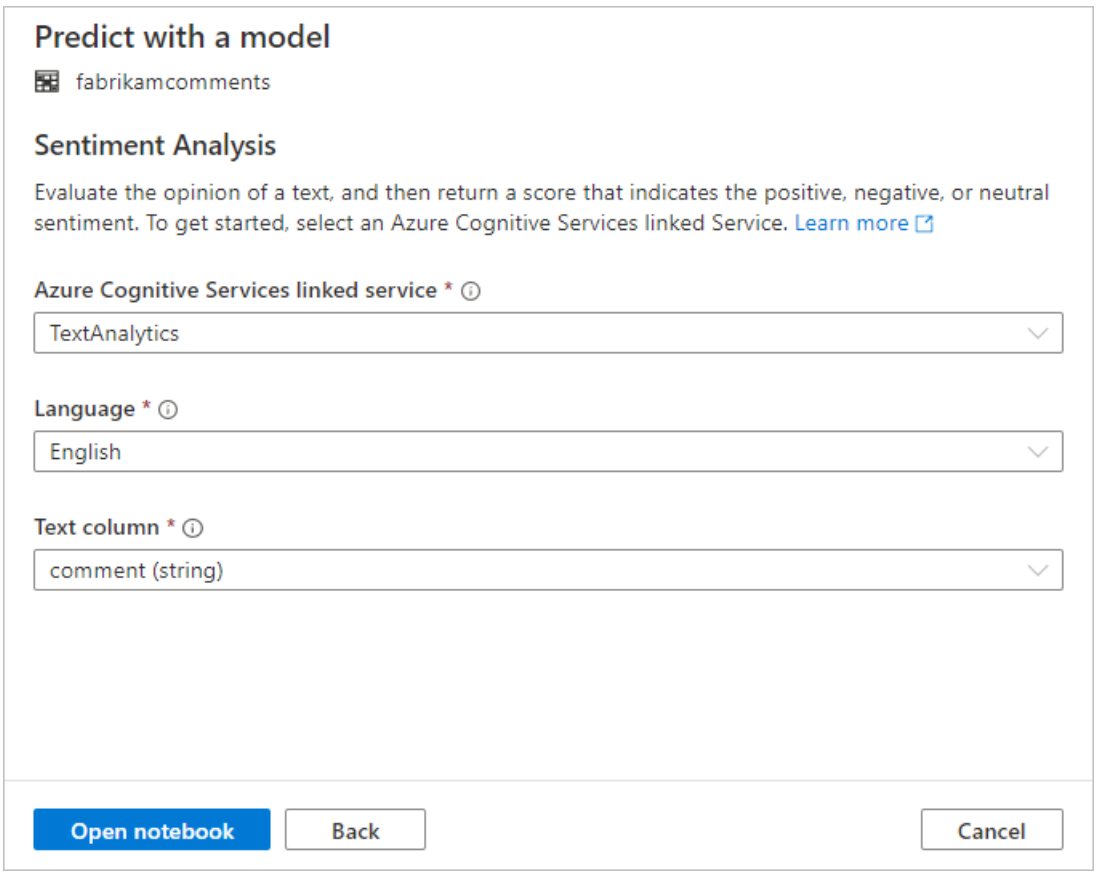


Configure sentiment analysis

Next, configure the sentiment analysis. Select the following details:

- **Azure Cognitive Services linked service**: As part of the prerequisite steps, you created a linked service to your Cognitive Services . Select it here.
- **Language**: Select **English** as the language of the text that you want to perform sentiment analysis on.
- **Text column**: Select **comment (string)** as the text column in your dataset that you want to analyze to determine the sentiment.

When you're done, select **Open notebook**. This generates a notebook for you with PySpark code that performs the sentiment analysis with Azure Cognitive Services.

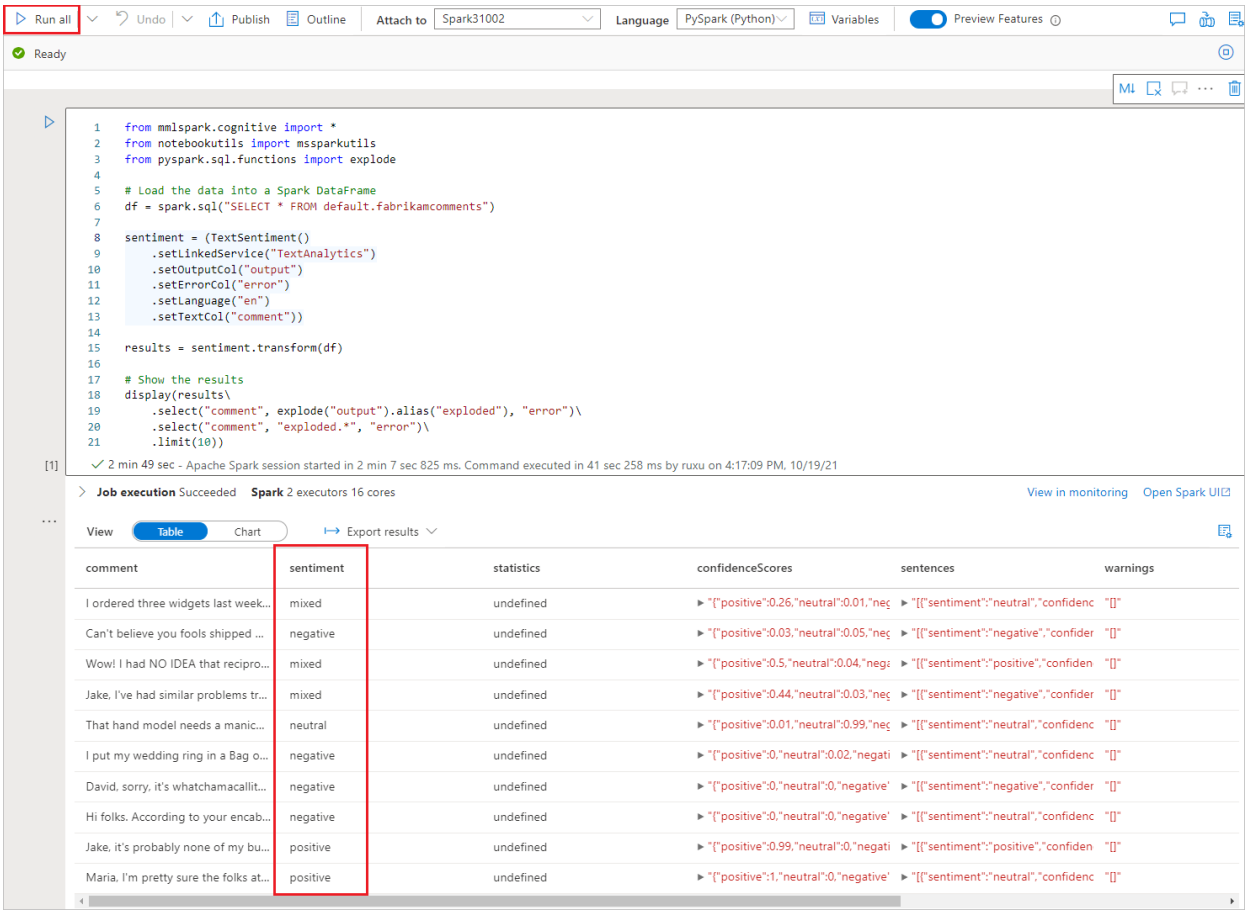


Run the notebook

The notebook that you just opened uses the SynapseML library to connect to Cognitive Services. The Azure Cognitive Services linked service that you provided allow you to securely reference your cognitive service from this experience without revealing any secrets.

You can now run all cells to enrich your data with sentiments. Select **Run all**.

The sentiments are returned as **positive**, **negative**, **neutral**, or **mixed**. You also get probabilities per sentiment. Learn more about sentiment analysis in Cognitive Services.



Next steps

- Tutorial: Anomaly detection with Azure Cognitive Services
- Tutorial: Machine learning model scoring in Azure Synapse dedicated SQL pools
- Machine Learning capabilities in Azure Synapse Analytics

Tutorial: Anomaly detection with Cognitive Services

8/19/2022 • 2 minutes to read • Edit Online

In this tutorial, you'll learn how to easily enrich your data in Azure Synapse Analytics with Azure Cognitive Services. You'll use Anomaly Detector to find anomalies. A user in Azure Synapse can simply select a table to enrich for detection of anomalies.

This tutorial covers:

- Steps for getting a Spark table dataset that contains time series data.
- Use of a wizard experience in Azure Synapse to enrich data by using Anomaly Detector in Cognitive Services.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Spark pool in your Azure Synapse Analytics workspace. For details, see Create a Spark pool in Azure Synapse.
- Completion of the pre-configuration steps in the Configure Cognitive Services in Azure Synapse tutorial.

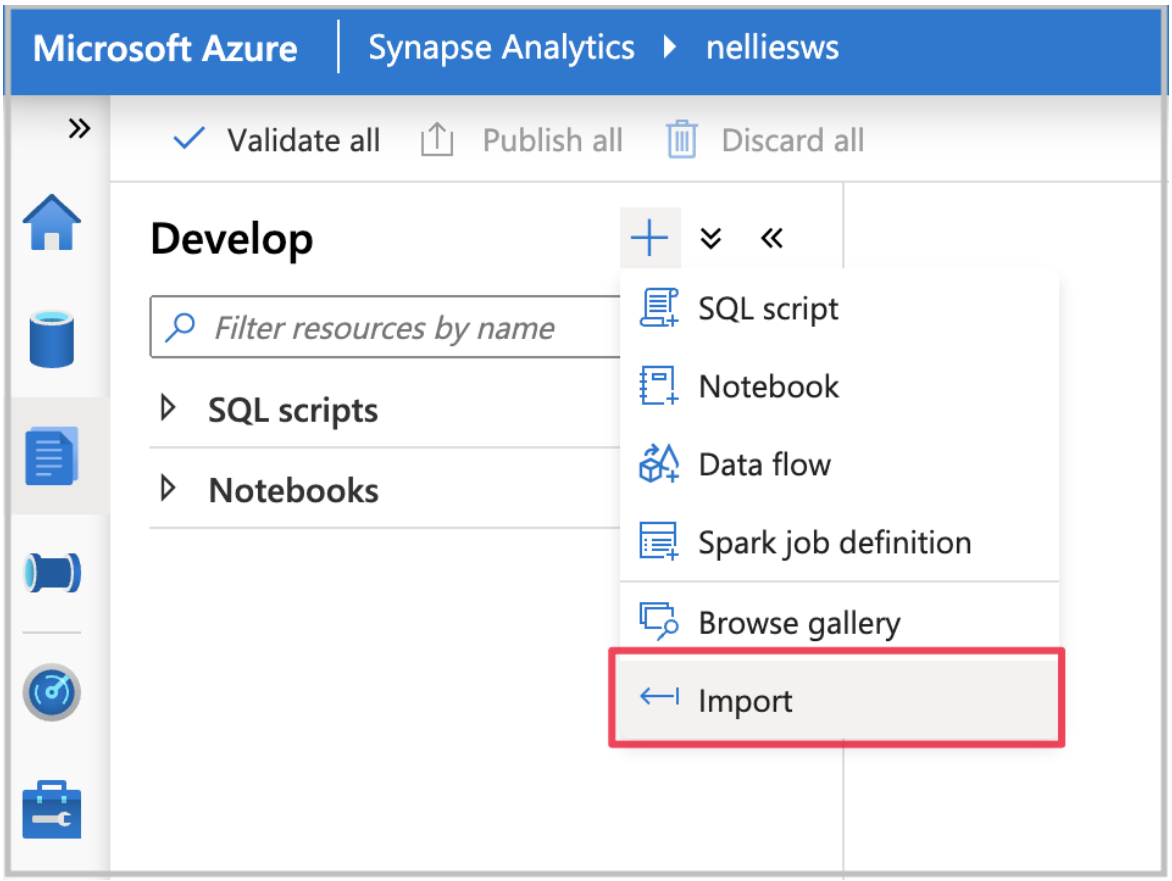
Sign in to the Azure portal

Sign in to the Azure portal.

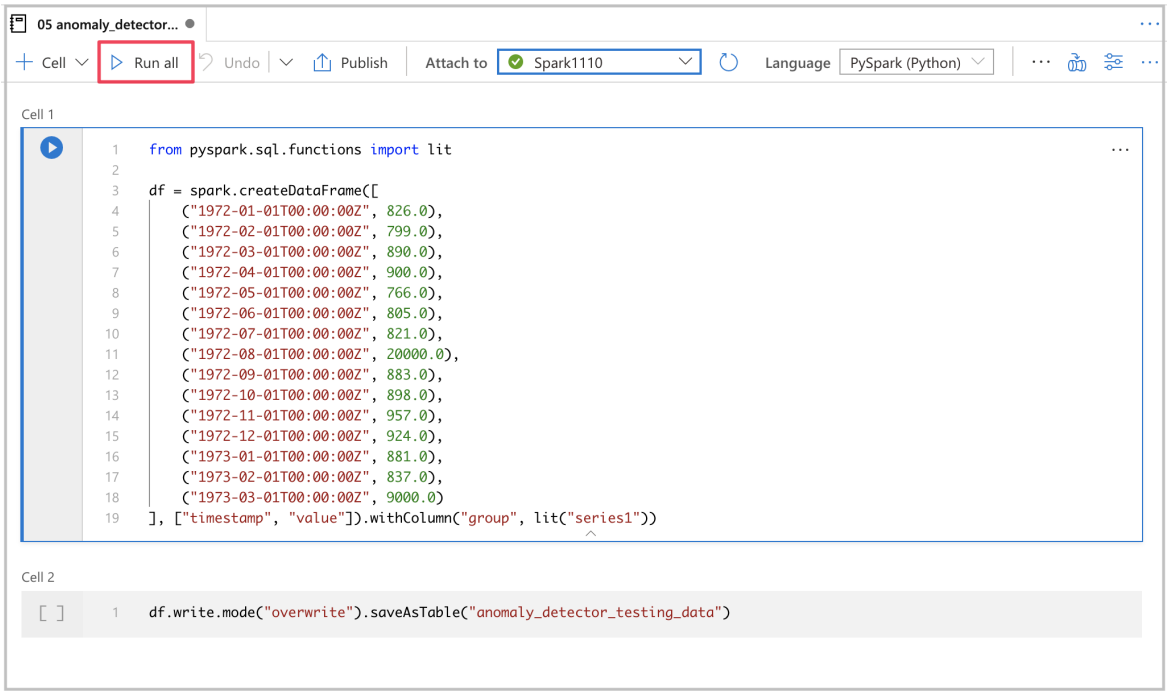
Create a Spark table

You need a Spark table for this tutorial.

1. Download the following notebook file that contains code to generate a Spark table: prepare_anomaly_detector_data.ipynb.
2. Upload the file to your Azure Synapse workspace.



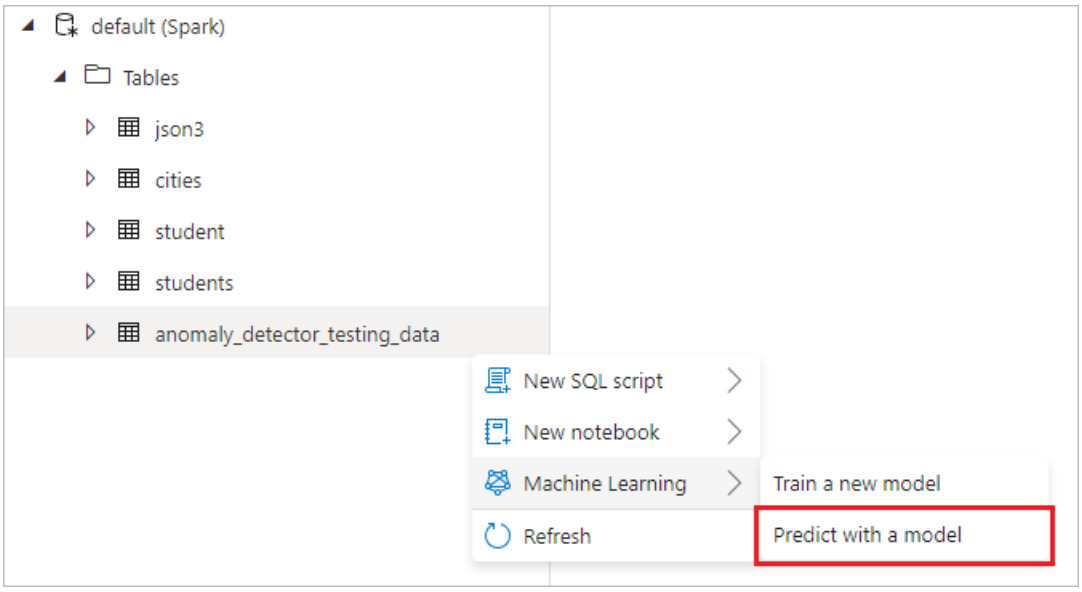
3. Open the notebook file and select **Run All** to run all cells.



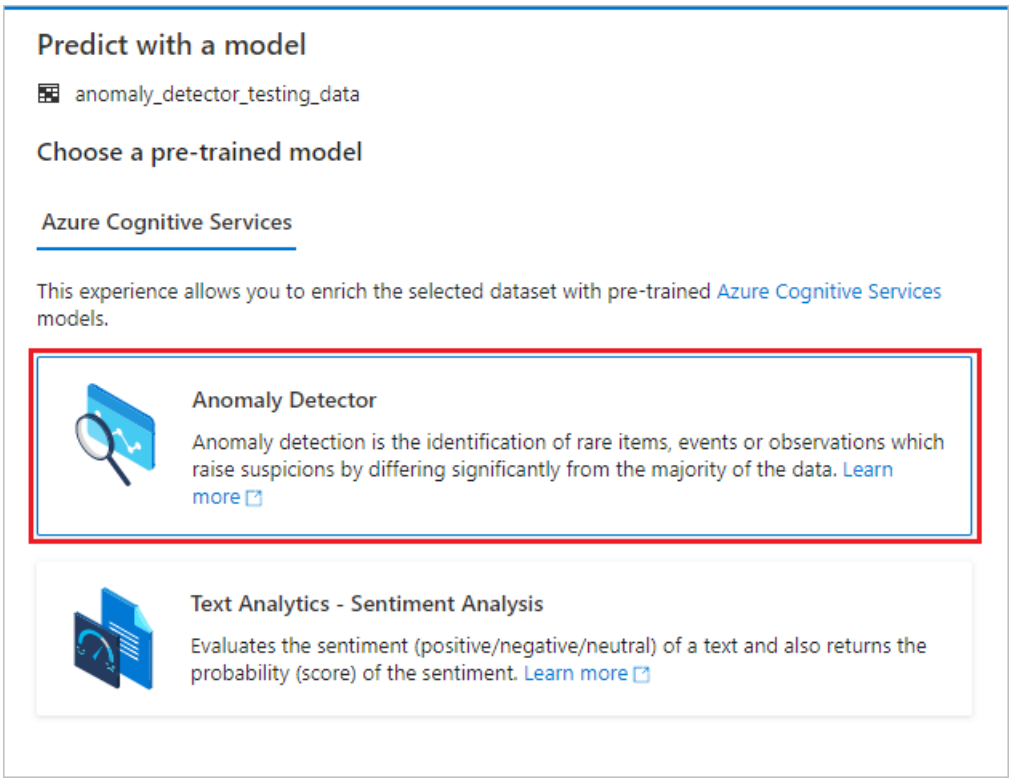
4. A Spark table named **anomaly_detector_testing_data** should now appear in the default Spark database.

Open the Cognitive Services wizard

1. Right-click the Spark table created in the previous step. Select **Machine Learning** > **Predict with a model** to open the wizard.



2. A configuration panel appears, and you're asked to select a Cognitive Services model. Select **Anomaly Detector**.

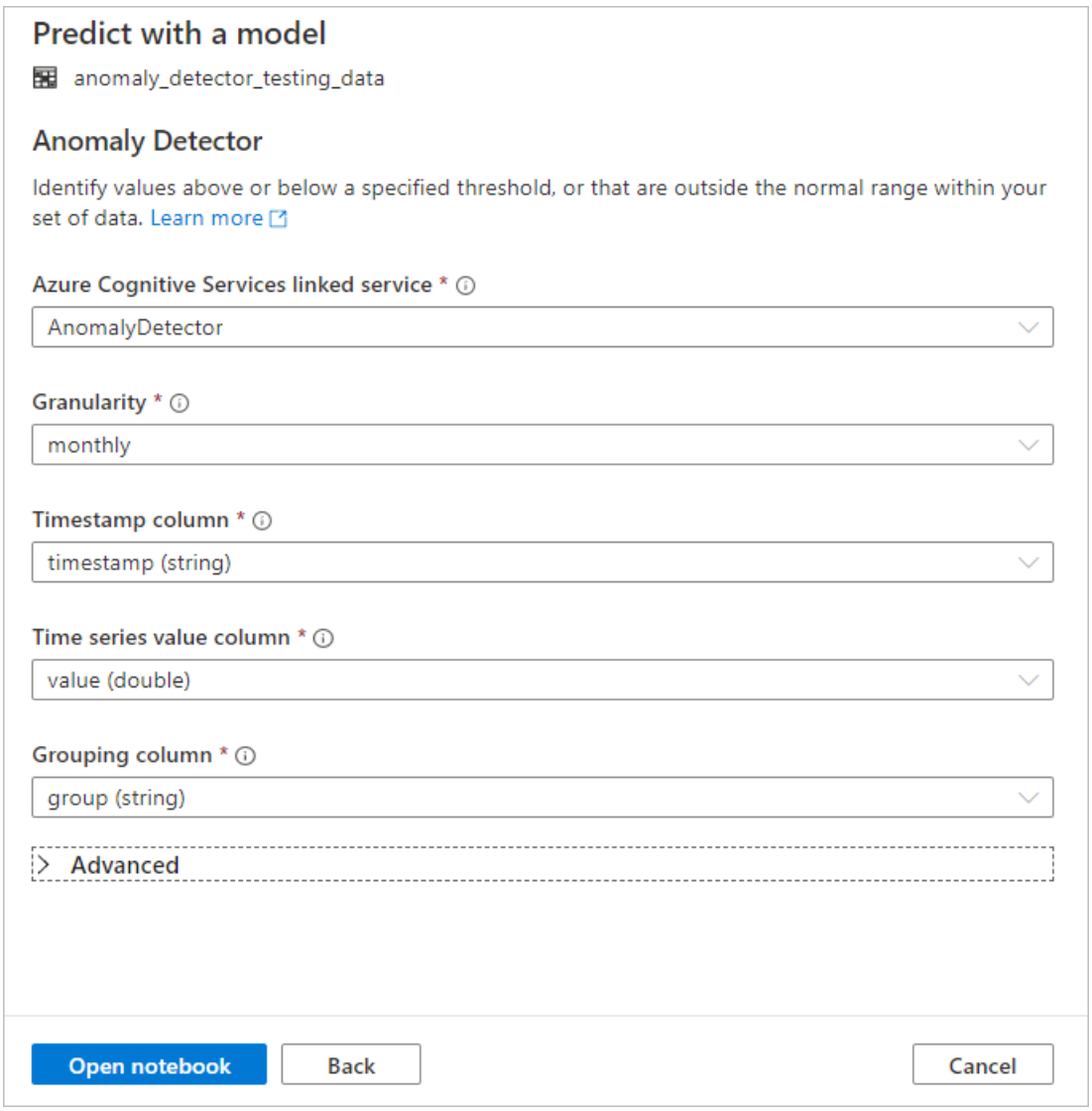


Configure Anomaly Detector

Provide the following details to configure Anomaly Detector:

- **Azure Cognitive Services linked service**: As part of the prerequisite steps, you created a linked service to your Cognitive Services . Select it here.
- **Granularity**: The rate at which your data is sampled. Choose **monthly**.
- **Timestamp column**: The column that represents the time of the series. Choose **timestamp (string)**.
- **Time series value column**: The column that represents the value of the series at the time specified by the Timestamp column. Choose **value (double)**.
- **Grouping column**: The column that groups the series. That is, all rows that have the same value in this column should form one time series. Choose **group (string)**.

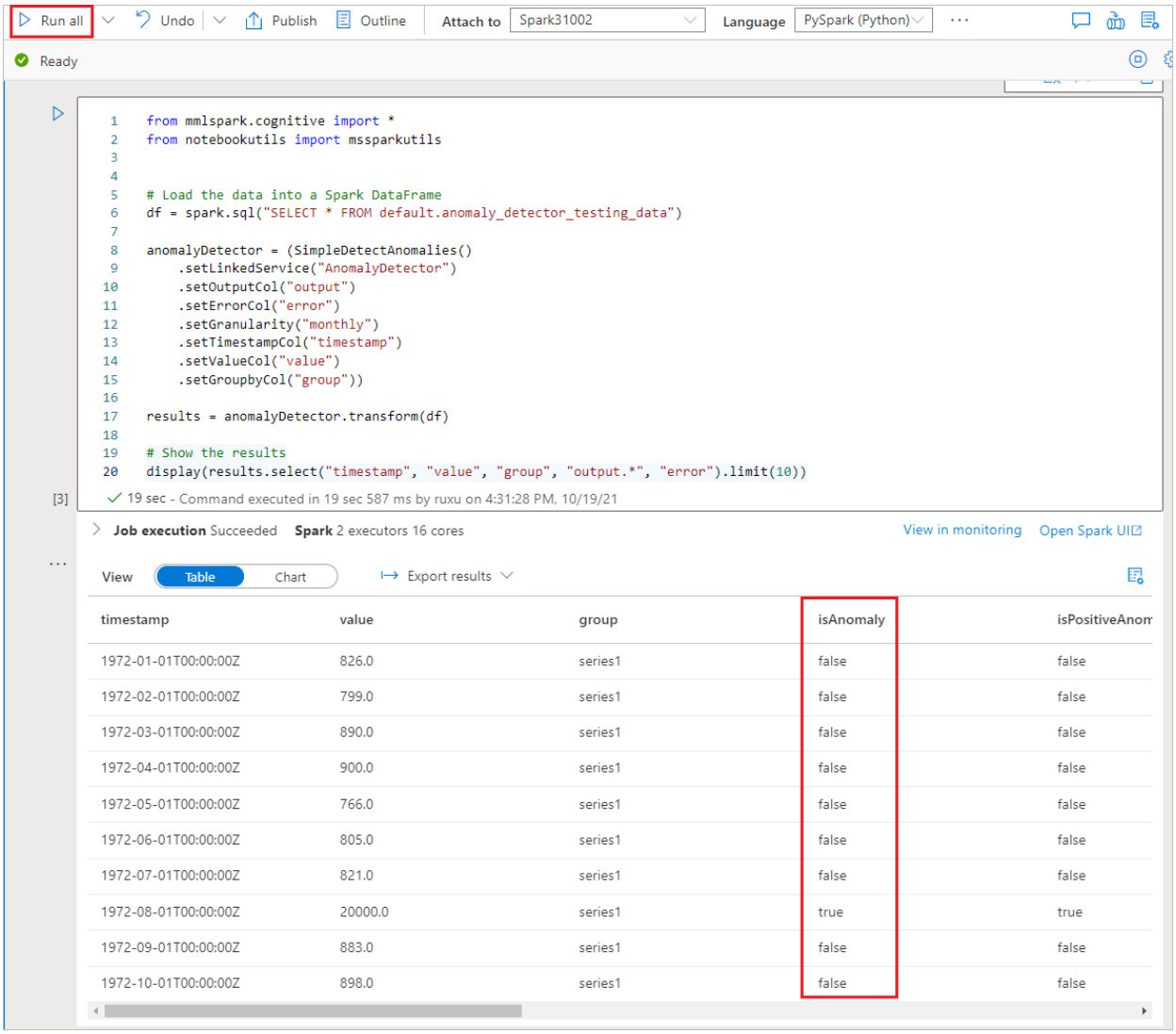
When you're done, select **Open notebook**. This will generate a notebook for you with PySpark code that uses Azure Cognitive Services to detect anomalies.



Run the notebook

The notebook that you just opened uses the SynapseML library to connect to Cognitive Services. The Azure Cognitive Services linked service that you provided allow you to securely reference your cognitive service from this experience without revealing any secrets.

You can now run all cells to perform anomaly detection. Select **Run All**. Learn more about Anomaly Detector in Cognitive Services.



Next steps

- Tutorial: Sentiment analysis with Azure Cognitive Services
- Tutorial: Machine learning model scoring in Azure Synapse dedicated SQL pools
- Machine Learning capabilities in Azure Synapse Analytics

Tutorial: Text Analytics with Cognitive Service

8/19/2022 • 6 minutes to read • Edit Online

Text Analytics is an Azure Cognitive Service that enables you to perform text mining and text analysis with Natural Language Processing (NLP) features. In this tutorial, you'll learn how to use Text Analytics to analyze unstructured text on Azure Synapse Analytics.

This tutorial demonstrates using text analytics with SynapseML to:

- Detect sentiment labels at the sentence or document level
- Identify the language for a given text input
- Recognize entities from a text with links to a well-known knowledge base
- Extract key phases from a text
- Identify different entities in text and categorize them into pre-defined classes or types
- Identify and redact sensitive entities in a given text

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Spark pool in your Azure Synapse Analytics workspace. For details, see Create a Spark pool in Azure Synapse.
- Pre-configuration steps described in the tutorial Configure Cognitive Services in Azure Synapse.

Get started

Open Synapse Studio and create a new notebook. To get started, import SynapseML.

```
import synapse.ml
from synapse.ml.cognitive import *
from pyspark.sql.functions import col
```

Configure text analytics

Use the linked text analytics you configured in the pre-configuration steps .

```
cognitive_service_name = "<Your linked service for text analytics>"
```

Text Sentiment

The Text Sentiment Analysis provides a way for detecting the sentiment labels (such as "negative", "neutral" and "positive") and confidence scores at the sentence and document-level. See the Supported languages in Text Analytics API for the list of enabled languages.

V2

```
# Create a dataframe that's tied to it's column names
df = spark.createDataFrame([
  ("I am so happy today, its sunny!", "en-US"),
  ("I am frustrated by this rush hour traffic", "en-US"),
  ("The cognitive services on spark aint bad", "en-US"),
], ["text", "language"])

# Run the Text Analytics service with options
sentimentv2 = (TextSentimentV2()
    .setLinkedService(linked_service_name)
    .setTextCol("text")
    .setOutputCol("sentiment")
    .setErrorCol("error")
    .setLanguageCol("language"))

# Show the results of your text query in a table format
display(sentimentv2.transform(df).select("text", col("sentiment")[0].getItem("score").alias("positive
score")))
```

Expected results

TEXT	POSITIVE SCORE
I am so happy today, its sunny!	0.99511755
I am frustrated by this rush hour traffic	0.007274598
The cognitive services on spark aint bad	0.9144157

V3.1

```
# Create a dataframe that's tied to it's column names
df = spark.createDataFrame([
  ("I am so happy today, its sunny!", "en-US"),
  ("I am frustrated by this rush hour traffic", "en-US"),
  ("The cognitive services on spark aint bad", "en-US"),
], ["text", "language"])

# Run the Text Analytics service with options
sentiment = (TextSentiment()
    .setLinkedService(linked_service_name)
    .setTextCol("text")
    .setOutputCol("sentiment")
    .setErrorCol("error")
    .setLanguageCol("language"))

# Show the results of your text query in a table format
display(sentiment.transform(df).select("text", col("sentiment")[0].getItem("sentiment").alias("sentiment")))
```

Expected results

TEXT	SENTIMENT
I am so happy today, its sunny!	positive
I am frustrated by this rush hour traffic	negative

TEXT	SENTIMENT
The cognitive services on spark aint bad	positive

Language Detector

The Language Detector evaluates text input for each document and returns language identifiers with a score that indicates the strength of the analysis. This capability is useful for content stores that collect arbitrary text, where language is unknown. See the Supported languages in Text Analytics API for the list of enabled languages.

V2

```
# Create a dataframe that's tied to it's column names
df = spark.createDataFrame([
  ("Hello World",),
  ("Bonjour tout le monde",),
  ("La carretera estaba atascada. Había mucho tráfico el día de ayer.",),
  ("你好",),
  ("こんにちは",),
  (":) :( :D",)
], ["text",])

# Run the Text Analytics service with options
languagev2 = (LanguageDetectorV2()
    .setLinkedService(linked_service_name)
    .setTextCol("text")
    .setOutputCol("language")
    .setErrorCol("error"))

# Show the results of your text query in a table format
display(languagev2.transform(df))
```

Expected results

text	error	language
Hello World	undefined	▼ "[{"detectedLanguages":[{"name":"English","iso6391N..." ▼ 0: "{"detectedLanguages":[{"name":"English","iso6391Na..." ▼ detectedLanguages: "[{"name":"English","iso6391Name":"en","score":1}]" ▶ 0: "{"name":"English","iso6391Name":"en","score":1}"
Bonjour tout le monde	undefined	▼ "[{"detectedLanguages":[{"name":"French","iso6391Na..." ▼ 0: "{"detectedLanguages":[{"name":"French","iso6391Nam..." ▼ detectedLanguages: "[{"name":"French","iso6391Name":"fr","score":1}]" ▶ 0: "{"name":"French","iso6391Name":"fr","score":1}"
La carretera estaba atascada. Ha...	undefined	▼ "[{"detectedLanguages":[{"name":"Spanish","iso6391N..." ▼ 0: "{"detectedLanguages":[{"name":"Spanish","iso6391Na..." ▼ detectedLanguages: "[{"name":"Spanish","iso6391Name":"es","score":1}]" ▶ 0: "{"name":"Spanish","iso6391Name":"es","score":1}"
你好	undefined	▼ "[{"detectedLanguages":[{"name":"Chinese","iso6391N..." ▼ 0: "{"detectedLanguages":[{"name":"Chinese","iso6391Na..." ▶ detectedLanguages: "[{"name":"Chinese","iso6391Name":"zh","score":1}]"
こんにちは	undefined	▶ "[{"detectedLanguages":[{"name":"Japanese","iso6391..."
:) :(:D	undefined	▶ "[{"detectedLanguages":[{"name":"(Unknown)","iso639..."

V3.1

```
# Create a dataframe that's tied to it's column names
df = spark.createDataFrame([
  ("Hello World",),
  ("Bonjour tout le monde",),
  ("La carretera estaba atascada. Había mucho tráfico el día de ayer.",),
  ("你好",),
  ("こんにちは",),
  (":) :( :D",)
], ["text",])

# Run the Text Analytics service with options
language = (LanguageDetector()
    .setLinkedService(linked_service_name)
    .setTextCol("text")
    .setOutputCol("language")
    .setErrorCol("error"))

# Show the results of your text query in a table format
display(language.transform(df))
```

Expected results

text	error	language
Hello World	undefined	▼ "[{"detectedLanguage":{"name":"English","iso6391Nam..." ▼ 0: "{"detectedLanguage":{"name":"English","iso6391Name..." ▼ detectedLanguage: "{"name":"English","iso6391Name":"en","confidenceSc..." name: ""English"" iso6391Name: ""en"" confidenceScore: "0.81" warnings: "[]"
Bonjour tout le monde	undefined	▼ "[{"detectedLanguage":{"name":"French","iso6391Name..." ▼ 0: "{"detectedLanguage":{"name":"French","iso6391Name"..." ▼ detectedLanguage: "{"name":"French","iso6391Name":"fr","confidenceSco..." name: ""French"" iso6391Name: ""fr"" confidenceScore: "0.88" warnings: "[]"
La carretera estaba atascada. Ha...	undefined	▼ "[{"detectedLanguage":{"name":"Spanish","iso6391Nam..." ▼ 0: "{"detectedLanguage":{"name":"Spanish","iso6391Name..." ▼ detectedLanguage: "{"name":"Spanish","iso6391Name":"es","confidenceSc..." name: ""Spanish"" iso6391Name: ""es"" confidenceScore: "1" warnings: "[]"
你好	undefined	▶ "[{"detectedLanguage":{"name":"Chinese","iso6391Nam..."
こんにちは	undefined	▶ "[{"detectedLanguage":{"name":"Japanese","iso6391Na..."
:) :(:D	undefined	▶ "[{"detectedLanguage":{"name":"(Unknown)","iso6391N..."

Entity Detector

The Entity Detector returns a list of recognized entities with links to a well-known knowledge base. See the Supported languages in Text Analytics API for the list of enabled languages.

V2

```
df = spark.createDataFrame([
    ("1", "Microsoft released Windows 10"),
    ("2", "In 1975, Bill Gates III and Paul Allen founded the company.")
], ["if", "text"])

entityv2 = (EntityDetectorV2()
    .setLinkedService(linked_service_name)
    .setLanguage("en")
    .setOutputCol("replies")
    .setErrorCol("error"))

display(entityv2.transform(df).select("if", "text", col("replies")
[0].getItem("entities").alias("entities")))
```

Expected results

if ↑	text	entities
1	Microsoft released Windows 10	▼ "[{"wikipediaId":"Windows 10","name":"Windows 10","..." ▼ 0: "{"wikipediaId":"Windows 10","name":"Windows 10","w..." wikipediaId: ""Windows 10"" name: ""Windows 10"" wikipediaUrl: ""https://en.wikipedia.org/wiki/Windows_10"" wikipediaLanguage: ""en"" bingId: ""5f9fbd03-49c4-39ef-cc95-de83ab897b94"" ▶ matches: "[{"text":"Windows 10","offset":19,"length":10}]" ▼ 1: "{"wikipediaId":"Microsoft","name":"Microsoft","wik..." wikipediaId: ""Microsoft"" name: ""Microsoft"" wikipediaUrl: ""https://en.wikipedia.org/wiki/Microsoft"" wikipediaLanguage: ""en"" bingId: ""a093e9b9-90f5-a3d5-c4b8-5855e1b01f85"" ▶ matches: "[{"text":"Microsoft","offset":0,"length":9}]"
2	In 1975, Bill Gates III and Paul Allen founded the company.	▶ "[{"wikipediaId":"Bill Gates","name":"Bill Gates","..."

V3.1

```
df = spark.createDataFrame([
    ("1", "Microsoft released Windows 10"),
    ("2", "In 1975, Bill Gates III and Paul Allen founded the company.")
], ["if", "text"])

entity = (EntityDetector()
    .setLinkedService(linked_service_name)
    .setLanguage("en")
    .setOutputCol("replies")
    .setErrorCol("error"))

display(entity.transform(df).select("if", "text", col("replies")[0].getItem("entities").alias("entities")))
```

Expected results

if	text	entities
1	Microsoft released Windows 10	▼ "[{"name":"Microsoft","url":"https://en.wikipedia.o..." ▼ 0: "{"name":"Microsoft","url":"https://en.wikipedia.or..." name: ""Microsoft"" url: ""https://en.wikipedia.org/wiki/Microsoft"" dataSource: ""Wikipedia"" id: ""Microsoft"" language: ""en"" ▼ matches: "[{"confidenceScore":0.13,"text":"Microsoft","offse..." ▶ 0: "{"confidenceScore":0.13,"text":"Microsoft","offset..." ▶ 1: "{"name":"Windows 10","url":"https://en.wikipedia.o..."
2	In 1975, Bill Gates III and Paul Allen founded the company.	▼ "[{"name":"Bill Gates","url":"https://en.wikipedia...." ▼ 0: "{"name":"Bill Gates","url":"https://en.wikipedia.o..." name: ""Bill Gates"" url: ""https://en.wikipedia.org/wiki/Bill_Gates"" dataSource: ""Wikipedia"" id: ""Bill Gates"" language: ""en"" ▶ matches: "[{"confidenceScore":0.87,"text":"Bill Gates III","..." ▶ 1: "{"name":"Paul Allen","url":"https://en.wikipedia.o..."

Key Phrase Extractor

The Key Phrase Extraction evaluates unstructured text and returns a list of key phrases. This capability is useful if you need to quickly identify the main points in a collection of documents. See the Supported languages in Text Analytics API for the list of enabled languages.

V2

```
df = spark.createDataFrame([
    ("en", "Hello world. This is some input text that I love."),
    ("fr", "Bonjour tout le monde"),
    ("es", "La carretera estaba atascada. Había mucho tráfico el día de ayer.")
], ["lang", "text"])

keyPhrasesv2 = (KeyPhraseExtractorV2()
    .setLinkedService(linked_service_name)
    .setLanguageCol("lang")
    .setOutputCol("replies")
    .setErrorCol("error"))

display(keyPhrasesv2.transform(df).select("text", col("replies")
[0].getItem("keyPhrases").alias("keyPhrases")))
```

Expected results

TEXT	KEYPHRASES
Hello world. This is some input text that I love.	"["input text","world"]"
Bonjour tout le monde	"["monde"]"
La carretera estaba atascada. Había mucho tráfico el día de ayer.	"["carretera","tráfico","día"]"

V3.1

```
df = spark.createDataFrame([
    ("en", "Hello world. This is some input text that I love."),
    ("fr", "Bonjour tout le monde"),
    ("es", "La carretera estaba atascada. Había mucho tráfico el día de ayer.")
], ["lang", "text"])

keyPhrase = (KeyPhraseExtractor()
    .setLinkedService(linked_service_name)
    .setLanguageCol("lang")
    .setOutputCol("replies")
    .setErrorCol("error"))

display(keyPhrase.transform(df).select("text", col("replies")[0].getItem("keyPhrases").alias("keyPhrases")))
```

Expected results

TEXT	KEYPHRASES
Hello world. This is some input text that I love.	"["Hello world","input text"]"
Bonjour tout le monde	"["Bonjour","monde"]"
La carretera estaba atascada. Había mucho tráfico el día de ayer.	"["mucho tráfico","día","carretera","ayer"]"

Named Entity Recognition (NER)

Named Entity Recognition (NER) is the ability to identify different entities in text and categorize them into pre-defined classes or types such as: person, location, event, product, and organization. See the Supported languages in Text Analytics API for the list of enabled languages.

V2

```
df = spark.createDataFrame([
    ("1", "en", "I had a wonderful trip to Seattle last week."),
    ("2", "en", "I visited Space Needle 2 times.")
], ["id", "language", "text"])

nerv2 = (NERV2()
    .setLinkedService(linked_service_name)
    .setLanguageCol("language")
    .setOutputCol("replies")
    .setErrorCol("error"))

display(nerv2.transform(df).select("text", col("replies")[0].getItem("entities").alias("entities")))
```

Expected results

text	entities
I had a wonderful trip to Seattle last week.	▼ "[{"wikipediaId":"Seattle","name":"Seattle","wikipe..." ▼ 0: "{"wikipediaId":"Seattle","name":"Seattle","wikiped..." wikipediaId: ""Seattle"" name: ""Seattle"" wikipediaUrl: ""https://en.wikipedia.org/wiki/Seattle"" wikipediaLanguage: ""en"" bingId: ""5fbba6b8-85e1-4d41-9444-d9055436e473"" type: ""Location"" ▶ matches: "[{"text":"Seattle","offset":26,"length":7}]" ▼ 1: "{"name":"last week","matches":[{"text":"last week"..." name: ""last week"" ▶ matches: "[{"text":"last week","offset":34,"length":9}]" type: ""DateTime""
I visited Space Needle 2 times.	▶ "[{"wikipediaId":"Space Needle","name":"Space Needl..."

V3.1

```
df = spark.createDataFrame([
    ("1", "en", "I had a wonderful trip to Seattle last week."),
    ("2", "en", "I visited Space Needle 2 times.")
], ["id", "language", "text"])

ner = (NER()
    .setLinkedService(linked_service_name)
    .setLanguageCol("language")
    .setOutputCol("replies")
    .setErrorCol("error"))

display(ner.transform(df).select("text", col("replies")[0].getItem("entities").alias("entities")))
```

Expected results

text	entities
I had a wonderful trip to Seattle last week.	▼ "[{"confidenceScore":0.74,"text":"trip","offset":18..." ▼ 0: "{"confidenceScore":0.74,"text":"trip","offset":18,..." confidenceScore: "0.74" text: ""trip"" offset: "18" category: ""Event"" length: "4" ▼ 1: "{"confidenceScore":1,"text":"Seattle","subcategory..." confidenceScore: "1" text: ""Seattle"" subcategory: ""GPE"" offset: "26" category: ""Location"" length: "7" ▼ 2: "{"confidenceScore":0.8,"text":"last week","subcate..." confidenceScore: "0.8" text: ""last week"" subcategory: ""DateRange"" offset: "34" category: ""DateTime"" length: "9"
I visited Space Needle 2 times.	▶ "[{"confidenceScore":0.96,"text":"Space Needle","of..."

Personally Identifiable Information (PII) V3.1

The PII feature is part of NER and it can identify and redact sensitive entities in text that are associated with an individual person such as: phone number, email address, mailing address, passport number. See the Supported languages in Text Analytics API for the list of enabled languages.

V3.1

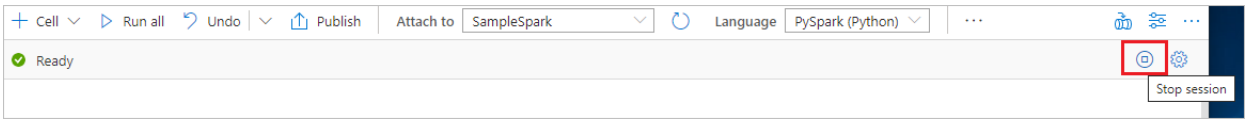
```
df = spark.createDataFrame([
    ("1", "en", "My SSN is 859-98-0987"),
    ("2", "en", "Your ABA number - 111000025 - is the first 9 digits in the lower left hand corner of your
personal check."),
    ("3", "en", "Is 998.214.865-68 your Brazilian CPF number?")
], ["id", "language", "text"])

pii = (PII()
    .setLinkedService(linked_service_name)
    .setLanguageCol("language")
    .setOutputCol("replies")
    .setErrorCol("error"))

display(pii.transform(df).select("text", col("replies")[0].getItem("entities").alias("entities")))
```

Expected results

text	entities
My SSN is 859-98-0987	▼ "[{"confidenceScore":0.65,"text":"859-98-0987","off..." ▼ 0: "{"confidenceScore":0.65,"text":"859-98-0987","offs..." confidenceScore: "0.65" text: ""859-98-0987"" offset: "10" category: ""USSocialSecurityNumber"" length: "11"
Your ABA number - 111000025 - is the first 9 digits in the lower left hand corner of your personal check.	▼ "[{"confidenceScore":0.8,"text":"111000025","offset..." ▼ 0: "{"confidenceScore":0.8,"text":"111000025","offset"..." confidenceScore: "0.8" text: ""111000025"" offset: "18" category: ""PhoneNumber"" length: "9" ▼ 1: "{"confidenceScore":0.75,"text":"111000025","offset..." confidenceScore: "0.75" text: ""111000025"" offset: "18" category: ""ABARoutingNumber"" length: "9" ▼ 2: "{"confidenceScore":0.65,"text":"111000025","offset..." confidenceScore: "0.65" text: ""111000025"" offset: "18" category: ""NZSocialWelfareNumber"" length: "9"
Is 998.214.865-68 your Brazilian CPF number?	"[]"

Clean up resources

To ensure the Spark instance is shut down, end any connected sessions(notebooks). The pool shuts down when the **idle time** specified in the Apache Spark pool is reached. You can also select **stop session** from the status bar at the upper right of the notebook.



Next steps

- Check out Synapse sample notebooks
- SynapseML GitHub Repo

Tutorial: Computer Vision with Cognitive Service

8/19/2022 • 3 minutes to read • Edit Online

Computer Vision is an Azure Cognitive Service that enables you to process images and return information based on the visual features. In this tutorial, you'll learn how to use Computer Vision to analyze images on Azure Synapse Analytics.

This tutorial demonstrates using text analytics with SynapseML to:

- Extract visual features from the image content
- Recognize characters from images (OCR)
- Analyze image content and generate thumbnail
- Detect and identify domain-specific content in an image
- Generate tags related to an image
- Generate a description of an entire image in human-readable language

Analyze image

Extracts a rich set of visual features based on the image content, such as objects, faces, adult content, and autogenerated text descriptions.

Example input

```
# Create a dataframe with the image URLs
df = spark.createDataFrame([
        ("<replace with your file path>/dog.jpg", )
    ], ["image", ])

# Run the Computer Vision service. Analyze Image extracts infortmation from/about the images.
analysis = (AnalyzeImage()
    .setLinkedService(cognitive_service_name)
    .setVisualFeatures(["Categories","Color","Description","Faces","Objects","Tags"])
    .setOutputCol("analysis_results")
    .setImageUrlCol("image")
    .setErrorCol("error"))

# Show the results of what you wanted to pull out of the images.
display(analysis.transform(df).select("image", "analysis_results.description.tags"))
```

Expected results

```
["dog","outdoor","fence","wooden","small","brown","building","sitting","front","bench","standing","table","w
alking","board","beach","holding","bridge"]
```

Optical character recognition (OCR)

Extract printed text, handwritten text, digits, and currency symbols from images, such as photos of street signs and products, as well as from documents—invoices, bills, financial reports, articles, and more. It's optimized to extract text from text-heavy images and multi-page PDF documents with mixed languages. It supports detecting both printed and handwritten text in the same image or document.

Example input

```
df = spark.createDataFrame([
        ("<replace with your file path>/ocr.jpg", )
    ], ["url", ])

ri = (ReadImage()
    .setLinkedService(cognitive_service_name)
    .setImageUrlCol("url")
    .setOutputCol("ocr"))

display(ri.transform(df))
```

Expected results

url ↓	ReadImage_ececcae91fa0_er...	ocr
https://mmlspark.bl...	undefined	▼ "{"status":"succeeded","createdDateTime":"2021-10-2..." status: ""succeeded"" createdDateTime: ""2021-10-21T10:39:29Z"" lastUpdatedDateTime: ""2021-10-21T10:39:30Z"" ▼ analyzeResult: "{"version":"3.0.0","readResults":[{"angle":20.4048..." version: ""3.0.0"" ▼ readResults: "[{"angle":20.4048,"page":1,"height":1008,"unit":"p..." ▼ 0: "{"angle":20.4048,"page":1,"height":1008,"unit":"pi..." angle: "20.4048" page: "1" height: "1008" unit: ""pixel"" ▼ lines: "[{"boundingBox":[182,511,652,677,609,796,140,630],..." ▼ 0: "{"boundingBox":[182,511,652,677,609,796,140,630],"..." ▶ boundingBox: "[182,511,652,677,609,796,140,630]" text: ""CLOSED"" ▶ words: "[{"boundingBox":[181,512,652,681,601,796,140,630],..." ▼ 1: "{"boundingBox":[146,643,603,813,593,841,136,671],"..." ▶ boundingBox: "[146,643,603,813,593,841,136,671]" text: ""WHEN ONE DOOR CLOSES, ANOTHER"" ▶ words: "[{"boundingBox":[146,644,220,672,210,699,136,671],..." ▼ 2: "{"boundingBox":[75,676,644,887,633,919,65,706],"te..." ▶ boundingBox: "[75,676,644,887,633,919,65,706]" text: ""OPENS. ALL YOU HAVE TO DO IS WALK IN"" ▶ words: "[{"boundingBox":[75,677,161,710,152,739,65,706],"t..." width: "756"

Generate thumbnails

Analyze the contents of an image to generate an appropriate thumbnail for that image. Computer Vision first generates a high-quality thumbnail and then analyzes the objects within the image to determine the area of interest. Computer Vision then crops the image to fit the requirements of the area of interest. The generated thumbnail can be presented using an aspect ratio that is different from the aspect ratio of the original image, depending on your needs.

Example input

```
df = spark.createDataFrame([
        ("<replace with your file path>/satya.jpeg", )
    ], ["url", ])

gt =  (GenerateThumbnails()
    .setLinkedService(cognitive_service_name)
    .setHeight(50)
    .setWidth(50)
    .setSmartCropping(True)
    .setImageUrlCol("url")
    .setOutputCol("thumbnails"))

thumbnails = gt.transform(df).select("thumbnails").toJSON().first()

import json
img = json.loads(thumbnails)["thumbnails"]

displayHTML("<img src='" + img + "'>")
```

Expected results

Tag image

Generates a list of words, or tags, that are relevant to the content of the supplied image. Tags are returned based on thousands of recognizable objects, living beings, scenery, or actions found in images. Tags may contain hints to avoid ambiguity or provide context, for example the tag "ascomycete" may be accompanied by the hint "fungus".

Let continue using Satya's image as an example.

```
df = spark.createDataFrame([
        ("<replace with your file path>/satya.jpeg", )
    ], ["url", ])

ti = (TagImage()
    .setLinkedService(cognitive_service_name)
    .setImageUrlCol("url")
    .setOutputCol("tags"))

display(ti.transform(df))
```

Expected result

url	TagImage_b02b2481c189_error	tags
https://docs.mi...	undefined	▼ "{"tags":[{"name":"person","confidence":0.999554514..." ▼ tags: "[{"name":"person","confidence":0.9995545148849487}..." ▶ 0: "{"name":"person","confidence":0.9995545148849487}" ▶ 1: "{"name":"suit","confidence":0.9894102215766907}" ▶ 2: "{"name":"man","confidence":0.9881249666213989}" ▶ 3: "{"name":"tie","confidence":0.9581113457679749}" ▶ 4: "{"name":"human face","confidence":0.95434975624084..." ▶ 5: "{"name":"clothing","confidence":0.8706792593002319..." ▶ 6: "{"name":"outdoor","confidence":0.8646519184112549}" ▶ 7: "{"name":"smile","confidence":0.8594179749488831}" ▶ 8: "{"name":"glasses","confidence":0.6877670288085938}" requestId: ""d8625b53-ed72-4a35-a3f2-ba9825d88142""

Describe image

Generate a description of an entire image in human-readable language, using complete sentences. Computer Vision's algorithms generate various descriptions based on the objects identified in the image. The descriptions are each evaluated and a confidence score generated. A list is then returned ordered from highest confidence score to lowest.

Let continue using Satya's image as an example.

```
df = spark.createDataFrame([
        ("<replace with your file path>/satya.jpeg", )
    ], ["url", ])

di = (DescribeImage()
    .setLinkedService(cognitive_service_name)
    .setMaxCandidates(3)
    .setImageUrlCol("url")
    .setOutputCol("descriptions"))

display(di.transform(df))
```

Expected result

Recognize domain-specific content

Use domain models to detect and identify domain-specific content in an image, such as celebrities and landmarks. For example, if an image contains people, Computer Vision can use a domain model for celebrities to determine if the people detected in the image are known celebrities.

Let continue using Satya's image as an example.

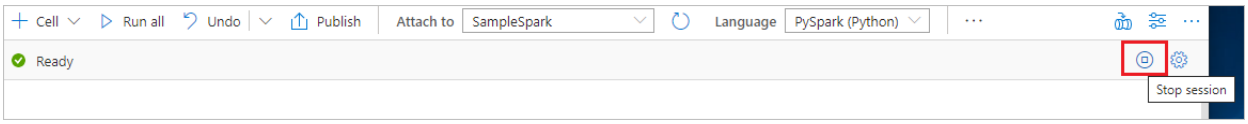
```
df = spark.createDataFrame([
        ("<replace with your file path>/satya.jpeg", )
    ], ["url", ])

celeb =  (RecognizeDomainSpecificContent()
    .setLinkedService(cognitive_service_name)
    .setModel("celebrities")
    .setImageUrlCol("url")
    .setOutputCol("celebs"))

display(celeb.transform(df))
```

Expected result

url	RecognizeDomain...	celebs
https://d...	undefined	▼ "{"requestId":"295bc3d5-31b9-4870-88b2-3f8f0f23bec6..." requestId: ""295bc3d5-31b9-4870-88b2-3f8f0f23bec6"" ▶ metadata: "{"width":600,"height":900,"format":"Jpeg"}" ▼ result: "{"celebrities":[{"name":"Satya Nadella","faceRecta..." ▼ celebrities: "[{"name":"Satya Nadella","faceRectangle":{"left":2..." ▼ 0: "{"name":"Satya Nadella","faceRectangle":{"left":23..." name: ""Satya Nadella"" ▶ faceRectangle: "{"left":239,"top":293,"width":138,"height":138}" confidence: "0.9998459815979004"

Clean up resources

To ensure the Spark instance is shut down, end any connected sessions(notebooks). The pool shuts down when the **idle time** specified in the Apache Spark pool is reached. You can also select **stop session** from the status bar at the upper right of the notebook.



Next steps

- Check out Synapse sample notebooks
- SynapseML GitHub Repo

Tutorial: Form Recognizer with Applied AI Service

8/19/2022 • 3 minutes to read • Edit Online

Azure Form Recognizer is an Azure Applied AI Service that enables you to build automated data processing application using machine learning technology. In this tutorial, you'll learn how to easily enrich your data in Azure Synapse Analytics. You'll use Form Recognizer to analyze your forms and documents, extracts text and data, and returns a structured JSON output. You quickly get accurate results that are tailored to your specific content without excessive manual intervention or extensive data science expertise.

This tutorial demonstrates using form recognizer with SynapseML to:

- Extract text and layout from a given document
- Detect and extract data from receipts
- Detect and extract data from business cards
- Detect and extract data from invoices
- Detect and extract data from identification documents

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Spark pool in your Azure Synapse Analytics workspace. For details, see Create a Spark pool in Azure Synapse.
- Pre-configuration steps described in the tutorial Configure Cognitive Services in Azure Synapse.

Get started

Open Synapse Studio and create a new notebook. To get started, import SynapseML.

```
import synapse.ml
from synapse.ml.cognitive import *
```

Configure form recognizer

Use the linked form recognizer you configured in the pre-configuration steps .

```
cognitive_service_name = "<Your linked service for form recognizer>"
```

Analyze Layout

Extract text and layout information from a given document. The input document must be of one of the supported content types - 'application/pdf', 'image/jpeg', 'image/png' or 'image/tiff'.

Example input

Purchase Order

Hero Limited

Company Phone: 555-348-6512
Website: www.herolimited.com
Email:
accounts@herolimited.com

Purchase Order

Dated As: 12/20/2020
Purchase Order #: 948284

Shipped To

Vendor Name: Balozi Khamisi
Company Name: Higgly Wiggly Books
Address: 938 NE Burner Road
Boulder City, CO 92848 **Phone:** 938-294-2949

Shipped From

Name: Kidane Tsehaye
Company Name: Jupiter Book Supply
Address: 383 N Kinnick Road
Seattle, WA 38383 **Phone:** 932-299-0292

Details	Quantity	Unit Price	Total
Bindings	20	1.00	20.00
Covers Small	20	1.00	20.00
Feather Bookmark	20	5.00	100.00
Copper Swirl Marker	20	5.00	100.00

SUBTOTAL	$140.00
TAX	$4.00
TOTAL	$144.00

Kidane Tsehaye

Kidane Tsehaye
Manager

Additional Notes:

Do not Jostle Box. Unpack carefully. Enjoy.

Jupiter Book Supply will refund you 50% per book if returned within 60 days of reading and offer you 25% off you next total purchase.

```
from pyspark.sql.functions import col, flatten, regexp_replace, explode, create_map, lit

imageDf = spark.createDataFrame([
  ("<replace with your file path>/layout.jpg",)
], ["source",])

analyzeLayout = (AnalyzeLayout()
                 .setLinkedService(cognitive_service_name)
                 .setImageUrlCol("source")
                 .setOutputCol("layout")
                 .setConcurrency(5))


display(analyzeLayout
        .transform(imageDf)
        .withColumn("lines", flatten(col("layout.analyzeResult.readResults.lines")))
        .withColumn("readLayout", col("lines.text"))
        .withColumn("tables", flatten(col("layout.analyzeResult.pageResults.tables")))
        .withColumn("cells", flatten(col("tables.cells")))
        .withColumn("pageLayout", col("cells.text"))
        .select("source", "readLayout", "pageLayout"))
```

Expected results

source	readLayout	pageLayout
https://m...	▼ "["Purchase Order","Hero Limited","Purchase Ord 0: ""Purchase Order"" 1: ""Hero Limited"" 2: ""Purchase Order"" 3: ""Company Phone: 555-348-6512"" 4: ""Website: www.herolimited.com"" 5: ""Email:"" 6: ""Dated As: 12/20/2020"" 7: ""accounts@herolimited.com"" 8: ""Purchase Order #: 948284"" 9: ""Shipped To"" 10: ""Vendor Name: Balozi Khamisi"" 11: ""Company Name: Higgly Wiggly Books"" 12: ""Address: 938 NE Burner Road"" 13: ""Boulder City, CO 92848"" 14: ""Phone: 938-294-2949"" 15: ""Shipped From"" 16: ""Name: Kidane Tsehaye"" 17: ""Company Name: Jupiter Book Supply"" 18: ""Address: 383 N Kinnick Road"" 19: ""Seattle, WA 38383"" 20: ""Phone: 932-299-0292"" 21: ""Details"" 22: ""Quantity"" 23: ""Unit Price"" 24: ""Total"" 25: ""Bindings"" 26: ""20"" 27: ""1.00""	▼ "["Details","Quantity","Unit Price","Total","Bindin..." 0: ""Details"" 1: ""Quantity"" 2: ""Unit Price"" 3: ""Total"" 4: ""Bindings"" 5: ""20"" 6: ""1.00"" 7: ""20.00"" 8: ""Covers Small"" 9: ""20"" 10: ""1.00"" 11: ""20.00"" 12: ""Feather Bookmark"" 13: ""20"" 14: ""5.00"" 15: ""100.00"" 16: ""Copper Swirl Marker"" 17: ""20"" 18: ""5.00"" 19: ""100.00"" 20: ""SUBTOTAL"" 21: ""$140.00"" 22: ""TAX"" 23: ""$4.00"" 24: """" 25: """" 26: ""TOTAL"" 27: ""$144.00""

Analyze Receipts

Detects and extracts data from receipts using optical character recognition (OCR) and our receipt model, enabling you to easily extract structured data from receipts such as merchant name, merchant phone number, transaction date, transaction total, and more.

Example input

Contoso

Contoso
123 Main Street
Redmond, WA 98052

123-456-7890

6/10/2019 13:59
Sales Associate: Paul

---

1 Surface Pro 6 256GB / Intel Core i5 / 8GB RAM (Black)	$ 999.00
1 SurfacePen	$ 99.99

Sub-Total	$ 1098.99
Tax	$ 104.40
Total	$ 1203.39

```
imageDf2 = spark.createDataFrame([
  ("<replace with your file path>/receipt1.png",)
], ["image",])

analyzeReceipts = (AnalyzeReceipts()
                  .setLinkedService(cognitive_service_name)
                  .setImageUrlCol("image")
                  .setOutputCol("parsed_document")
                  .setConcurrency(5))


results = analyzeReceipts.transform(imageDf2).cache()
display(results.select("image", "parsed_document"))
```

Expected results

image	parsed_document
https://mmlspark.blob.core.wi...	▼ "{"status":"succeeded","createdDateTime":"2021-10-2..." status: ""succeeded"" createdDateTime: ""2021-10-20T09:19:54Z"" lastUpdatedDateTime: ""2021-10-20T09:19:57Z"" ▼ analyzeResult: "{"version":"2.1.0","readResults":[{"angle":-0.0752..." version: ""2.1.0"" ▼ readResults: "[{"angle":-0.0752,"page":1,"height":3000,"unit":"p..." ▶ 0: "{"angle":-0.0752,"page":1,"height":3000,"unit":"pi..." ▼ documentResults: "[{"docType":"prebuilt:receipt","pageRange":[1,1],"..." ▼ 0: "{"docType":"prebuilt:receipt","pageRange":[1,1],"f..." docType: ""prebuilt:receipt"" ▶ pageRange: "[1,1]" ▼ fields: "{"Tax":{"valueNumber":104.4,"page":1,"boundingBox"..." ▶ Tax: "{"valueNumber":104.4,"page":1,"boundingBox":[939.7..." ▶ Subtotal: "{"valueNumber":1098.99,"page":1,"boundingBox":[963..." ▶ MerchantName: "{"page":1,"valueString":"Contoso","boundingBox":[3..." ▶ TransactionTime: "{"page":1,"boundingBox":[511,1223,628,1224,625,129..." ▶ MerchantPhoneNumber: "{"page":1,"boundingBox":[303,1003,621,1009,619,107..." ▶ Total: "{"valueNumber":1203.39,"page":1,"boundingBox":[914..." ▶ TransactionDate: "{"page":1,"boundingBox":[299,1221,497,1222,496,129..." ▶ ReceiptType: "{"type":"string","confidence":0.99,"valueString":"..." ▶ MerchantAddress: "{"page":1,"valueString":"123 Main Street Redmond, ..." ▶ Items: "{"type":"array","valueArray":["{\"type\":\"object\..."

Analyze business cards

Detects and extracts data from business cards using optical character recognition (OCR) and our business card model, enabling you to easily extract structured data from business cards such as contact names, company names, phone numbers, emails, and more.

Example input

```
imageDf3 = spark.createDataFrame([
  ("<replace with your file path>/business_card.jpg",)
], ["source",])

analyzeBusinessCards = (AnalyzeBusinessCards()
                 .setLinkedService(cognitive_service_name)
                 .setImageUrlCol("source")
                 .setOutputCol("businessCards")
                 .setConcurrency(5))

display(analyzeBusinessCards
        .transform(imageDf3)
        .withColumn("documents", explode(col("businessCards.analyzeResult.documentResults.fields")))
        .select("source", "documents"))
```

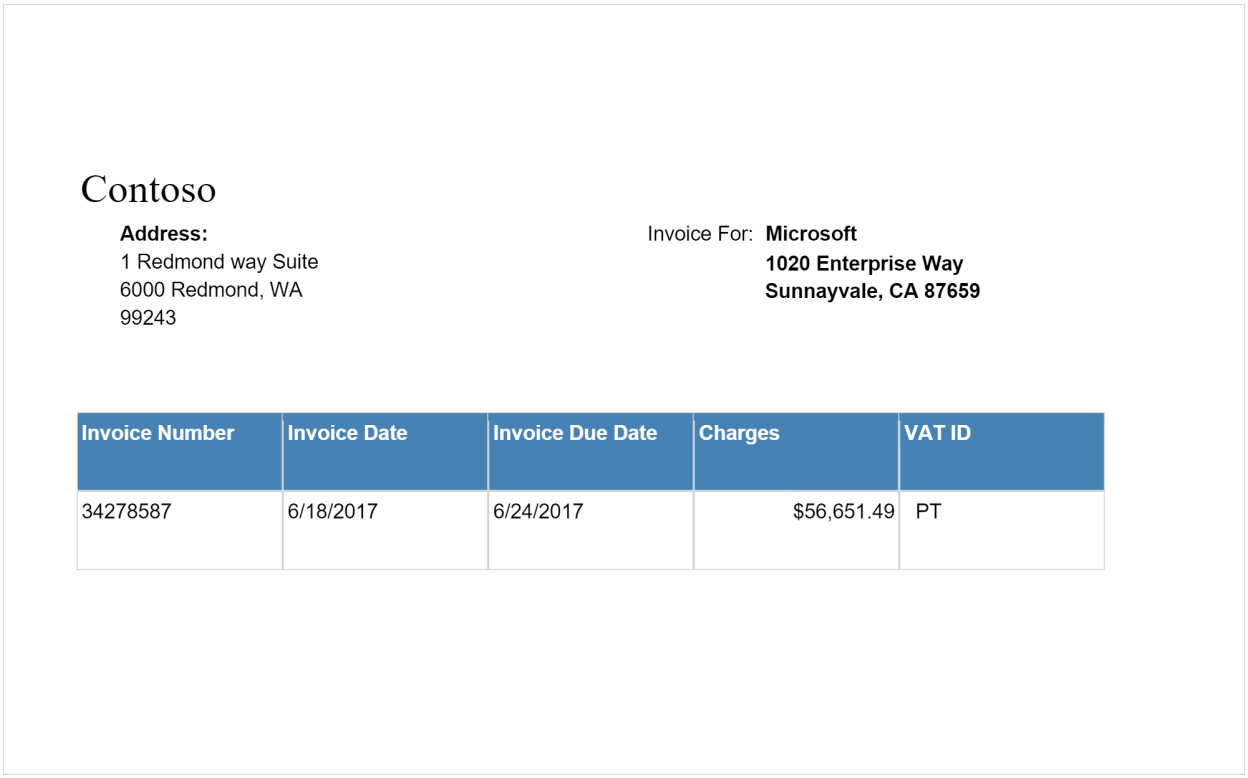
Expected results

source	documents
https://mmlspark.blob.core.wind...	▼ "{"Addresses":{"type":"array","valueArray":["{\"typ..."
	▶ Addresses: "{"type":"array","valueArray":["{\"type\":\"string\..."
	▼ MobilePhones: "{"type":"array","valueArray":["{\"type\":\"phoneNu..."
	type: ""array""
	▼ valueArray: "["{\"type\":\"phoneNumber\",\"text\":\"+44 (0) 791..."
	0: ""{\"type\":\"phoneNumber\",\"text\":\"+44 (0) 7911 123456\",\"boundingBox\":[2426.8,1040.1,3065.5,846,"
	▼ Departments: "{"type":"array","valueArray":["{\"type\":\"string\..."
	type: ""array""
	▼ valueArray: "["{\"type\":\"string\",\"valueString\":\"Cloud & A..."
	0: ""{\"type\":\"string\",\"valueString\":\"Cloud & AI Department\",\"text\":\"Cloud & AI Department\",\"bour"
	▶ CompanyNames: "{"type":"array","valueArray":["{\"type\":\"string\..."
	▶ Emails: "{"type":"array","valueArray":["{\"type\":\"string\..."
	▶ Websites: "{"type":"array","valueArray":["{\"type\":\"string\..."
	▶ JobTitles: "{"type":"array","valueArray":["{\"type\":\"string\..."
	▶ ContactNames: "{"type":"array","valueArray":["{\"type\":\"object\..."
	▶ WorkPhones: "{"type":"array","valueArray":["{\"type\":\"phoneNu..."
	▶ Faxes: "{"type":"array","valueArray":["{\"type\":\"phoneNu..."

Analyze invoices

Detects and extracts data from invoices using optical character recognition (OCR) and our invoice understanding deep learning models, enabling you to easily extract structured data from invoices such as customer, vendor, invoice ID, invoice due date, total, invoice amount due, tax amount, ship to, bill to, line items and more.

Example input



Contoso

Address:
1 Redmond way Suite
6000 Redmond, WA
99243

Invoice For: **Microsoft**
1020 Enterprise Way
Sunnayvale, CA 87659

Invoice Number	Invoice Date	Invoice Due Date	Charges	VAT ID
34278587	6/18/2017	6/24/2017	$56,651.49	PT

```
imageDf4 = spark.createDataFrame([
  ("<replace with your file path>/invoice.png",)
], ["source",])

analyzeInvoices = (AnalyzeInvoices()
                  .setLinkedService(cognitive_service_name)
                  .setImageUrlCol("source")
                  .setOutputCol("invoices")
                  .setConcurrency(5))

display(analyzeInvoices
        .transform(imageDf4)
        .withColumn("documents", explode(col("invoices.analyzeResult.documentResults.fields")))
        .select("source", "documents"))
```

Expected results

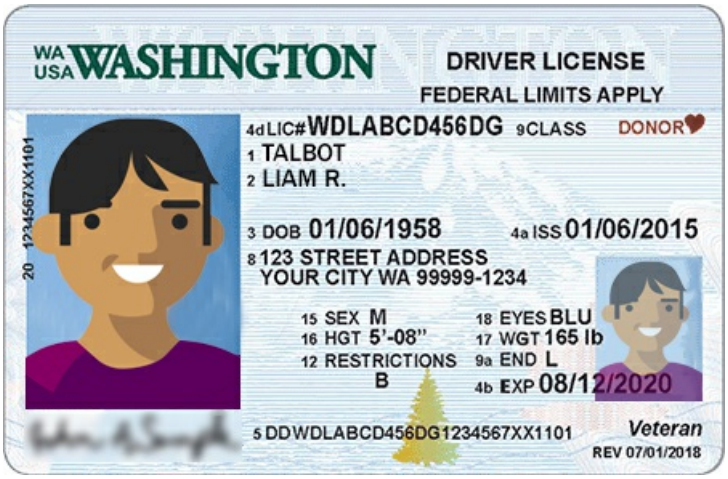
source	documents
https://mmlspark.blob.core.wind...	▼ "{"CustomerAddress":{"page":1,"valueString":"1020 E..."

▼ CustomerAddress: "{"page":1,"valueString":"1020 Enterprise Way Sunna..."
page: "1"
valueString: ""1020 Enterprise Way Sunnayvale, CA 87659""
▶ boundingBox: "[1221.3,399.5,1567.3,401.6,1566.8,480.1,1220.8,478..."
text: ""1020 Enterprise Way Sunnayvale, CA 87659""
confidence: "0.954"
type: ""string""
▶ DueDate: "{"page":1,"boundingBox":[785,799,929,798,927,829,7..."
▶ CustomerName: "{"page":1,"valueString":"Microsoft","boundingBox":..."
▶ VendorName: "{"page":1,"valueString":"Contoso","boundingBox":[1..."
▶ InvoiceId: "{"page":1,"valueString":"34278587","boundingBox":[..."
▶ InvoiceDate: "{"page":1,"boundingBox":[456,800,599,799,598,829,4..."
▶ VendorAddress: "{"page":1,"valueString":"1 Redmond way Suite 6000 ..."
▶ CustomerAddressRecipient: "{"page":1,"valueString":"Microsoft","boundingBox":..."
▶ Items: "{"type":"array","valueArray":["{\"type\":\"object\..."

Analyze ID documents

Detects and extracts data from identification documents using optical character recognition (OCR) and our ID document model, enabling you to easily extract structured data from ID documents such as first name, last name, date of birth, document number, and more.

Example input



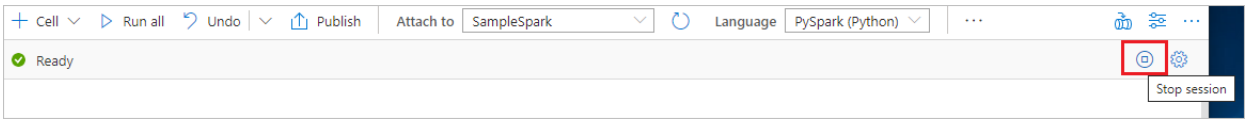

```
imageDf5 = spark.createDataFrame([
  ("<replace with your file path>/id.jpg",)
], ["source",])

analyzeIDDocuments = (AnalyzeIDDocuments()
                  .setLinkedService(cognitive_service_name)
                  .setImageUrlCol("source")
                  .setOutputCol("ids")
                  .setConcurrency(5))

display(analyzeIDDocuments
        .transform(imageDf5)
        .withColumn("documents", explode(col("ids.analyzeResult.documentResults.fields")))
        .select("source", "documents"))
```

Expected results

source	documents
https://mmlspark.blob.core.wind...	▼ "{"DateOfExpiration":{"page":1,"boundingBox":[332,2..."
	▼ DateOfExpiration: "{"page":1,"boundingBox":[332,230,414,228,414,244,3..."
	page: "1"
	▶ boundingBox: "[332,230,414,228,414,244,332,245]"
	text: ""08/12/2020""
	confidence: "0.99"
	valueDate: ""2020-08-12""
	type: ""date""
	▼ LastName: "{"page":1,"valueString":"TALBOT","boundingBox":[16..."
	page: "1"
	valueString: ""TALBOT""
	▶ boundingBox: "[160,86,213,85,213,99,160,100]"
	text: ""TALBOT""
	confidence: "0.987"
	type: ""string""
	▶ CountryRegion: "{"type":"countryRegion","confidence":0.99}"
	▶ Address: "{"page":1,"valueString":"123 STREET ADDRESS YOUR C..."
	▶ FirstName: "{"page":1,"valueString":"LIAM R.","boundingBox":[1..."
	▶ Region: "{"type":"string","confidence":0.99,"valueString":"..."
	▶ DocumentNumber: "{"page":1,"valueString":"WDLABCD456DG","boundingBo..."
	▶ Sex: "{"page":1,"valueString":"M","boundingBox":[226,190..."
	▶ DateOfBirth: "{"page":1,"boundingBox":[187,133,272,132,272,148,1..."

Clean up resources

To ensure the Spark instance is shut down, end any connected sessions(notebooks). The pool shuts down when the **idle time** specified in the Apache Spark pool is reached. You can also select **stop session** from the status bar at the upper right of the notebook.



Next steps

- Train a custom form recognizer model
- Check out Synapse sample notebooks
- SynapseML GitHub Repo

Tutorial: Translator with Cognitive Service

8/19/2022 • 3 minutes to read • Edit Online

Translator is an Azure Cognitive Service that enables you to perform language translation and other language-related operations. In this tutorial, you'll learn how to use Translator to build intelligent, multi-language solutions on Azure Synapse Analytics.

This tutorial demonstrates using translator with MMLSpark to:

- Translate text
- Transliterate text
- Detect language
- Break sentence
- Dictionary lookup
- Dictionary example

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Spark pool in your Azure Synapse Analytics workspace. For details, see Create a Spark pool in Azure Synapse.
- Pre-configuration steps described in the tutorial Configure Cognitive Services in Azure Synapse.

Get started

Open Synapse Studio and create a new notebook. To get started, import MMLSpark.

```
import mmlspark
from mmlspark.cognitive import *
from notebookutils import mssparkutils
from pyspark.sql.functions import col, flatten
```

Configure translator

Use the linked translator you configured in the pre-configuration steps .

```
cognitive_service_name = "<Your linked service for translator>"
```

Translate Text

The core operation of the Translator service is to translate text.

```
df = spark.createDataFrame([
  (["Hello, what is your name?", "Bye"],)
], ["text",])

translate = (Translate()
    .setLinkedService(cognitive_service_name)
    .setTextCol("text")
    .setToLanguage(["zh-Hans", "fr"])
    .setOutputCol("translation")
    .setConcurrency(5))

display(translate
      .transform(df)
      .withColumn("translation", flatten(col("translation.translations")))
      .withColumn("translation", col("translation.text"))
      .select("translation"))
```

Expected results

```
["你好，你叫什么名字？","Bonjour, quel est votre nom?","再见","Au revoir"]
```

Transliterate Text

Transliteration is the process of converting a word or phrase from the script (alphabet) of one language to another based on phonetic similarity.

```
transliterateDf =  spark.createDataFrame([
  (["こんにちは", "さようなら"],)
], ["text",])

transliterate = (Transliterate()
    .setLinkedService(cognitive_service_name)
    .setLanguage("ja")
    .setFromScript("Jpan")
    .setToScript("Latn")
    .setTextCol("text")
    .setOutputCol("result"))

display(transliterate
    .transform(transliterateDf)
    .withColumn("text", col("result.text"))
    .withColumn("script", col("result.script"))
    .select("text", "script"))
```

Expected results

TEXT	SCRIPT
"["Kon'nichiwa","sayonara"]"	"["Latn","Latn"]"

Detect Language

If you know that you'll need translation, but don't know the language of the text that will be sent to the Translator service, you can use the language detection operation.

```
detectDf =  spark.createDataFrame([
  (["Hello, what is your name?"],)
], ["text",])

detect = (Detect()
    .setLinkedService(cognitive_service_name)
    .setTextCol("text")
    .setOutputCol("result"))

display(detect
    .transform(detectDf)
    .withColumn("language", col("result.language"))
    .select("language"))
```

Expected results

```
"["en"]"
```

Break Sentence

Identifies the positioning of sentence boundaries in a piece of text.

```
bsDf =  spark.createDataFrame([
  (["Hello, what is your name?"],)
], ["text",])

breakSentence = (BreakSentence()
    .setLinkedService(cognitive_service_name)
    .setTextCol("text")
    .setOutputCol("result"))

display(breakSentence
    .transform(bsDf)
    .withColumn("sentLen", flatten(col("result.sentLen")))
    .select("sentLen"))
```

Expected results

```
"[25]"
```

Dictionary lookup (alternate translations)

With the endpoint, you can get alternate translations for a word or phrase.

```
dictDf = spark.createDataFrame([
  (["fly"],)
], ["text",])

dictionaryLookup = (DictionaryLookup()
    .setLinkedService(cognitive_service_name)
    .setFromLanguage("en")
    .setToLanguage("es")
    .setTextCol("text")
    .setOutputCol("result"))

display(dictionaryLookup
    .transform(dictDf)
    .withColumn("translations", flatten(col("result.translations")))
    .withColumn("normalizedTarget", col("translations.normalizedTarget"))
    .select("normalizedTarget"))
```

Expected results

NORMALIZEDTARGET
"["volar","mosca","operan","pilotar","moscas","marcha"]"

Dictionary examples (translations in context)

After you've performed a dictionary lookup, you can pass the source text and translation to the dictionary/examples endpoint to get a list of examples that show both terms in the context of a sentence or phrase.

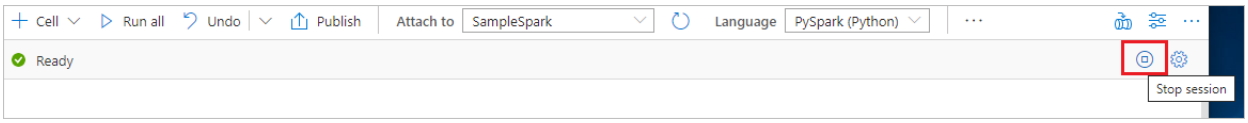
```
dictDf = spark.createDataFrame([
  ([("fly", "volar")],)
], ["textAndTranslation",])

dictionaryExamples = (DictionaryExamples()
    .setLinkedService(cognitive_service_name)
    .setFromLanguage("en")
    .setToLanguage("es")
    .setTextAndTranslationCol("textAndTranslation")
    .setOutputCol("result"))

display(dictionaryExamples
    .transform(dictDf)
    .withColumn("examples", flatten(col("result.examples")))
    .select("examples"))
```

Expected results

```
[{"sourcePrefix":"I mean, for a guy who could ","sourceSuffix":".","targetPrefix":"Quiero decir, para un
tipo que podía ","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."},{"sourcePrefix":"Now it's time
to make you ","sourceSuffix":".","targetPrefix":"Ahora es hora de que te haga
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."},{"sourcePrefix":"One happy thought will make
you ","sourceSuffix":".","targetPrefix":"Uno solo te hará
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."},{"sourcePrefix":"They need machines to
","sourceSuffix":".","targetPrefix":"Necesitan máquinas para
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."},{"sourcePrefix":"That should really
","sourceSuffix":".","targetPrefix":"Eso realmente debe
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."},{"sourcePrefix":"It sure takes longer when you
can't ","sourceSuffix":".","targetPrefix":"Lleva más tiempo cuando no puedes
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."},{"sourcePrefix":"I have to ","sourceSuffix":"
home in the morning.","targetPrefix":"Tengo que ","targetTerm":"volar","sourceTerm":"fly","targetSuffix":" a
casa por la mañana."},{"sourcePrefix":"You taught me to ","sourceSuffix":".","targetPrefix":"Me enseñaste a
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."},{"sourcePrefix":"I think you should
","sourceSuffix":" with the window closed.","targetPrefix":"Creo que debemos
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":" con la ventana cerrada."},{"sourcePrefix":"They
look like they could ","sourceSuffix":".","targetPrefix":"Parece que pudieran
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."},{"sourcePrefix":"But you can
","sourceSuffix":", for instance?","targetPrefix":"Que puedes
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":", por ejemplo."},{"sourcePrefix":"At least until
her kids can be able to ","sourceSuffix":".","targetPrefix":"Al menos hasta que sus hijos sean capaces de
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."},{"sourcePrefix":"I thought you could
","sourceSuffix":".","targetPrefix":"Pensé que podías
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."},{"sourcePrefix":"I was wondering what it would
be like to ","sourceSuffix":".","targetPrefix":"Me preguntaba cómo sería
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."},{"sourcePrefix":"But nobody else can
","sourceSuffix":".","targetPrefix":"Pero nadie puede
","targetTerm":"volar","sourceTerm":"fly","targetSuffix":"."}]
```


Clean up resources

To ensure the Spark instance is shut down, end any connected sessions(notebooks). The pool shuts down when the **idle time** specified in the Apache Spark pool is reached. You can also select **stop session** from the status bar at the upper right of the notebook.



Next steps

- Check out Synapse sample notebooks
- MMLSpark GitHub Repo

Tutorial: Build machine learning applications using Synapse Machine Learning

8/19/2022 • 6 minutes to read • Edit Online

In this article, you will learn how to use Synapse Machine Learning (SynapseML) to create machine learning applications. SynapseML expands the distributed machine learning solution of Apache Spark by adding many deep learning and data science tools, such as Azure Cognitive Services, OpenCV, LightGBM and more. SynapseML allows you to build powerful and highly scalable predictive and analytical models from various Spark data sources. Synapse Spark provide built-in SynapseML libraries including:

- Vowpal Wabbit – Library services for Machine learning to enable Text analytics like sentiment analysis in tweets.
- Cognitive Services on Spark – To combine the feature of Azure Cognitive Services in SparkML pipelines in order to derive solution design for cognitive data modeling services like anomaly detection.
- LightGBM – LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and higher efficiency.
- Conditional KNN - Scalable KNN Models with Conditional Queries.
- HTTP on Spark – Enables distributed Microservices orchestration in integrating Spark and HTTP protocol-based accessibility.

This tutorial covers samples using Azure Cognitive Services in SynapseML for

- Text Analytics - get the sentiment (or mood) of a set of sentences.
- Computer Vision - get the tags (one-word descriptions) associated with a set of images.
- Bing Image Search - search the web for images related to a natural language query.
- Anomaly Detector - detect anomalies within a time series data.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Azure Synapse Analytics workspace with an Azure Data Lake Storage Gen2 storage account configured as the default storage. You need to be the *Storage Blob Data Contributor* of the Data Lake Storage Gen2 file system that you work with.
- Spark pool in your Azure Synapse Analytics workspace. For details, see Create a Spark pool in Azure Synapse.
- Pre-configuration steps described in the tutorial Configure Cognitive Services in Azure Synapse.

Get started

To get started, import SynapseML and configurate service keys.

```
import synapse.ml

from synapse.ml.cognitive import *
from notebookutils import mssparkutils

# A general Cognitive Services key for Text Analytics and Computer Vision (or use separate keys that belong
to each service)
cognitive_service_key = mssparkutils.credentials.getSecret("ADD_YOUR_KEY_VAULT_NAME",
"ADD_YOUR_SERVICE_KEY","ADD_YOUR_KEY_VAULT_LINKED_SERVICE_NAME")
# A Bing Search v7 subscription key
bingsearch_service_key = mssparkutils.credentials.getSecret("ADD_YOUR_KEY_VAULT_NAME",
"ADD_YOUR_BING_SEARCH_KEY","ADD_YOUR_KEY_VAULT_LINKED_SERVICE_NAME")
# An Anomaly Dectector subscription key
anomalydetector_key = mssparkutils.credentials.getSecret("ADD_YOUR_KEY_VAULT_NAME",
"ADD_YOUR_ANOMALY_KEY","ADD_YOUR_KEY_VAULT_LINKED_SERVICE_NAME")
```

Text analytics sample

The Text Analytics service provides several algorithms for extracting intelligent insights from text. For example, we can find the sentiment of given input text. The service will return a score between 0.0 and 1.0 where low scores indicate negative sentiment and high score indicates positive sentiment. This sample uses three simple sentences and returns the sentiment for each.

```
from pyspark.sql.functions import col

# Create a dataframe that's tied to it's column names
df_sentences = spark.createDataFrame([
  ("I am so happy today, its sunny!", "en-US"),
  ("this is a dog", "en-US"),
  ("I am frustrated by this rush hour traffic!", "en-US")
], ["text", "language"])

# Run the Text Analytics service with options
sentiment = (TextSentiment()
    .setTextCol("text")
    .setLocation("eastasia") # Set the location of your cognitive service
    .setSubscriptionKey(cognitive_service_key)
    .setOutputCol("sentiment")
    .setErrorCol("error")
    .setLanguageCol("language"))

# Show the results of your text query in a table format

display(sentiment.transform(df_sentences).select("text", col("sentiment")
[0].getItem("sentiment").alias("sentiment")))
```

Expected results

TEXT	SENTIMENT
I am frustrated by this rush hour traffic!	negative
this is a dog	neutral
I am so happy today, its sunny!	positive

Computer vision sample

Computer Vision analyzes images to identify structure such as faces, objects, and natural-language descriptions. In this sample, we tag the follow image. Tags are one-word descriptions of things in the image like recognizable objects, people, scenery, and actions.

```
# Create a dataframe with the image URL
df_images = spark.createDataFrame([
        ("https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-
files/master/ComputerVision/Images/objects.jpg", )
    ], ["image", ])

# Run the Computer Vision service. Analyze Image extracts information from/about the images.
analysis = (AnalyzeImage()
    .setLocation("eastasia") # Set the location of your cognitive service
    .setSubscriptionKey(cognitive_service_key)
    .setVisualFeatures(["Categories","Color","Description","Faces","Objects","Tags"])
    .setOutputCol("analysis_results")
    .setImageUrlCol("image")
    .setErrorCol("error"))

# Show the results of what you wanted to pull out of the images.
display(analysis.transform(df_images).select("image", "analysis_results.description.tags"))
```

Expected results

IMAGE	TAGS
`https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/objects.jpg`	[skating, person, man, outdoor, riding, sport, skateboard, young, board, shirt, air, park, boy, side, jumping, ramp, trick, doing, flying]

Bing image search sample

Bing Image Search searches the web to retrieve images related to a user's natural language query. In this sample, we use a text query that looks for images with quotes. It returns a list of image URLs that contain photos related to our query.

```
from pyspark.ml import PipelineModel

# Number of images Bing will return per query
imgsPerBatch = 2
# A list of offsets, used to page into the search results
offsets = [(i*imgsPerBatch,) for i in range(10)]
# Since web content is our data, we create a dataframe with options on that data: offsets
bingParameters = spark.createDataFrame(offsets, ["offset"])

# Run the Bing Image Search service with our text query
bingSearch = (BingImageSearch()
    .setSubscriptionKey(bingsearch_service_key)
    .setOffsetCol("offset")
    .setQuery("Martin Luther King Jr. quotes")
    .setCount(imgsPerBatch)
    .setOutputCol("images"))

# Transformer that extracts and flattens the richly structured output of Bing Image Search into a simple URL
column
getUrls = BingImageSearch.getUrlTransformer("images", "url")
pipeline_bingsearch = PipelineModel(stages=[bingSearch, getUrls])

# Show the results of your search: image URLs
res_bingsearch = pipeline_bingsearch.transform(bingParameters)
display(res_bingsearch.dropDuplicates())
```

Expected results

IMAGE
http://everydaypowerblog.com/wp-content/uploads/2014/01/Martin-Luther-King-Jr.-Quotes-16.jpg
http://www.scrolldroll.com/wp-content/uploads/2017/06/6-25.png
http://abettertodaymedia.com/wp-content/uploads/2017/01/86783bd7a92960aedd058c91a1d10253.jpg
https://weneedfun.com/wp-content/uploads/2016/05/martin-luther-king-jr-quotes-11.jpg
http://www.sofreshandsogreen.com/wp-content/uploads/2012/01/martin-luther-king-jr-quote-sofreshandsogreendotcom.jpg
https://cdn.quotesgram.com/img/72/57/1104209728-martin_luther_king_jr_quotes_16.jpg
http://comicbookandbeyond.com/wp-content/uploads/2019/05/Martin-Luther-King-Jr.-Quotes.jpg
https://exposingthepain.files.wordpress.com/2015/01/martin-luther-king-jr-quotes-08.png
https://topmemes.me/wp-content/uploads/2020/01/Top-10-Martin-Luther-King-jr.-Quotes2-1024x538.jpg
http://img.picturequotes.com/2/581/580286/dr-martin-luther-king-jr-quote-1-picture-quote-1.jpg
http://parryz.com/wp-content/uploads/2017/06/Amazing-Martin-Luther-King-Jr-Quotes.jpg
http://everydaypowerblog.com/wp-content/uploads/2014/01/Martin-Luther-King-Jr.-Quotes1.jpg
https://lessonslearnedinlife.net/wp-content/uploads/2020/05/Martin-Luther-King-Jr.-Quotes-2020.jpg

IMAGE
`https://quotesblog.net/wp-content/uploads/2015/10/Martin-Luther-King-Jr-Quotes-Wallpaper.jpg`

Anomaly detector sample

Anomaly Detector is great for detecting irregularities in your time series data. In this sample, we use the service to find anomalies in the entire time series.

```
from pyspark.sql.functions import lit

# Create a dataframe with the point data that Anomaly Detector requires
df_timeseriesdata = spark.createDataFrame([
    ("1972-01-01T00:00:00Z", 826.0),
    ("1972-02-01T00:00:00Z", 799.0),
    ("1972-03-01T00:00:00Z", 890.0),
    ("1972-04-01T00:00:00Z", 900.0),
    ("1972-05-01T00:00:00Z", 766.0),
    ("1972-06-01T00:00:00Z", 805.0),
    ("1972-07-01T00:00:00Z", 821.0),
    ("1972-08-01T00:00:00Z", 20000.0), # anomaly
    ("1972-09-01T00:00:00Z", 883.0),
    ("1972-10-01T00:00:00Z", 898.0),
    ("1972-11-01T00:00:00Z", 957.0),
    ("1972-12-01T00:00:00Z", 924.0),
    ("1973-01-01T00:00:00Z", 881.0),
    ("1973-02-01T00:00:00Z", 837.0),
    ("1973-03-01T00:00:00Z", 9000.0) # anomaly
], ["timestamp", "value"]).withColumn("group", lit("series1"))

# Run the Anomaly Detector service to look for irregular data
anamoly_detector = (SimpleDetectAnomalies()
  .setSubscriptionKey(anomalydetector_key)
  .setLocation("eastasia")
  .setTimestampCol("timestamp")
  .setValueCol("value")
  .setOutputCol("anomalies")
  .setGroupbyCol("group")
  .setGranularity("monthly"))

# Show the full results of the analysis with the anomalies marked as "True"
display(anamoly_detector.transform(df_timeseriesdata).select("timestamp", "value", "anomalies.isAnomaly"))
```

Expected results

TIMESTAMP	VALUE	ISANOMALY
1972-01-01T00:00:00Z	826.0	false
1972-02-01T00:00:00Z	799.0	false
1972-03-01T00:00:00Z	890.0	false
1972-04-01T00:00:00Z	900.0	false
1972-05-01T00:00:00Z	766.0	false
1972-06-01T00:00:00Z	805.0	false

TIMESTAMP	VALUE	ISANOMALY
1972-07-01T00:00:00Z	821.0	false
1972-08-01T00:00:00Z	20000.0	true
1972-09-01T00:00:00Z	883.0	false
1972-10-01T00:00:00Z	898.0	false
1972-11-01T00:00:00Z	957.0	false
1972-12-01T00:00:00Z	924.0	false
1973-01-01T00:00:00Z	881.0	false
1973-02-01T00:00:00Z	837.0	false
1973-03-01T00:00:00Z	9000.0	true

Speech-to-text sample

The Speech-to-text service converts streams or files of spoken audio to text. In this sample, we transcribe one audio file to text.

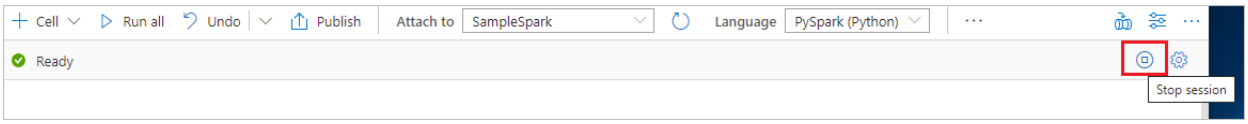
```
# Create a dataframe with our audio URLs, tied to the column called "url"
df = spark.createDataFrame([("https://mmlspark.blob.core.windows.net/datasets/Speech/audio2.wav",)
                            ], ["url"])

# Run the Speech-to-text service to translate the audio into text
speech_to_text = (SpeechToTextSDK()
    .setSubscriptionKey(service_key)
    .setLocation("northeurope") # Set the location of your cognitive service
    .setOutputCol("text")
    .setAudioDataCol("url")
    .setLanguage("en-US")
    .setProfanity("Masked"))

# Show the results of the translation
display(speech_to_text.transform(df).select("url", "text.DisplayText"))
```

Expected results

URL	DISPLAYTEXT
`https://mmlspark.blob.core.windows.net/datasets/Speech/audio2.wav`	Custom Speech provides tools that allow you to visually inspect the recognition quality of a model by comparing audio data with the corresponding recognition result from the custom speech portal. You can playback uploaded audio and determine if the provided recognition result is correct. This tool allows you to quickly inspect quality of Microsoft's baseline speech to text model or a trained custom model without having to transcribe any audio data.

Clean up resources

To ensure the Spark instance is shut down, end any connected sessions(notebooks). The pool shuts down when the **idle time** specified in the Apache Spark pool is reached. You can also select **stop session** from the status bar at the upper right of the notebook.



Next steps

- Check out Synapse sample notebooks
- SynapseML GitHub Repo

QuickStart:Connect a Synapse workspace to a Microsoft Purview account

8/19/2022 • 4 minutes to read • Edit Online

In this quickstart, you will register a Microsoft Purview Account to a Synapse workspace. That connection allows you to discover Microsoft Purview assets, interact with them through Synapse capabilities, and push lineage information to Microsoft Purview.

You can perform the following tasks in Synapse:

- Use the search box at the top to find Microsoft Purview assets based on keywords
- Understand the data based on metadata, lineage, annotations
- Connect those data to your workspace with linked services or integration datasets
- Analyze those datasets with Synapse Apache Spark, Synapse SQL, and Data Flow
- Execute pipelines and push lineage information to Microsoft Purview

Prerequisites

- Microsoft Purview account
- Synapseworkspace

Permissions for connecting a Microsoft Purview account

To connect a Microsoft Purview Account to a Synapse workspace, you need a **Contributor** role in Synapse workspace from Azure portal IAM and you need access to that Microsoft Purview Account. For more information, see Microsoft Purview permissions.

Connect a Microsoft Purview account

Follow the steps to connect a Microsoft Purview account:

1. Gotohttps://web.azuresynapse.net and sign in to your Synapse workspace.
2. Go to **Manage** -> **Microsoft Purview**, select **Connect to a Microsoft Purview account**.
3. You can choose **From Azure subscription** or **Enter manually**. **From Azure subscription**, you can select the account that you have access to.
4. Once connected, you can see the name of the Microsoft Purview account in the tab **Microsoft Purview account**.

If your Microsoft Purview account is protected by firewall, create the managed private endpoints for Microsoft Purview. Learn more about how to let Azure Synapse access a secured Microsoft Purview account. You can either do it during the initial connection or edit an existing connection later.

The Microsoft Purview connection information is stored in the Synapse workspace resource like the following. To establish the connection programmatically, you can update the Synapse workspace and add the `purviewConfiguration` settings.

```
{
    "name": "ContosoSynapseWorkspace",
    "type": "Microsoft.Synapse/workspaces",
    "location": "<region>",
    "properties": {
        ...
        "purviewConfiguration": {
            "purviewResourceId":
"/subscriptions/<subscriptionId>/resourceGroups/<resourceGroupname>/providers/Microsoft.Purview/accounts/<Pu
rviewAccountName>"
        }
    },
    "identity": {...},
    ...
}
```

Set up authentication

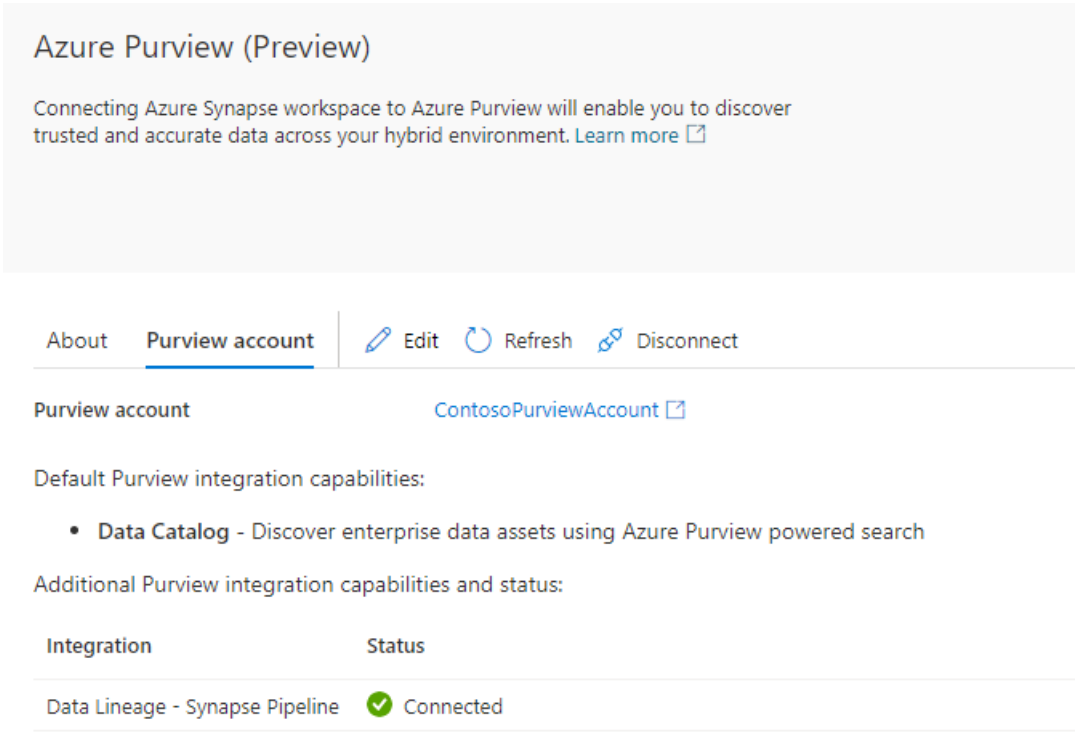
Synapse workspace's managed identity is used to authenticate lineage push operations from Synapse workspace to Microsoft Purview.

Grant the Synapse workspace's managed identity **Data Curator** role on your Microsoft Purview **root collection**. Learn more about Access control in Microsoft Purview and Add roles and restrict access through collections.

When connecting Synapse workspace to Microsoft Purview in Synapse Studio, Synapse tries to add such role assignment automatically. If you have **Collection admins** role on the Microsoft Purview root collection and have access to Microsoft Purview account from your network, this operation is done successfully.

Monitor Microsoft Purview connection

Once you connect the Synapse workspace to a Microsoft Purview account, you see the following page with details on the enabled integration capabilities.



For **Data Lineage - Synapse Pipeline**, you may see one of below status:

- **Connected**: The Synapse workspace is successfully connected to the Microsoft Purview account. Note this indicates Synapse workspace is associated with a Microsoft Purview account and has permission to push lineage to it. If your Microsoft Purview account is protected by firewall, you also need to make sure the integration runtime used to execute the activities and conduct lineage push can reach the Microsoft Purview account. Learn more from Access a secured Microsoft Purview account.
- **Disconnected**: The Synapse workspace cannot push lineage to Microsoft Purview because Microsoft Purview Data Curator role is not granted to Synapse workspace's managed identity. To fix this issue, go to your Microsoft Purview account to check the role assignments, and manually grant the role as needed. Learn more from Set up authentication section.
- **Unknown**: Azure Synapse cannot check the status. Possible reasons are:
  - Cannot reach the Microsoft Purview account from your current network because the account is protected by firewall. You can launch the Synapse Studio from a private network that has connectivity to your Microsoft Purview account instead.
  - You don't have permission to check role assignments on the Microsoft Purview account. You can contact the Microsoft Purview account admin to check the role assignments for you. Learn about the needed Microsoft Purview role from Set up authentication section.

Report lineage to Microsoft Purview

Once you connect the Synapse workspace to a Microsoft Purview account, when you execute pipelines, Synapse reports lineage information to the Microsoft Purview account. For detailed supported capabilities and an end to end walkthrough, see Metadata and lineage from Azure Synapse Analytics.

Discover and explore data using Microsoft Purview

Once you connect the Synapse workspace to a Microsoft Purview account, you can use the search bar at the top center of Synapse workspace to search for data and perform actions. Learn more from Discover, connect and explore data in Synapse using Microsoft Purview.

Nextsteps

Discover, connect and explore data in Synapse using Microsoft Purview

Metadata and lineage from Azure Synapse Analytics

Access a secured Microsoft Purview account

Register and scan Azure Synapse assets in Microsoft Purview

Get lineage from Power BI into Microsoft Purview

Connect Azure Data Share and Microsoft Purview

Access a secured Microsoft Purview account from Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

This article describes how to access a secured Microsoft Purview account from Azure Synapse Analytics for different integration scenarios.

Microsoft Purview private endpoint deployment scenarios

You can use Azure private endpoints for your Microsoft Purview accounts to allow secure access from a virtual network (VNet) to the catalog over a Private Link. Microsoft Purview provides different types of private points for various access need: *account* private endpoint, *portal* private endpoint, and *ingestion* private endpoints. Learn more from Microsoft Purview private endpoints conceptual overview.

If your Microsoft Purview account is protected by firewall and denies public access, make sure you follow below checklist to set up the private endpoints so Synapse can successfully connect to Microsoft Purview.

SCENARIO	REQUIRED MICROSOFT PURVIEW PRIVATE ENDPOINTS
Run pipeline and report lineage to Microsoft Purview	For Synapse pipeline to push lineage to Microsoft Purview, Microsoft Purview ***account*** and ***ingestion*** private endpoints are required. - When using **Azure Integration Runtime**, follow the steps in Managed private endpoints for Microsoft Purview section to create managed private endpoints in the Synapse managed virtual network. - When using **Self-hosted Integration Runtime**, follow the steps in this section to create the *account* and *ingestion* private endpoints in your integration runtime's virtual network.
Discover and explore data using Microsoft Purview on Synapse Studio	To use the search bar at the top center of Synapse Studio to search for Microsoft Purview data and perform actions, you need to create Microsoft Purview ***account*** and ***portal*** private endpoints in the virtual network that you launch the Synapse Studio. Follow the steps in Enable *account* and *portal* private endpoint.

Managed private endpoints for Microsoft Purview

Managed private endpoints are private endpoints created a Managed Virtual Network associated with your Azure Synapse workspace. When you run pipeline and report lineage to a firewall protected Microsoft Purview account, make sure your Synapse workspace is created with "Managed virtual network" option enabled, then create the Microsoft Purview ***account*** and ***ingestion*** managed private endpoints as follows.

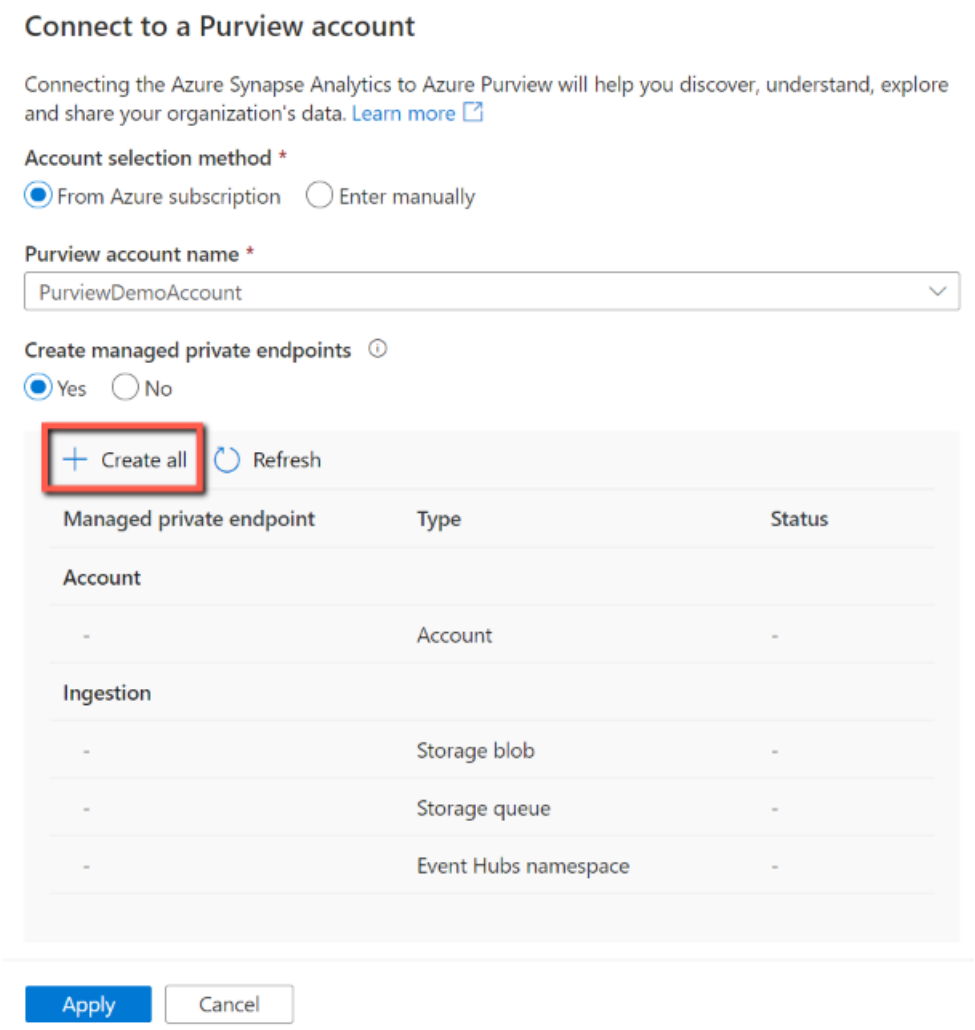
Create managed private endpoints

To create managed private endpoints for Microsoft Purview on Synapse Studio:

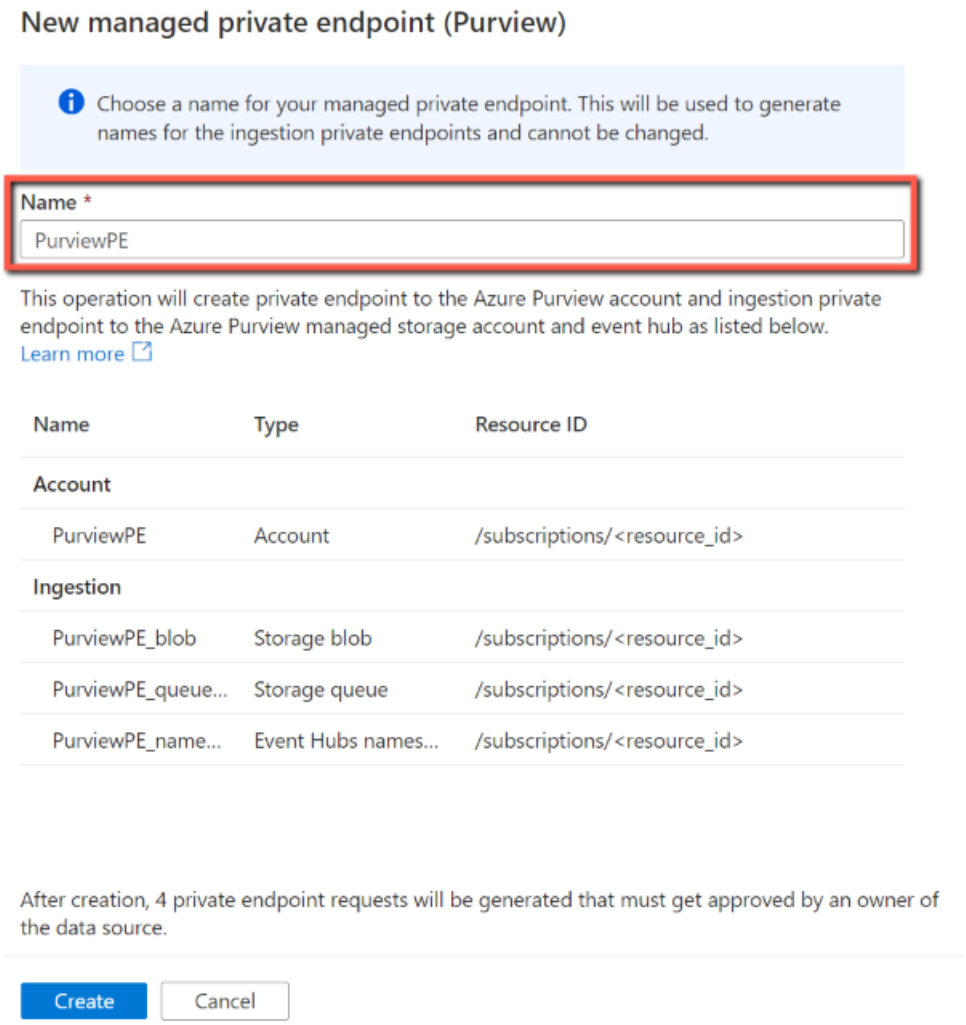
1. Go to **Manage** -> **Microsoft Purview**, and click **Edit** to edit your existing connected Microsoft Purview account or click **Connect to a Microsoft Purview account** to connect to a new Microsoft Purview account.

2. Select **Yes** for **Create managed private endpoints**. You need to have "**workspaces/managedPrivateEndpoint/write**" permission, e.g. Synapse Administrator or Synapse Linked Data Manager role.

3. Click **+ Create all** button to batch create the needed Microsoft Purview private endpoints, including the ***account*** private endpoint and the ***ingestion*** private endpoints for the Microsoft Purview managed resources - Blob storage, Queue storage, and Event Hubs namespace. You need to have at least **Reader** role on your Microsoft Purview account for Synapse to retrieve the Microsoft Purview managed resources' information.



4. In the next page, specify a name for the private endpoint. It will be used to generate names for the ingestion private endpoints as well with suffix.



5. Click **Create** to create the private endpoints. After creation, 4 private endpoint requests will be generated that must get approved by an owner of Microsoft Purview.

Such batch managed private endpoint creation is provided on the Synapse Studio only. If you want to create the managed private endpoints programmatically, you need to create those PEs individually. You can find Microsoft Purview managed resources' information from Azure portal -> your Microsoft Purview account -> Managed resources.

Approve private endpoint connections

After you create the managed private endpoints for Microsoft Purview, you see "Pending" state first. The Microsoft Purview owner need to approve the private endpoint connections for each resource.

If you have permission to approve the Microsoft Purview private endpoint connection, from Synapse Studio:

1. Go to **Manage** -> **Microsoft Purview** -> **Edit**
2. In the private endpoint list, click the **Edit** (pencil) button next to each private endpoint name
3. Click **Manage approvals in Azure portal** which will bring you to the resource.
4. On the given resource, go to **Networking** -> **Private endpoint connection** to approve it. The private endpoint is named as `data_factory_name.your_defined_private_endpoint_name` with description as "Requested by data_factory_name".
5. Repeat this operation for all private endpoints.

If you don't have permission to approve the Microsoft Purview private endpoint connection, ask the Microsoft Purview account owner to do as follows.

- For *account* private endpoint, go to Azure portal -> your Microsoft Purview account -> Networking -> Private endpoint connection to approve.

- For *ingestion* private endpoints, go to Azure portal -> your Microsoft Purview account -> Managed resources, click into the Storage account and Event Hubs namespace respectively, and approve the private endpoint connection in Networking -> Private endpoint connection page.

Monitor managed private endpoints

You can monitor the created managed private endpoints for Microsoft Purview at two places:

- Go to **Manage** -> **Microsoft Purview** -> **Edit** to open your existing connected Microsoft Purview account. To see all the relevant private endpoints, you need to have at least **Reader** role on your Microsoft Purview account for Synapse to retrieve the Microsoft Purview managed resources' information. Otherwise, you only see *account* private endpoint with warning.
- Go to **Manage** -> **Managed private endpoints** where you see all the managed private endpoints created under the Synapse workspace. If you have at least **Reader** role on your Microsoft Purview account, you see Microsoft Purview relevant private endpoints being grouped together. Otherwise, they show up separately in the list.

Nextsteps

- Connect Synapse workspace to Microsoft Purview
- Metadata and lineage from Azure Synapse Analytics
- Discover, connect and explore data in Synapse using Microsoft Purview

Discover, connect, and explore data in Synapse using Microsoft Purview

8/19/2022 • 2 minutes to read • Edit Online

In this document,you will learn the type of interactions that you can perform when registering a Microsoft Purview Account into Synapse.

Prerequisites

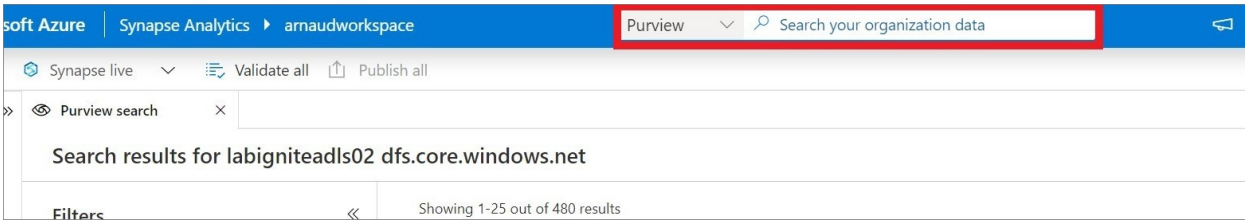
- Microsoft Purview account
- Synapseworkspace
- Connect a Microsoft Purview Account into Synapse

Using Microsoft Purview in Synapse

The use Microsoft Purview in Synapse requires you to have access to that Microsoft Purview account. Synapse passes-through your Microsoft Purview permission. As an example, if you have a curator permission role, you will be able to edit metadata scanned by Microsoft Purview.

Data discovery: search datasets

To discover data registered and scanned by Microsoft Purview, you can use the Search bar at the top center of Synapse workspace. Make sure that you select Microsoft Purview to search for all of your organization data.



Microsoft Purview actions

Here is a list of the Microsoft Purview features that are available in Synapse:

- **Overview** of the metadata
- View and edit **schema** of the metadata with classifications, glossary terms, data types, and descriptions
- View **lineage** to understand dependencies and do impact analysis. For more information about, see lineage
- View and edit **Contacts** to know who is an owner or expert over a dataset
- **Related** to understand the hierarchical dependencies of a specific dataset. This experience is helpful to browse through data hierarchy.
- Click **Request Access** to request access to the data by triggering the self-service data access workflow configured by Workflow Admin in Microsoft Purview. For more information see Request access and Self-Service data access workflow.

Actions that you can perform over datasets with Synapse resources

Connect data to Synapse

- You can create a **new linked service** to Synapse. That action will be required to copy data to Synapse or have them in your data hub (for supported data sources like ADLSg2)

- For objects like files, folders, or tables, you can directly create a **new integration dataset** and leverage an existing linked service if already created

We are not yet able to infer if there is an existing linked service or integration dataset.

Develop in Synapse

There are three actions that you can perform: **New SQL Script**, **New Notebook**, and **New Data Flow**.

With **New SQL Script**, depending on the type of support, you can:

- View the top 100 rows in order to understand the shape of the data.
- Create an external table from Synapse SQL database
- Load the data into a Synapse SQL database

With **New notebook**, you can:

- Load data into a Spark DataFrame
- Create a Spark Table (if you do that over Parquet format, it also creates a serverless SQL pool table).

With **New data flow**, you can create an integration dataset that can be used a source in a data flow pipeline. Data flow is a no-code developer capability to perform data transformation. For more information about using data flow in Synapse.

Nextsteps

- Register and scan Azure Synapse assets in Microsoft Purview
- How to Search Data in Microsoft Purview Data Catalog

Plan and manage costs for Azure Synapse Analytics

8/19/2022 • 7 minutes to read • Edit Online

This article describes how you plan for and manage costs for Azure Synapse Analytics. First, you use the Azure pricing calculator to help plan forAzure Synapse Analytics costs before you add any resources for the service to estimate costs. Next, as you add Azure resources, review the estimated costs. After you've started using Azure Synapse Analytics resources, use Cost Management features to set budgets and monitor costs. You can also review forecasted costs and identify spending trends to identify areas where you might want to act. Costs for Azure Synapse Analytics are only a portion of the monthly costs in your Azure bill. Although this article explains how to plan for and manage costs for Azure Synapse Analytics, you're billed for all Azure services and resources used in your Azure subscription, including the third-party services.

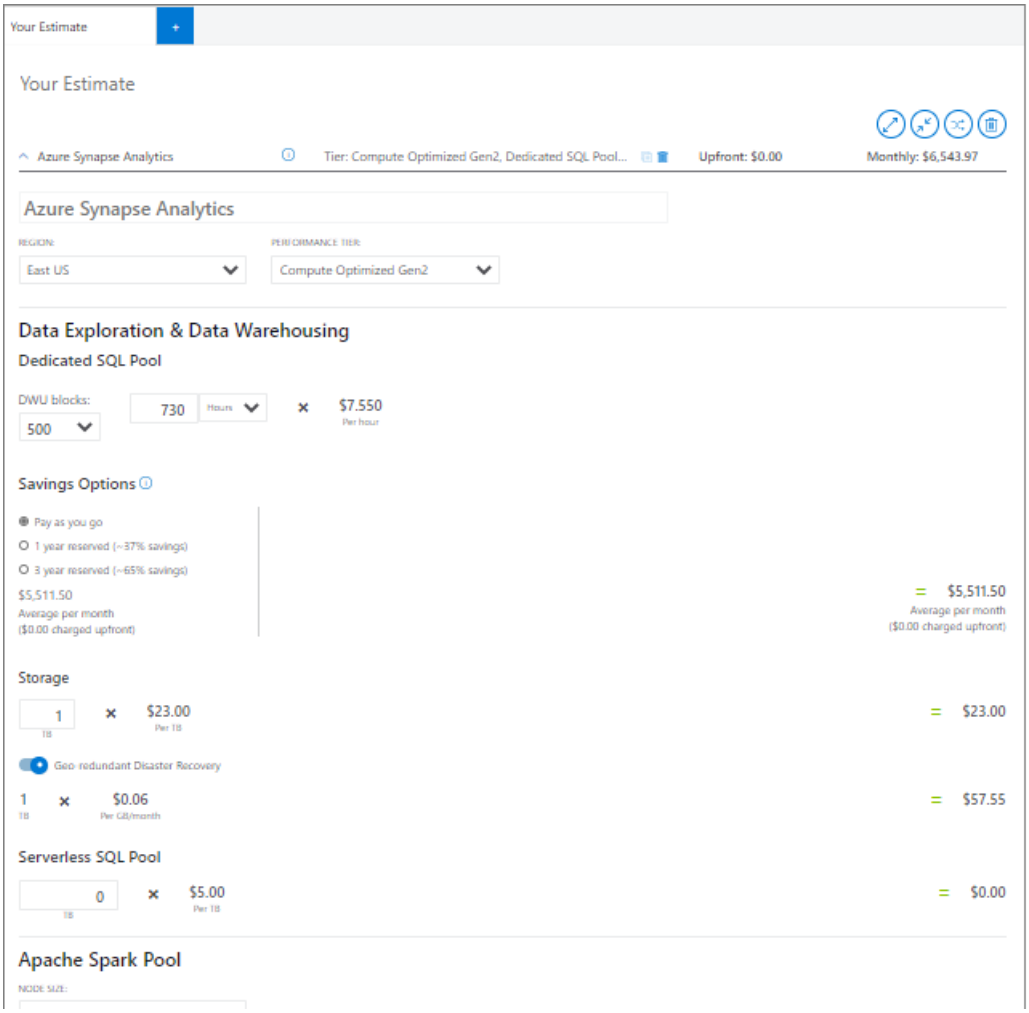
Prerequisites

Cost analysis in Cost Management supports most Azure account types, but not all of them. To view the full list of supported account types, see Understand Cost Management data. To view cost data, you need at least read access for an Azure account. For information about assigning access to Azure Cost Management data, see Assign access to data.

Estimate costs before using Azure Synapse Analytics

Use the Azure pricing calculator to estimate costs before you add Azure Synapse Analytics.

Azure Synapse has various resources that have different charges as seen in the cost estimate below.



Small (4 vCPU(s) / 32 GB)

2 Instances × 730 Hours × $0.16 Per vCore-hour = $951.92

Data Integration

Data Pipelines

Azure-Hosted $183.50

Orchestration

1 Activity Runs (X 1,000) × $1.00 Per 1,000 runs = $1.00

Data Movement

1 Data Integration Unit × 730 Hours × $0.25 Per DIU-hour = $182.50

Pipeline Activity Executions

Integration Runtime

1 Integration Hours × $0.00 Per hour = $0.00

External Integration Runtime

1 Integration Hours × $0.00 Per hour = $0.00

Azure-Hosted Managed VNET $343.50

Orchestration

1 Activity Runs (X 1,000) × $1.00 Per 1,000 runs = $1.00

Data Movement

1 Data Integration Unit × 730 Hours × $0.25 Per DIU-hour = $182.50

Pipeline Activity Executions

Integration Runtime

80 Integration Hours × $1.00 Per hour = $80.00

External Integration Runtime

80 Integration Hours × $1.00 Per hour = $80.00

Self-Hosted $15.50

Orchestration

5 Activity Runs (X 1,000) × $1.50 Per 1,000 runs = $7.50

Data Movement

80 Hours × $0.10 Per hour = $8.00

Pipeline Activity Executions

Integration Runtime

80 Integration Hours × $0.00 Per hour = $0.00

External Integration Runtime

80 Integration Hours × $0.00 Per hour = $0.00

Data Flows

Data Flows $9,617.02

Compute Optimized vCores

INSTANCE:

8 vCores

5 Instances × 730 Hours = $5,613.70

General Purpose vCores

INSTANCE:

8 vCores

0 Instances × 730 Hours = $0.00

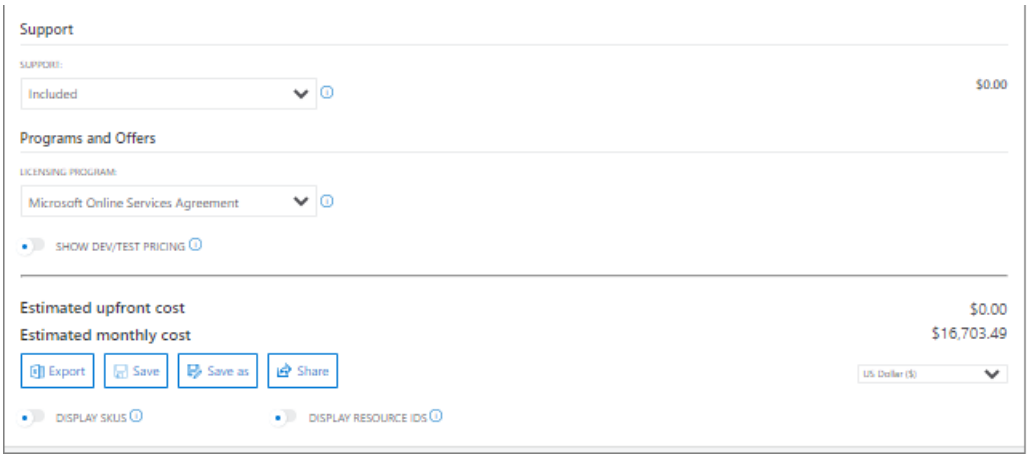
Memory Optimized vCores

INSTANCE:

8 vCores

2 Instances × 730 Hours = $4,003.32

Upfront cost	$0.00
Monthly cost	$16,703.49



Understand the full billing model for Azure Synapse Analytics

Azure Synapse runs on Azure infrastructure that accrues costs along with Azure Synapse when you deploy the new resource. It's important to understand that additional infrastructure might accrue cost.

How you're charged for Azure Synapse Analytics

When you create or use Azure Synapse Analytics resources, you might get charged for the following meters:

- Data Exploration & Data Warehousing
  - Dedicated SQL Pool - You're charged for it based on the number of DWU blocks and hours running.
  - Storage - You're charged for it based on the number of TBs stored.
  - Serverless SQL pool - You're charged by the TB of data processed.
- Apache Spark Pool - You're charged for the number of instances and the hours running.
- Data Integration
  - Orchestration Activity Runs - You are charged based on the number of activity runs.
  - Data Movement – For copy activities run on Azure Integration Runtime, you are charged based on number of DIU used and execution duration.
  - Data Flows vCore Hours – for data flow execution and debugging, you are charged for based on compute type, number of vCores, and execution duration.

At the end of your billing cycle, the charges for each meter are summed. Your bill or invoice shows a section for all Azure Synapse Analytics costs. There's a separate line item for each meter.

Other costs that might accrue with Azure Synapse Analytics

When you create resources for Azure Synapse, resources for other Azure services are also created. They include:

- Data Lake Storage Gen2

Costs might accrue after resource deletion

After you delete Azure Synapse resources, the following resources might continue to exist. They continue to accrue costs until you delete them.

- Data Lake Storage Gen2

Using Azure Prepayment credit with Azure Synapse

You can pay for Azure Synapse charges with your Azure Prepayment credit. However, you can't use Azure Prepayment credit to pay for charges for third-party products and services including those from the Azure Marketplace.

Pre-purchase plan for Azure Synapse

You can save on your Azure Synapse Analytics costs when you pre-purchase Azure Synapse commit units (SCU) for one year. You can use the pre-purchased SCUs at any time during the purchase term. For more information

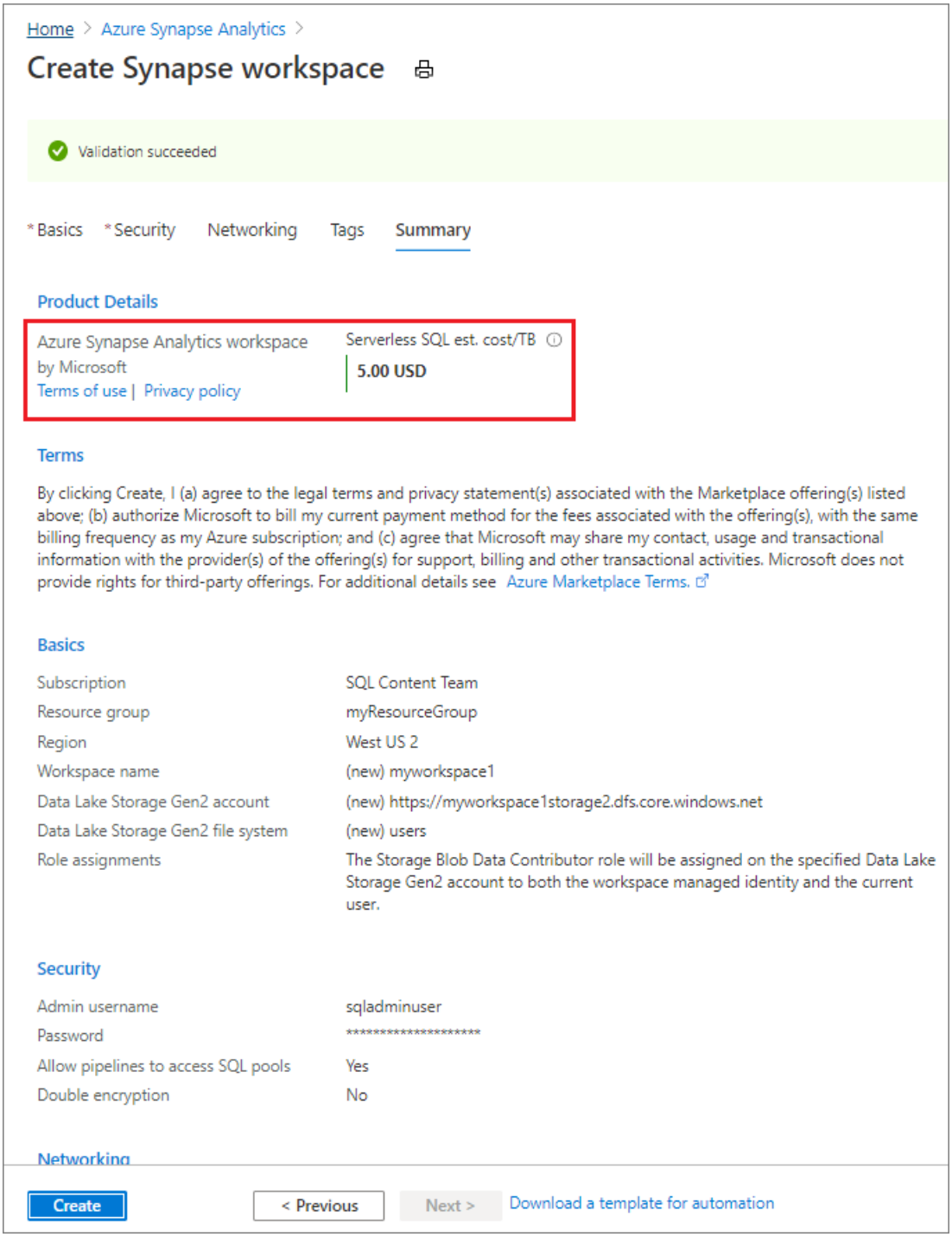
see Optimize Azure Synapse Analytics costs with a Pre-Purchase Plan

Review estimated costs in the Azure portal

As you create resources for Azure Synapse Analytics, you see estimated costs. A workspace has a serverless SQL pool created with the workspace. Serverless SQL pool will not incur charges until you run queries. Other resources, such as dedicated SQL pools and serverless Apache Spark pools, will need to be created within the workspace.

To create an Azure Synapse Analytics workspace and view the estimated price:

1. Navigate to the service in the Azure portal.
2. Create the resource.
3. Review the estimated price shown in the summary.
4. Finish creating the resource.



If your Azure subscription has a spending limit, Azure prevents you from spending over your credit amount. As

you create and use Azure resources, your credits are used. When you reach your credit limit, the resources that you deployed are disabled for the rest of that billing period. You can't change your credit limit, but you can remove it. For more information about spending limits, see Azure spending limit.

Monitor costs

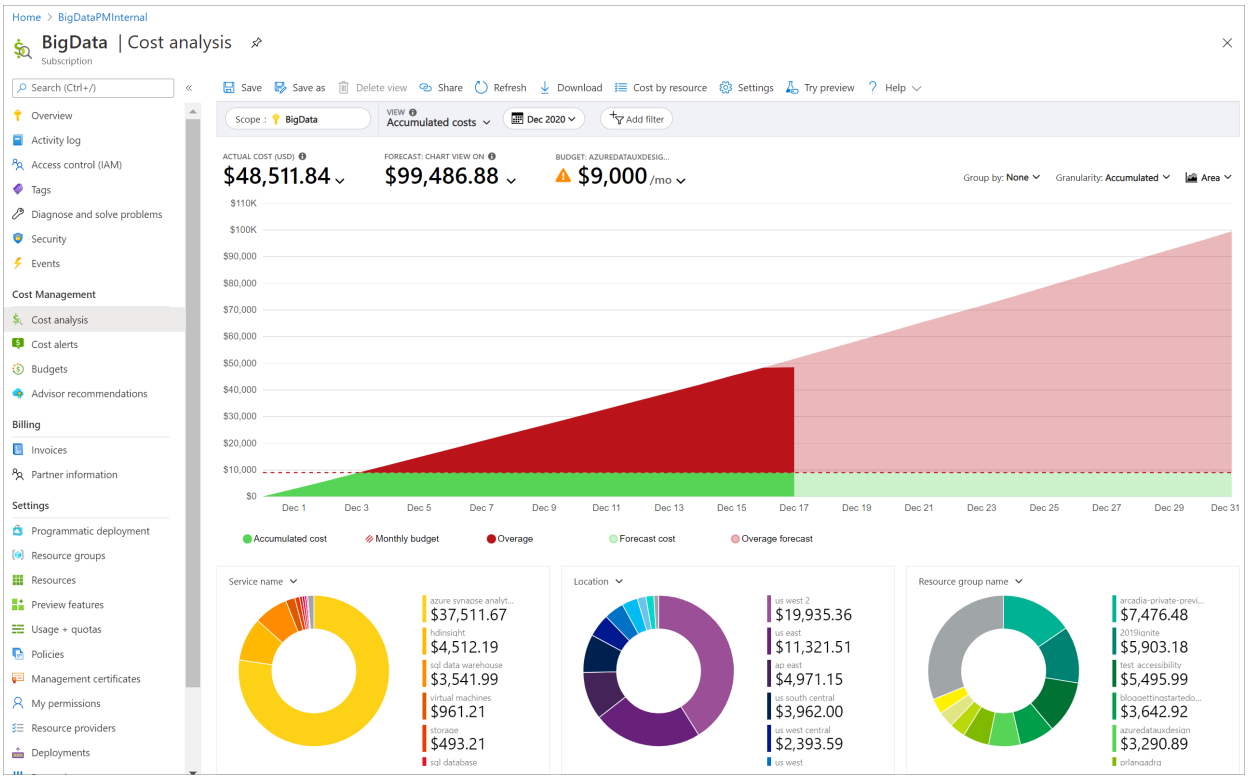
As you use Azure Synapse resources, you incur costs. Azure resource usage unit costs vary by time intervals (seconds, minutes, hours, and days) or by unit usage (bytes, megabytes, and so on.) As soon as you start using resources in Azure Synapse, costs are incurred and you can see the costs in cost analysis.

When you use cost analysis, you view costs for Azure Synapse Analytics in graphs and tables for different time intervals. Some examples are by day, current and prior month, and year. You also view costs against budgets and forecasted costs. Switching to longer views over time can help you identify spending trends. And you see where overspending might have occurred. If you've created budgets, you can also easily see where they're exceeded.

To view Azure Synapse costs in cost analysis:

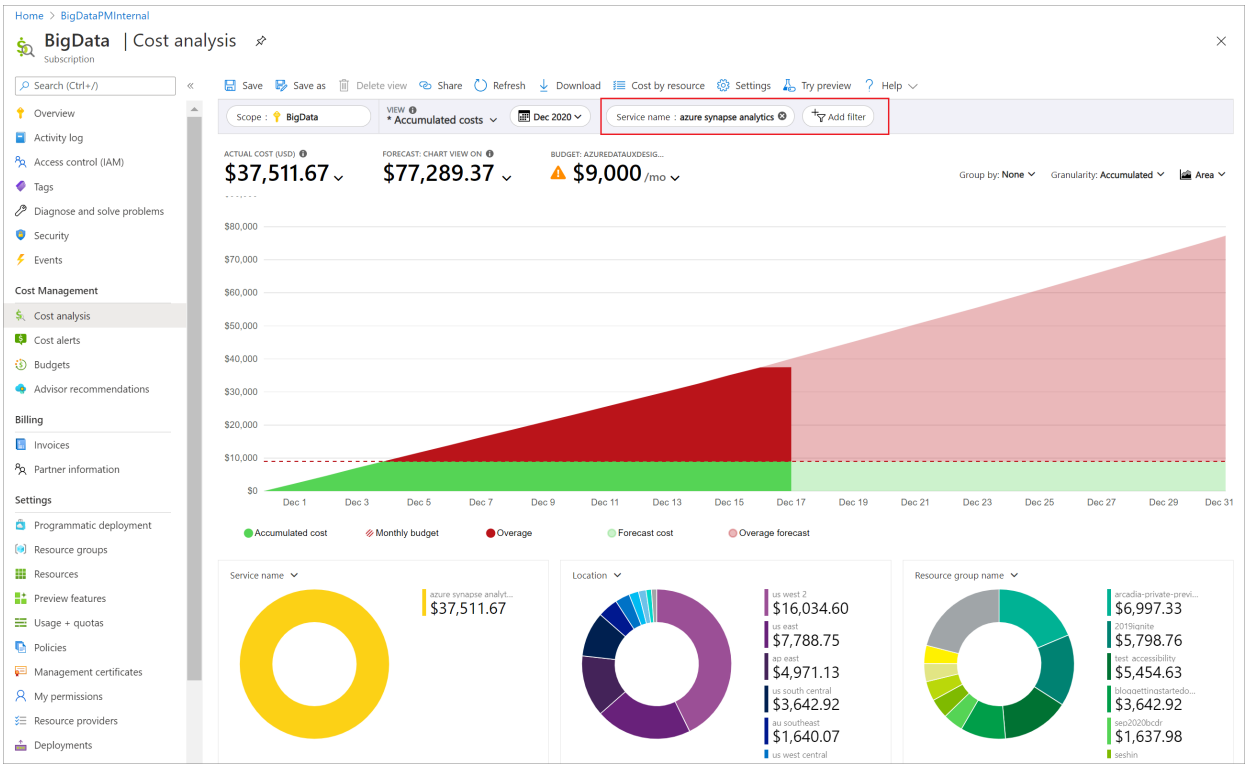
1. Sign in to the Azure portal.
2. Open the scope, either the subscription or the resource group, in the Azure portal and select **Cost analysis** in the menu. For example, go to **Subscriptions**, select a subscription from the list, and then select **Cost analysis** in the menu. Select **Scope** to switch to a different scope in cost analysis.
3. By default, cost for services are shown in the first donut chart. Select the area in the chart labeled Azure Synapse.

Actual monthly costs are shown when you initially open cost analysis. Here's an example showing all monthly usage costs.



- To narrow costs for a single service, like Azure Synapse, select **Add filter** and then select **Service name**. Then, select **Azure Synapse Analytics**.

Here's an example showing costs for just Azure Synapse.



In the preceding example, you see the current cost for the service. Costs by Azure regions (locations) and Azure Synapse costs by resource group are also shown. From here, you can explore costs on your own.

Create budgets

You can create budgets to manage costs and create alerts that automatically notify stakeholders of spending anomalies and overspending risks. Alerts are based on spending compared to budget and cost thresholds. Budgets and alerts are created for Azure subscriptions and resource groups, so they're useful as part of an overall cost monitoring strategy.

Budgets can be created with filters for specific resources or services in Azure if you want more granularity present in your monitoring. Filters help ensure that you don't accidentally create new resources that cost you additional money. For more information about the filter options available when you create a budget, see Group and filter options.

Export cost data

You can also export your cost data to a storage account. This is helpful when you need or others to do additional data analysis for costs. For example, a finance team can analyze the data using Excel or Power BI. You can export your costs on a daily, weekly, or monthly schedule and set a custom date range. Exporting cost data is the recommended way to retrieve cost datasets.

Other ways to manage and reduce costs for Azure Synapse

Serverless SQL pool

To learn more about costs for serverless SQL pool see Cost management for serverless SQL pool in Azure Synapse Analytics

Dedicated SQL pool

You can control costs for a dedicated SQL pool by pausing the resource when it is not is use. For example, if you won't be using the database during the night and on weekends, you can pause it during those times, and resume it during the day. For more information, see Pause and resume compute in dedicated SQL pool via the Azure portal

Serverless Apache Spark pool

To control costs for your serverless Apache Spark pool, enable the serverless Apache Spark automatic pause feature and set your timeout value accordingly. When using Synapse Studio for your development, the studio sends a keep alive message to keep the session active, which is also configurable, so set a short timeout value for automatic pause. When you are done, close your session and the Apache Spark pool will automatically pause once the timeout value is reached.

During development, create multiple Apache Spark pool definitions of various sizes. Creating Apache Spark pool definitions are free and you will only be charged for usage. Apache Spark usage in Azure Synapse is charged per vCore hour and prorated by the minute. For example, use small pool sizes for code development and validation while using larger pool sizes for performance testing.

Data integration - pipelines and data flows

To learn more about data integration costs, see Plan and manage costs for Azure Data Factory

Next steps

- Learn how to optimize your cloud investment with Azure Cost Management.
- Learn more about managing costs with cost analysis.
- Learn about how to prevent unexpected costs.
- Take the Cost Management guided learning course.
- Learn about planning and managing costs for Azure Machine Learning

Browse ADLS Gen2 folders (preview) in an Azure Synapse Analytics workspace

8/19/2022 • 2 minutes to read • Edit Online

You can now browse an Azure Data Lake Storage Gen2 (ADLS Gen2) container or folder in your Azure Synapse Analytics workspace by connecting to the container or specific folder in Data Hub. If your organization does not grant users the Storage Blob Data Contributor role on the storage account, but allows creation of POSIX-like access control lists (ACLs) to the container or specific folders, then you can follow the steps listed in this article to browse files and folders in ADLS Gen2.

> **NOTE**
>
> This preview feature enables users to browse ADLS Gen2 folders that they have ACLs to, but does not support upload, download, create, edit, delete, or rename actions on files or folders. Users with the Storage Blob Data Contributor role can perform all actions on ADLS Gen2 folders.

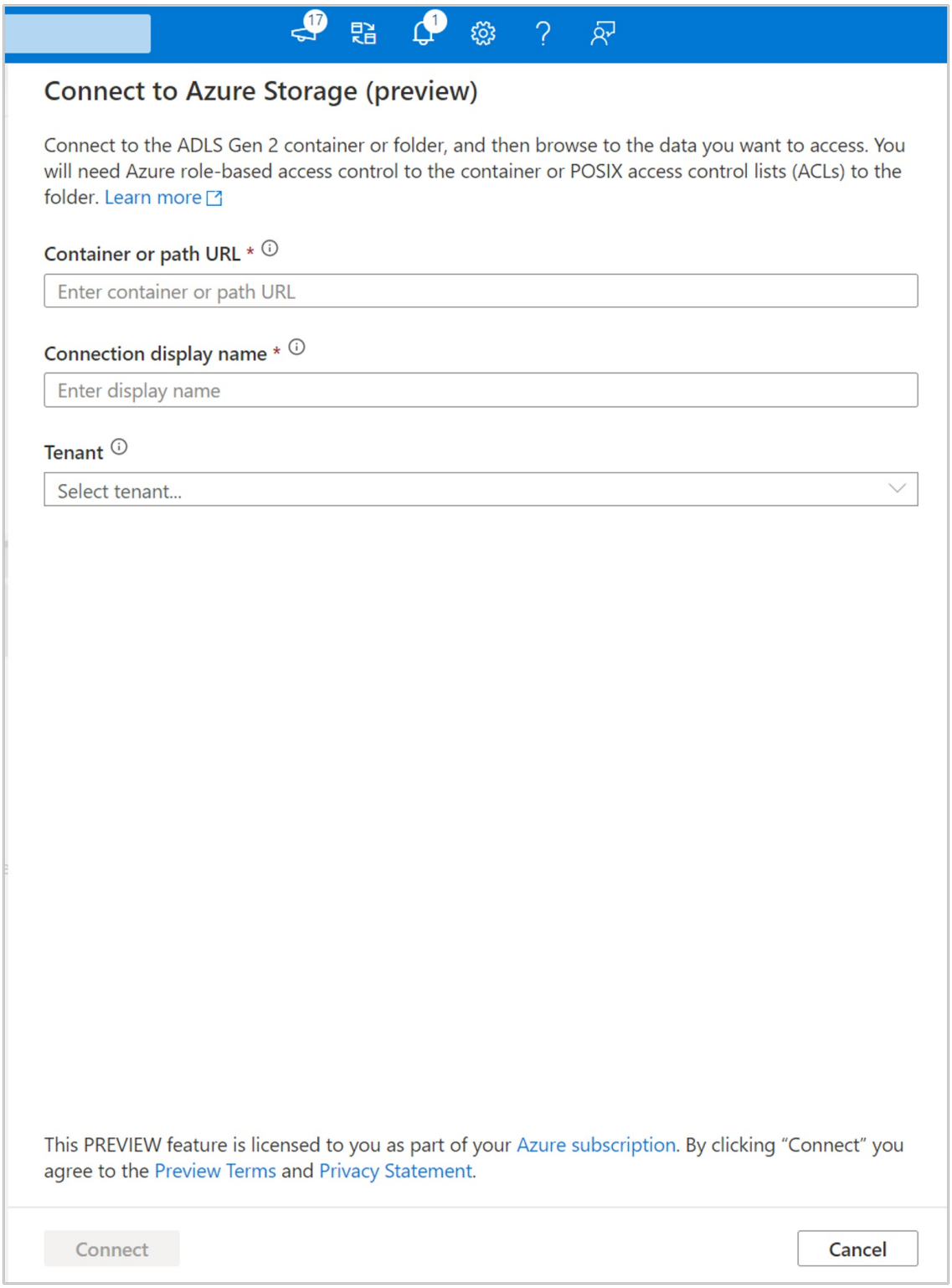
Prerequisites

The following prerequisites must be met prior to connecting a container or folder in Azure Synapse:

- The Storage Blob Data Contributor (Azure RBAC) role or access control lists (ACLs) must be granted to your Azure AD identity.
- A linked service to the ADLS Gen2 container must be created in the Synapse workspace.

Connect ADLS Gen2 folder to your Azure Synapse Analytics workspace

1. Go to the **Linked** tab of Data Hub
2. Right click on **Azure Data Lake Storage Gen2** and select **Connect to Azure Storage (preview)**
   - Provide the ADLS Gen2 container or folder URL in the format `https://storageaccountname.dfs.core.windows.net/containername/foldername/`
   - Provide a unique name for the connection
   - Provide the name of the tenant that contains the ADLS Gen2 account. When left blank, the tenant associated with the Synapse workspace will be used.



3. Click on **Connect**. You will see the connected container or folder under **Attached Containers**.

Next steps

Learn more about ACLs in Azure Data Lake Storage Gen2.

- Access control lists in Azure Data Lake Storage Gen2
- Use the Azure portal to manage ACLs in Azure Data Lake Storage Gen2

Success by design

8/19/2022 • 2 minutes to read • Edit Online

Welcome to the Azure Synapse Customer Success Engineering Success by Design repository.

As part of the Synapse Engineering Group at Microsoft, the Azure Synapse Customer Success Engineering (CSE) team is relied upon to help with complex projects that involve Azure Synapse. As a team of experienced professionals, we provide guidance for all facets of Azure Synapse, including working with customers on architecture reviews, configuration guidance, and performance analysis and recommendations. It's from this experience that the CSE team created this repository of guidance articles to help you produce successful solutions.

This guidance is intended to supplement the official Azure Synapse documentation with additional specialized content that:

- Helps you evaluate Azure Synapse by using a Proof of Concept (POC) project.
- Helps you successfully implement a solution that incorporates Azure Synapse by using our implementation success method. This method can guide you through the key phases of your implementation, including planning, solution development, deployment, and post go-live evaluation.
- Provides detailed guidance on complex topics, including security, networking, troubleshooting, performance tuning, and migration.

> **NOTE**
> This guidance undergoes continuous improvement by the Synapse CSE team. In time, guidance articles will cover additional topics, including enterprise development support using continuous integration and continuous deployment (CI/CD) methods, business continuity and disaster recovery (BCDR), high availability (HA), and Azure Synapse monitoring.

Next steps:

Synapse proof of concept playbook

Synapse implementation success method

Security white paper

Azure Synapse proof of concept playbook

8/19/2022 • 2 minutes to read • Edit Online

Whether it's an enterprise data warehouse migration, a big data re-platforming, or a greenfield implementation, each project traditionally starts with a proof of concept (POC).

The *Synapse proof of concept playbook* is a series of related articles that provide a high-level methodology for planning, preparing, and running an effective Azure Synapse Analytics POC project. The overall objective of a POC is to validate potential solutions to technical problems, such as how to integrate systems or how to achieve certain results through a specific configuration. As emphasized by this series, an effective POC validates that certain concepts have the potential for real-world production application.

> **TIP**
> If you're new to Azure Synapse, we recommend you work through the Introduction to Azure Synapse Analytics module.

Playbook audiences

The playbook helps you to evaluate the use of Azure Synapse when migrating from an existing workload. We designed it for the following audiences:

- Technical experts who are planning their own in-house Azure Synapse POC project.
- Business owners who will be part of the execution or evaluation of an Azure Synapse POC project.
- Anyone looking to learn more about data warehousing POC projects.

Playbook content

The playbook delivers the following content:

- Guidance on what makes an effective POC.
- Guidance on how to make valid comparisons between systems.
- Guidance on the technical aspects of running an Azure Synapse POC.
- A road map to relevant technical content from Azure Synapse.
- Guidance on how to evaluate POC results to support business decisions.
- Guidance on how and where to find additional help.

The playbook includes three subjects:

- Data warehousing with dedicated SQL pool in Azure Synapse Analytics
- Data lake exploration with serverless SQL pool in Azure Synapse Analytics
- Big data analytics with Apache Spark pool in Azure Synapse Analytics

Next steps

Data warehousing with dedicated SQL pool in Azure Synapse Analytics

Data lake exploration with serverless SQL pool in Azure Synapse Analytics

Big data analytics with Apache Spark pool in Azure Synapse Analytics

Azure Synapse Analytics frequently asked questions

Synapse POC playbook: Data warehousing with dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 14 minutes to read • Edit Online

This article presents a high-level methodology for preparing and running an effective Azure Synapse Analytics proof of concept (POC) project for dedicated SQL pool.

> **NOTE**
> This article forms part of the *Azure Synapse proof of concept playbook* series of articles. For an overview of the series, see Azure Synapse proof of concept playbook.

> **TIP**
> If you're new to dedicated SQL pools, we recommend you work through the Work with Data Warehouses using Azure Synapse Analytics learning path.

Prepare for the POC

Before deciding on your Azure Synapse POC goals, we recommend that you first read the Azure Synapse SQL architecture article to familiarize yourself with how a dedicated SQL pool separates compute and storage to provide industry-leading performance.

Identify sponsors and potential blockers

Once you're familiar with Azure Synapse, it's time to make sure that your POC has the necessary support and won't hit any roadblocks. You should:

- Identify any restrictions or policies that your organization has about moving data to, and storing data in, the cloud.
- Identify executive and business sponsorship for a cloud-based data warehouse project.
- Verify that your workload is appropriate for Azure Synapse. For more information, see Dedicated SQL pool architecture in Azure Synapse Analytics.

Set the timeline

A POC is a scoped, time-bounded exercise with specific, measurable goals and metrics that define success. Ideally, it should have some basis in business reality so that the results are meaningful.

POCs have the best outcome when they're *timeboxed*. Timeboxing allocates a fixed and maximum unit of time to an activity. In our experience, two weeks provides enough time to complete the work without the burden of too many use cases or complex test matrices. Working within this fixed time period, we suggest that you follow this timeline:

1. **Data loading:** Three days or less
2. **Querying:** Five days or less
3. **Value added tests:** Two days or less

Here are some tips:

- Make realistic estimates of the time that you will require to complete the tasks in your plan.

- Recognize that the time to complete your POC will be related to the size of your dataset, the number of database objects (tables, views, and stored procedures), the complexity of the database objects, and the number of interfaces you will test.
- If you estimate that your POC will run longer than four weeks, consider reducing the scope to focus only on the most important goals.
- Get support from all the lead resources and sponsors for the timeline before commencing the POC.

Once you've determined that there aren't any immediate obstacles and you've set the timeline, you can scope a high-level architecture.

Create a high-level scoped architecture

A high-level future architecture likely contains many data sources and data consumers, big data components, and possibly machine learning and AI data consumers. To keep your POC goals achievable (and within the bounds of your set timeline), decide which of these components will form part of the POC and which will be excluded.

Additionally, if you're already using Azure, identify the following:

- Any existing Azure resources that you can use during the POC. For example, resources can include Azure Active Directory (Azure AD), or Azure ExpressRoute.
- What Azure region(s) your organization prefers.
- A subscription you can use for non-production POC work.
- The throughput of your network connection to Azure.

  > **IMPORTANT**
  > Be sure to check that your POC can consume some of that throughput without having an adverse effect on production solutions.

Apply migration options

If you're migrating from a legacy data warehouse system to Azure Synapse, here are some questions to consider:

- Are you migrating and want to make as few changes to existing Extract, Transform, and Load (ETL) processes and data warehouse consumption as possible?
- Are you migrating but want to do some extensive improvements along the way?
- Are you building an entirely new data analytics environment (sometimes called a *greenfield project*)?

Next, you need to consider your pain points.

Identify current pain points

Your POC should contain use cases to prove potential solutions to address your current pain points. Here are some questions to consider:

- What gaps in your current implementation do you expect Azure Synapse to fill?
- What new business needs are you required to support?
- What service level agreements (SLAs) are you required to meet?
- What will be the workloads (for example, ETL, batch queries, analytics, reporting queries, or interactive queries)?

Next, you need to set your POC success criteria.

Set POC success criteria

Identify why you're doing a POC and be sure to define clear goals. It's also important to know what outputs you

want from your POC and what you plan to do with them.

Keep in mind that a POC should be a short and focused effort to quickly prove or test a limited set of concepts. If you have a long list of items to prove, you may want to dive them into multiple POCs. POCs can have gates between them so you can determine whether to proceed to the next POC.

Here are some example POC goals:

- We need to know that the query performance for our big complex reporting queries will meet our new SLAs.
- We need to know the query performance for our interactive users.
- We need to know whether our existing ETL processes are a good fit and where improvements need to be made.
- We need to know whether we can shorten our ETL runtimes and by how much.
- We need to know that Synapse Analytics has sufficient security capabilities to adequately secure our data.

Next, you need to create a test plan.

Create a test plan

Using your goals, identify specific tests to run in order to support those goals and provide your identified outputs. It's important to make sure that you have at least one test for each goal and the expected output. Identify specific queries, reports, ETL and other processes that you will run to provide quantifiable results.

Refine your tests by adding multiple testing scenarios to clarify any table structure questions that arise.

Good planning usually defines an effective POC execution. Make sure all stakeholders agree to a written test plan that ties each POC goal to a set of clearly stated test cases and measurements of success.

Most test plans revolve around performance and the expected user experience. What follows is an example of a test plan. It's important to customize your test plan to meet your business requirements. Clearly defining what you are testing will pay dividends later in this process.

GOAL	TEST	EXPECTED OUTCOMES
We need to know that the query performance for our big complex reporting queries will meet our new SLAs	- Sequential test of complex queries - Concurrency test of complex queries against stated SLAs	- Queries A, B, and C completed in 10, 13, and 21 seconds, respectively - With 10 concurrent users, queries A, B, and C completed in 11, 15, and 23 seconds, on average
We need to know the query performance for our interactive users	- Concurrency test of selected queries at an expected concurrency level of 50 users. - Run the preceding query with result set caching	- At 50 concurrent users, average execution time is expected to be under 10 seconds, and without result set caching - At 50 concurrent users, average execution time is expected to be under five seconds with result set caching
We need to know whether our existing ETL processes can run within the SLA	- Run one or two ETL processes to mimic production loads	- Loading incrementally into a core fact table must complete in less than 20 minutes (including staging and data cleansing) - Dimension processing needs to take less than five minutes
We need to know that the data warehouse has sufficient security capabilities to secure our data	- Review and enable network security (VNet and private endpoints), access control (row-level security, dynamic data masking)	- Prove that data never leaves our tenant. - Ensure that customer content is easily secured

Next, you need to identify and validate the POC dataset.

Identify and validate the POC dataset

Using the scoped tests, you can now identify the dataset required to execute those tests in Azure Synapse. Review your dataset by considering the following:

- Verify that the dataset adequately represents your production dataset in terms of content, complexity, and scale.
- Don't use a dataset that's too small (less than 1TB), as you might not achieve representative performance.
- Don't use a dataset that's too large, as the POC isn't intended to complete a full data migration.
- Identify the distribution pattern, indexing option, and partitioning for each table. If there are any questions regarding distribution, indexing, or partitioning, add tests to your POC to answer them. Bear in mind that you may want to test more than one distribution option or indexing option for some tables.
- Check with the business owners for any blockers for moving the POC dataset to the cloud.
- Identify any security or privacy concerns.

> **IMPORTANT**
> Make sure you check with business owners for any blockers before moving any data to the cloud. Identify any security or privacy concerns or any data obfuscation needs that should be done before moving data to the cloud.

Next, you need to assemble the team of experts.

Assemble the team

Identify the team members and their commitment to support your POC. Team members should include:

- A project manager to run the POC project.
- A business representative to oversee requirements and results.
- An application data expert to source the data for the POC dataset.
- An Azure Synapse specialist.
- An expert advisor to optimize the POC tests.
- Any person who will be required for specific POC project tasks but who aren't required for its entire duration. These supporting resources could include network administrators, Azure administrators, or Azure AD administrators.

> **TIP**
> We recommend engaging an expert advisor to assist with your POC. Microsoft's partner community has global availability of expert consultants who can help you assess, evaluate, or implement Azure Synapse.

Now that you are fully prepared, it's time to put your POC into practice.

Put the POC into practice

It's important to keep the following in mind:

- Implement your POC project with the discipline and rigor of any production project.
- Run the POC according to plan.
- Have a change request process in place to prevent your POC scope from growing or changing.

Before tests can start, you need to set up the test environment. It involves four stages:

1. Setup

2. Data loading
3. Querying
4. Value added tests

Setup

You can set up a POC on Azure Synapse by following these steps:

1. Use this quickstart to provision a Synapse workspace and set up storage and permissions according to the POC test plan.
2. Use this quickstart to add a dedicated SQL pool to the Synapse workspace.
3. Set up networking and security according to your requirements.
4. Grant appropriate access to POC team members. See this article about authentication and authorization for accessing dedicated SQL pools.

> **TIP**
> We recommend that you *develop code and unit testing* by using the DW500c service level (or below). We recommend that you *run load and performance tests* by using the DW1000c service level (or above). You can pause compute of the dedicated SQL pool at any time to cease compute billing, which will save on costs.

Data loading

Once you've set up the dedicated SQL pool, you can follow these steps to load data:

1. Load the data into Azure Blob Storage. For a POC, we recommend that you use a general-purpose V2 storage account with locally-redundant storage (LRS). While there are several tools for migrating data to Azure Blob Storage, the easiest way is to use Azure Storage Explorer, which can copy files into a storage container.
2. Load the data into the dedicated SQL pool. Azure Synapse supports two T-SQL loading methods: PolyBase and the COPY statement. You can use SSMS to connect to the dedicated SQL pool to use either method.

When you load data into the dedicated SQL pool for the first time, you need to consider which distribution pattern and index option to use. While a dedicated SQL pool supports a variety of both, it's a best practice to rely on default settings. Default settings use round-robin distribution and a clustered columnstore index. If necessary, you can adjust these settings later, which is described later in this article.

The following example shows the COPY load method:

```
--Note when specifying the column list, input field numbers start from 1
COPY INTO
    test_1 (Col_1 default 'myStringDefault' 1, Col_2 default 1 3)
FROM
    'https://myaccount.blob.core.windows.net/myblobcontainer/folder1/'
WITH (
    FILE_TYPE = 'CSV',
    CREDENTIAL = (IDENTITY = 'Storage Account Key' SECRET = '<Your_Account_Key>'),
    FIELDQUOTE = '"',
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '0x0A',
    ENCODING = 'UTF8',
    FIRSTROW = 2
);
```

Querying

The primary purpose of an data warehouse is to perform analytics, which requires querying the data warehouse. Most POCs start by running a small number of representative queries against the data warehouse, at first sequentially and then concurrently. You should define both approaches in your test plan.

Sequential query tests

It's easy to run sequential query tests in SSMS. It's important to run these tests by using a user with a sufficiently large resource class. A resource class is a pre-determined resource limit in dedicated SQL pool that governs compute resources and concurrency for query execution. For simple queries, we recommend using the pre-defined **staticrc20** resource class. For more complex queries, we recommend using the pre-defined **staticrc40** resource class.

Notice that the following first query uses a query label to provide a mechanism to keep track of the query. The second query uses the `sys.dm_pdw_exec_requests` dynamic management view to search by the label.

```
/* Use the OPTION(LABEL = '') Syntax to add a query label to track the query in DMVs */
SELECT TOP (1000)
    *
FROM
    [dbo].[Date]
OPTION (LABEL = 'Test1');

/* Use sys.dm_pdw_exec_requests to determine query execution duration (ms) */
SELECT
    Total_elapsed_time AS [Elapsed_Time_ms],
    [label]
FROM
    sys.dm_pdw_exec_requests
WHERE
    [label] = 'Test1';
```

Concurrent query tests

After recording sequential query performance, you can then run multiple queries concurrently. That way, you can simulate a business intelligence workload running against the dedicated SQL pool. The easiest way to run this test is to download a stress testing tool. The most popular tool is Apache JMeter, which is a third-party open source tool.

The tool reports on minimum, maximum, and median query durations for a given concurrency level. For example, suppose that you want to simulate a business intelligence workload that generates 100 concurrent queries. You can setup JMeter to run those 100 concurrent queries in a loop and then review the steady state execution. It can be done with result set caching on or off to evaluate the suitability of that feature.

Be sure to document your results. Here's an example of some results:

CONCURRENCY	# QUERIES RUN	DWU	MIN DURATION(S)	MAX DURATION(S)	MEDIAN DURATION(S)
100	1,000	5,000	3	10	5
50	5,000	5,000	3	6	4

Mixed workload tests

Mixed workload testing is an extension of the concurrent query tests. By adding a data loading process into the workload mix, the workload will better simulate a real production workload.

Optimize the data

Depending on the query workload running on Azure Synapse, you may need to optimize your data warehouse's distributions and indexes and rerun the tests. For more information, see Best practices for dedicated SQL pools

in Azure Synapse Analytics.

The most common mistakes seen during setup are:

- Large queries run with a resource class that's too low.
- The dedicated SQL pool service level DWUs are too low for the workload.
- Large tables require hash distribution.

To improve query performance, you can:

- Create materialized views, which can accelerate queries involving common aggregations.
- Replicate tables, especially for small dimension tables.
- Hash distribute large fact tables that are joined or aggregated.

Value added tests

Once query performance testing is complete, it's a good time to test specific features to verify that they satisfy your intended use cases. These features include:

- Row-level security
- Column-level security
- Dynamic data masking
- Intra-cluster scaling via workload isolation

Finally, you need to interpret your POC results.

Interpret the POC results

Once you have test results for your data warehouse, it's important to interpret that data. A common approach you can take is to compare the runs in terms of *price/performance*. Simply put, price/performance removes the differences in price per DWU or service hardware and provides a single comparable number for each performance test.

Here's an example:

TEST	TEST DURATION	DWU	$/HR FOR DWU	COST OF TEST
Test 1	10 min	1000	$12/hr	$2
Test 2	30 min	500	$6/hr	$3

This example makes it easy to see that **Test 1** at DWU1000 is more cost effective at $2 per test run compared with $3 per test run.

> **NOTE**
> You can also use this methodology to compare results *across vendors* in a POC.

In summary, once you complete all the POC tests, you're ready to evaluate the results. Begin by evaluating whether the POC goals have been met and the desired outputs collected. Make note of where additional testing is warranted and additional questions that were raised.

Next steps

Data lake exploration with serverless SQL pool in Azure Synapse Analytics

Big data analytics with Apache Spark pool in Azure Synapse Analytics

Azure Synapse Analytics frequently asked questions

Synapse POC playbook: Data lake exploration with serverless SQL pool in Azure Synapse Analytics

8/19/2022 • 10 minutes to read • Edit Online

This article presents a high-level methodology for preparing and running an effective Azure Synapse Analytics proof of concept (POC) project for serverless SQL pool.

> **NOTE**
> This article forms part of the *Azure Synapse proof of concept playbook* series of articles. For an overview of the series, see Azure Synapse proof of concept playbook.

Prepare for the POC

A POC project can help you make an informed business decision about implementing a big data and advanced analytics environment on a cloud-based platform that leverages serverless SQL pool in Azure Synapse. If you need to explore or gain insights from data in the data lake, or optimize your existing data transformation pipeline, you can benefit from using the serverless SQL pool. It's suitable for the following scenarios:

- **Basic discovery and exploration:** Quickly reason about data stored in various formats (Parquet, CSV, JSON) in your data lake, so you can plan how to unlock insights from it.
- **Logical data warehouse:** Produce a relational abstraction on top of raw or disparate data without relocating or transforming it, providing an always up-to-date view of your data.
- **Data transformation:** Run simple, scalable, and highly performant data lake queries by using T-SQL. You can feed query results to business intelligence (BI) tools, or load them into a relational database. Target systems can include Azure Synapse dedicated SQL pools or Azure SQL Database.

Different professional roles can benefit from serverless SQL pool:

- **Data engineers** can explore the data lake, transform and prepare data by using serverless SQL pool, and simplify their data transformation pipelines.
- **Data scientists** can quickly reason about the contents and structure of the data stored in the data lake by using the OPENROWSET T-SQL function and its automatic schema inference.
- **Data analysts** can write T-SQL queries in their preferred query tools, which can connect to serverless SQL pool. They can explore data in Spark external tables that were created by data scientists or data engineers.
- **BI professionals** can quickly create Power BI reports that connect to data lake or Spark tables.

A serverless SQL pool POC project will identify your key goals and business drivers that serverless SQL pool is designed to support. It will also test key features and gather metrics to support your implementation decisions. A POC isn't designed to be deployed to a production environment. Rather, it's a short-term project that focuses on key questions, and its result can be discarded.

Before you begin planning your serverless SQL Pool POC project:

- Identify any restrictions or guidelines your organization has about moving data to the cloud.
- Identify executive or business sponsors for a big data and advanced analytics platform project. Secure their support for migration to the cloud.
- Identify availability of technical experts and business users to support you during the POC execution.

Before you start preparing for the POC project, we recommend you first read the serverless SQL pool documentation.

> **TIP**
> If you're new to serverless SQL pools, we recommend you work through the Build data analytics solutions using Azure Synapse serverless SQL pools learning path.

Set the goals

A successful POC project requires planning. Start by identify why you're doing a POC to fully understand the real motivations. Motivations could include modernization, cost saving, performance improvement, or integrated experience. Be sure to document clear goals for your POC and the criteria that will define its success. Ask yourself:

- What do you want as the outputs of your POC?
- What will you do with those outputs?
- Who will use the outputs?
- What will define a successful POC?

Keep in mind that a POC should be a short and focused effort to quickly prove a limited set of concepts and capabilities. These concepts and capabilities should be representative of the overall workload. If you have a long list of items to prove, you may want to plan more than one POC. In that case, define gates between the POCs to determine whether you need to continue with the next one. Given the different professional roles that can use a serverless SQL pool (and the different scenarios that serverless SQL pool supports), you may choose to execute multiple POCs. For example, one POC could focus on requirements for the data scientist role, such as discovery and exploration of data in different formats. Another could focus on requirements for the data engineering role, such as data transformation and the creation of a logical data warehouse.

As you consider your POC goals, ask yourself the following questions to help you shape the goals:

- Are you migrating from an existing big data and advanced analytics platform (on-premises or cloud)?
- Are you migrating but want to make as few changes as possible to existing ingestion and data processing?
- Are you migrating but want to do some extensive improvements along the way?
- Are you building an entirely new big data and advanced analytics platform (greenfield project)?
- What are your current pain points? For example, scalability, performance, or flexibility.
- What new business requirements do you need to support?
- What are the SLAs that you're required to meet?
- What will be the workloads? For example, data exploration over different data formats, basic exploration, a logical data warehouse, data preparation and/or transformation, T-SQL interactive analysis, T-SQL querying of Spark tables, or reporting queries over the data lake.
- What are the skills of the users who will own the project (should the POC be implemented)?

Here are some examples of POC goal setting:

- Why are we doing a POC?
  - We need to know if we can explore all of the raw file formats we store by using serverless SQL pool.
  - We need to know if our data engineers can quickly evaluate new data feeds.
  - We need to know if data lake query performance by using serverless SQL pool will meet our data exploration requirements.
  - We need to know if serverless SQL pool is a good choice for some of our visualizations and reporting requirements.
  - We need to know if serverless SQL pool is a good choice for some of our data ingestion and

processing requirements.
    - We need to know if our move to Azure Synapse will meet our budget.
- At the conclusion of this PoC:
    - We will have the data to identify the data transformations that are well suited to serverless SQL pool.
    - We will have the data to identify when serverless SQL pool can be best used during data visualization.
    - We will have the data to know the ease with which our data engineers and data scientists can adopt the new platform.
    - We will have gained insight to better estimate the effort required to complete the implementation or migration project.
    - We will have a list of items that may need more testing.
    - Our POC will be successful if we have the data needed and have completed the testing identified to determine how serverless SQL pool will support our cloud-based big data and advance analytics platform.
    - We will have determined whether we can move to the next phase or whether more POC testing is needed to finalize our decision.
    - We will be able to make a sound business decision supported by specific data points.

Plan the project

Use your goals to identify specific tests and to provide the outputs you identified. It's important to make sure that you have at least one test to support each goal and expected output. Also, identify specific data exploration and analysis tasks, specific transformations, and specific existing processing you want to test. Identify a specific dataset and codebase that you can use.

Here's an example of the needed level of specificity in planning:

- **Goal:** We need to know whether data engineers can achieve the equivalent processing of the existing ETL process named "Daily Batch Raw File Validation" within the required SLA.
- **Output:** We will have the data to determine whether we can use T-SQL queries to execute the "Daily Batch Raw File Validation" ETL process within the required SLA.
- **Test:** Validation queries A, B, and C are identified by data engineering, and they represent overall data processing needs. Compare the performance of these queries with the benchmark obtained from the existing system.

Evaluate the POC dataset

Using the specific tests you identified, select a dataset to support the tests. Take time to review this dataset. You should verify that the dataset will adequately represent your future processing in terms of content, complexity, and scale. Don't use a dataset that's too small because it won't deliver representative performance. Conversely, don't use a dataset that's too large because the POC shouldn't become a full data migration. Be sure to obtain the appropriate benchmarks from existing systems so you can use them for performance comparisons.

> **IMPORTANT**
> Make sure you check with business owners for any blockers before moving any data to the cloud. Identify any security or privacy concerns or any data obfuscation needs that should be done before moving data to the cloud.

Create a high-level architecture

Based upon the high-level architecture of your proposed future state architecture, identify the components that will form part of your POC. Your high-level future state architecture likely contains many data sources, numerous data consumers, big data components, and possibly machine learning and artificial intelligence (AI) data consumers. Your POC architecture should specifically identify components that will be part of the POC. Importantly, it should identify any components that won't form part of the POC testing.

If you're already using Azure, identify any resources you already have in place (Azure Active Directory, ExpressRoute, and others) that you can use during the POC. Also identify the Azure regions your organization uses. Now is a great time to identify the throughput of your ExpressRoute connection and to check with other business users that your POC can consume some of that throughput without adverse impact on production systems.

Identify POC resources

Specifically identify the technical resources and time commitments required to support your POC. Your POC will need:

- A business representative to oversee requirements and results.
- An application data expert, to source the data for the POC and provide knowledge of the existing processes and logic.
- A serverless SQL pool expert.
- An expert advisor, to optimize the POC tests.
- Resources that will be required for specific components of your POC project, but not necessarily required for the duration of the POC. These resources could include network admins, Azure admins, Active Directory admins, Azure portal admins, and others.
- Ensure all the required Azure services resources are provisioned and the required level of access is granted, including access to storage accounts.
- Ensure you have an account that has required data access permissions to retrieve data from all data sources in the POC scope.

> **TIP**
> We recommend engaging an expert advisor to assist with your POC. Microsoft's partner community has global availability of expert consultants who can help you assess, evaluate, or implement Azure Synapse.

Set the timeline

Review your POC planning details and business needs to identify a time frame for your POC. Make realistic estimates of the time that will be required to complete the POC goals. The time to complete your POC will be influenced by the size of your POC dataset, the number and complexity of tests, and the number of interfaces to test. If you estimate that your POC will run longer than four weeks, consider reducing the POC scope to focus on the highest priority goals. Be sure to obtain approval and commitment from all the lead resources and sponsors before continuing.

Put the POC into practice

We recommend you execute your POC project with the discipline and rigor of any production project. Run the project according to plan and manage a change request process to prevent uncontrolled growth of the POC's scope.

Here are some examples of high-level tasks:

1. Create a Synapse workspace, storage accounts, and the Azure resources identified in the POC plan.
2. Set up networking and security according to your requirements.
3. Grant appropriate access to POC team members. See this article about permissions for accessing files directly from Azure Storage.
4. Load the POC dataset.
5. Implement and configure the tests and/or migrate existing code to serverless SQL pool scripts and views.
6. Execute the tests:
    - Many tests can be executed in parallel.

- Record your results in a consumable and readily understandable format.

7. Monitor for troubleshooting and performance.
8. Evaluate your results and present findings.
9. Work with technical stakeholders and the business to plan for the next stage of the project. The next stage could be a follow-up POC or a production implementation.

Interpret the POC results

When you complete all the POC tests, you evaluate the results. Begin by evaluating whether the POC goals were met and the desired outputs were collected. Determine whether more testing is necessary or any questions need addressing.

Next steps

Data lake exploration with dedicated SQL pool in Azure Synapse Analytics

Big data analytics with Apache Spark pool in Azure Synapse Analytics

Build data analytics solutions using Azure Synapse serverless SQL pools

Azure Synapse Analytics frequently asked questions

Synapse POC playbook: Big data analytics with Apache Spark pool in Azure Synapse Analytics

8/19/2022 • 15 minutes to read • Edit Online

This article presents a high-level methodology for preparing and running an effective Azure Synapse Analytics proof of concept (POC) project for Apache Spark pool.

> **NOTE**
> This article forms part of the *Azure Synapse proof of concept playbook* series of articles. For an overview of the series, see Azure Synapse proof of concept playbook.

Prepare for the POC

A POC project can help you make an informed business decision about implementing a big data and advanced analytics environment on a cloud-based platform that leverages Apache Spark pool in Azure Synapse.

A POC project will identify your key goals and business drivers that cloud-based big data and advance analytics platform must support. It will test key metrics and prove key behaviors that are critical to the success of your data engineering, machine learning model building, and training requirements. A POC isn't designed to be deployed to a production environment. Rather, it's a short-term project that focuses on key questions, and its result can be discarded.

Before you begin planning your Spark POC project:

- Identify any restrictions or guidelines your organization has about moving data to the cloud.
- Identify executive or business sponsors for a big data and advanced analytics platform project. Secure their support for migration to the cloud.
- Identify availability of technical experts and business users to support you during the POC execution.

Before you start preparing for the POC project, we recommend you first read the Apache Spark documentation.

> **TIP**
> If you're new to Spark pools, we recommend you work through the Perform data engineering with Azure Synapse Apache Spark Pools learning path.

By now you should have determined that there are no immediate blockers and then you can start preparing for your POC. If you are new to Apache Spark Pools in Azure Synapse Analytics you can refer to this documentation where you can get an overview of the Spark architecture and learn how it works in Azure Synapse.

Develop an understanding of these key concepts:

- Apache Spark and its distributed architecture.
- Spark concepts like Resilient Distributed Datasets (RDD) and partitions (in-memory and physical).
- Azure Synapse workspace, the different compute engines, pipeline, and monitoring.
- Separation of compute and storage in Spark pool.
- Authentication and authorization in Azure Synapse.
- Native connectors that integrate with Azure Synapse dedicated SQL pool, Azure Cosmos DB, and others.

Azure Synapse decouples compute resources from storage so that you can better manage your data processing needs and control costs. The serverless architecture of Spark pool allows you to spin up and down as well as grow and shrink your Spark cluster, independent of your storage. You can pause (or setup auto-pause) a Spark cluster entirely. That way, you pay for compute only when it's in use. When it's not in use, you only pay for storage. You can scale up your Spark cluster for heavy data processing needs or large loads and then scale it back down during less intense processing times (or shut it down completely). You can effectively scale and pause a cluster to reduce costs. Your Spark POC tests should include data ingestion and data processing at different scales (small, medium, and large) to compare price and performance at different scale. For more information, see Automatically scale Azure Synapse Analytics Apache Spark pools.

It's important to understand the difference between the different sets of Spark APIs so you can decide what works best for your scenario. You can choose the one that provides better performance or ease of use, taking advantage of your team's existing skill sets. For more information, see A Tale of Three Apache Spark APIs: RDDs, DataFrames, and Datasets.

Data and file partitioning work slightly differently in Spark. Understanding the differences will help you to optimize for performance. For more information, see Apache Spark documentation: Partition Discovery and Partition Configuration Options.

Set the goals

A successful POC project requires planning. Start by identify why you're doing a POC to fully understand the real motivations. Motivations could include modernization, cost saving, performance improvement, or integrated experience. Be sure to document clear goals for your POC and the criteria that will define its success. Ask yourself:

- What do you want as the outputs of your POC?
- What will you do with those outputs?
- Who will use the outputs?
- What will define a successful POC?

Keep in mind that a POC should be a short and focused effort to quickly prove a limited set of concepts and capabilities. These concepts and capabilities should be representative of the overall workload. If you have a long list of items to prove, you may want to plan more than one POC. In that case, define gates between the POCs to determine whether you need to continue with the next one. Given the different professional roles that may use Spark pools and notebooks in Azure Synapse, you may choose to execute multiple POCs. For example, one POC could focus on requirements for the data engineering role, such as ingestion and processing. Another POC could focus on machine learning (ML) model development.

As you consider your POC goals, ask yourself the following questions to help you shape the goals:

- Are you migrating from an existing big data and advanced analytics platform (on-premises or cloud)?
- Are you migrating but want to make as few changes as possible to existing ingestion and data processing? For example, a Spark to Spark migration, or a Hadoop/Hive to Spark migration.
- Are you migrating but want to do some extensive improvements along the way? For example, re-writing MapReduce jobs as Spark jobs, or converting legacy RDD-based code to DataFrame/Dataset-based code.
- Are you building an entirely new big data and advanced analytics platform (greenfield project)?
- What are your current pain points? For example, scalability, performance, or flexibility.
- What new business requirements do you need to support?
- What are the SLAs that you're required to meet?
- What will be the workloads? For example, ETL, batch processing, stream processing, machine learning model training, analytics, reporting queries, or interactive queries?
- What are the skills of the users who will own the project (should the POC be implemented)? For example, PySpark vs Scala skills, notebook vs IDE experience.

Here are some examples of POC goal setting:

- Why are we doing a POC?
  - We need to know that the data ingestion and processing performance for our big data workload will meet our new SLAs.
  - We need to know whether near real-time stream processing is possible and how much throughput it can support. (Will it support our business requirements?)
  - We need to know if our existing data ingestion and transformation processes are a good fit and where improvements will need to be made.
  - We need to know if we can shorten our data integration run times and by how much.
  - We need to know if our data scientists can build and train machine learning models and leverage AI/ML libraries as needed in a Spark pool.
  - Will the move to cloud-based Synapse Analytics meet our cost goals?
- At the conclusion of this POC:
  - We will have the data to determine if our data processing performance requirements can be met for both batch and real-time streaming.
  - We will have tested ingestion and processing of all our different data types (structured, semi and unstructured) that support our use cases.
  - We will have tested some of our existing complex data processing and can identify the work that will need to be completed to migrate our portfolio of data integration to the new environment.
  - We will have tested data ingestion and processing and will have the data points to estimate the effort required for the initial migration and load of historical data, as well as estimate the effort required to migrate our data ingestion (Azure Data Factory (ADF), Distcp, Databox, or others).
  - We will have tested data ingestion and processing and can determine if our ETL/ELT processing requirements can be met.
  - We will have gained insight to better estimate the effort required to complete the implementation project.
  - We will have tested scale and scaling options and will have the data points to better configure our platform for better price-performance settings.
  - We will have a list of items that may need more testing.

Plan the project

Use your goals to identify specific tests and to provide the outputs you identified. It's important to make sure that you have at least one test to support each goal and expected output. Also, identify specific data ingestion, batch or stream processing, and all other processes that will be executed so you can identify a very specific dataset and codebase. This specific dataset and codebase will define the scope of the POC.

Here's an example of the needed level of specificity in planning:

- **Goal A:** We need to know whether our requirement for data ingestion and processing of batch data can be met under our defined SLA.
- **Output A:** We will have the data to determine whether our batch data ingestion and processing can meet the data processing requirement and SLA.
  - **Test A1:** Processing queries A, B, and C are identified as good performance tests as they are commonly executed by the data engineering team. Also, they represent overall data processing needs.
  - **Test A2:** Processing queries X, Y, and Z are identified as good performance tests as they contain near real-time stream processing requirements. Also, they represent overall event-based stream processing needs.
  - **Test A3:** Compare the performance of these queries at different scale of the Spark cluster (varying number of worker nodes, size of the worker nodes - like small, medium, and large - number and size of executors) with the benchmark obtained from the existing system. Keep the *law of diminishing*

returns in mind; adding more resources (either by scaling up or scaling out) can help to achieve parallelism, however there's a certain limit that's unique to each scenario to achieve the parallelism. Discover the optimal configuration for each identified use case in your testing.

- **Goal B:** We need to know if our data scientists can build and train machine learning models on this platform.
- **Output B:** We will have tested some of our machine learning models by training them on data in a Spark pool or a SQL pool, leveraging different machine learning libraries. These tests will help to determine which machine learning models can be migrated to the new environment
  - **Test B1:** Specific machine learning models will be tested.
  - **Test B2:** Test base machine learning libraries that come with Spark (Spark MLLib) along with an additional library that can be installed on Spark (like scikit-learn) to meet the requirement.
- **Goal C:** We will have tested data ingestion and will have the data points to:
  - Estimate the effort for our initial historical data migration to data lake and/or the Spark pool.
  - Plan an approach to migrate historical data.
- **Output C:** We will have tested and determined the data ingestion rate achievable in our environment and can determine whether our data ingestion rate is sufficient to migrate historical data during the available time window.
  - **Test C1:** Test different approaches of historical data migration. For more information, see Transfer data to and from Azure.
  - **Test C2:** Identify allocated bandwidth of ExpressRoute and if there is any throttling setup by the infra team. For more information, see What is Azure ExpressRoute? (Bandwidth options).
  - **Test C3:** Test data transfer rate for both online and offline data migration. For more information, see Copy activity performance and scalability guide.
  - **Test C4:** Test data transfer from the data lake to the SQL pool by using either ADF, Polybase, or the COPY command. For more information, see Data loading strategies for dedicated SQL pool in Azure Synapse Analytics.
- **Goal D:** We will have tested the data ingestion rate of incremental data loading and will have the data points to estimate the data ingestion and processing time window to the data lake and/or the dedicated SQL pool.
- **Output D:** We will have tested the data ingestion rate and can determine whether our data ingestion and processing requirements can be met with the identified approach.
  - **Test D1:** Test the daily update data ingestion and processing.
  - **Test D1:** Test the processed data load to the dedicated SQL pool table from the Spark pool. For more information, see Azure Synapse Dedicated SQL Pool Connector for Apache Spark.
  - **Test D1:** Execute the daily update load process concurrently while running end user queries.

Be sure to refine your tests by adding multiple testing scenarios. Azure Synapse makes it easy to test different scale (varying number of worker nodes, size of the worker nodes like small, medium, and large) to compare performance and behavior.

Here are some testing scenarios:

- **Spark pool test A:** We will execute data processing across multiple node types (small, medium, and large) as well as different numbers of worker nodes.
- **Spark pool test B:** We will load/retrieve processed data from the Spark pool to the dedicated SQL pool by using the connector.
- **Spark pool test C:** We will load/retrieve processed data from the Spark pool to Cosmos DB by using Azure Synapse Link.

Evaluate the POC dataset

Using the specific tests you identified, select a dataset to support the tests. Take time to review this dataset. You should verify that the dataset will adequately represent your future processing in terms of content, complexity, and scale. Don't use a dataset that's too small (less than 1TB) because it won't deliver representative

performance. Conversely, don't use a dataset that's too large because the POC shouldn't become a full data migration. Be sure to obtain the appropriate benchmarks from existing systems so you can use them for performance comparisons.

> **IMPORTANT**
> Make sure you check with business owners for any blockers before moving any data to the cloud. Identify any security or privacy concerns or any data obfuscation needs that should be done before moving data to the cloud.

Create a high-level architecture

Based upon the high-level architecture of your proposed future state architecture, identify the components that will form part of your POC. Your high-level future state architecture likely contains many data sources, numerous data consumers, big data components, and possibly machine learning and artificial intelligence (AI) data consumers. Your POC architecture should specifically identify components that will be part of the POC. Importantly, it should identify any components that won't form part of the POC testing.

If you're already using Azure, identify any resources you already have in place (Azure Active Directory, ExpressRoute, and others) that you can use during the POC. Also identify the Azure regions your organization uses. Now is a great time to identify the throughput of your ExpressRoute connection and to check with other business users that your POC can consume some of that throughput without adverse impact on production systems.

For more information, see Big data architectures.

Identify POC resources

Specifically identify the technical resources and time commitments required to support your POC. Your POC will need:

- A business representative to oversee requirements and results.
- An application data expert, to source the data for the POC and provide knowledge of the existing processes and logic.
- An Apache Spark and Spark pool expert.
- An expert advisor, to optimize the POC tests.
- Resources that will be required for specific components of your POC project, but not necessarily required for the duration of the POC. These resources could include network admins, Azure admins, Active Directory admins, Azure portal admins, and others.
- Ensure all the required Azure services resources are provisioned and the required level of access is granted, including access to storage accounts.
- Ensure you have an account that has required data access permissions to retrieve data from all data sources in the POC scope.

> **TIP**
> We recommend engaging an expert advisor to assist with your POC. Microsoft's partner community has global availability of expert consultants who can help you assess, evaluate, or implement Azure Synapse.

Set the timeline

Review your POC planning details and business needs to identify a time frame for your POC. Make realistic estimates of the time that will be required to complete the POC goals. The time to complete your POC will be influenced by the size of your POC dataset, the number and complexity of tests, and the number of interfaces to test. If you estimate that your POC will run longer than four weeks, consider reducing the POC scope to focus on the highest priority goals. Be sure to obtain approval and commitment from all the lead resources and sponsors

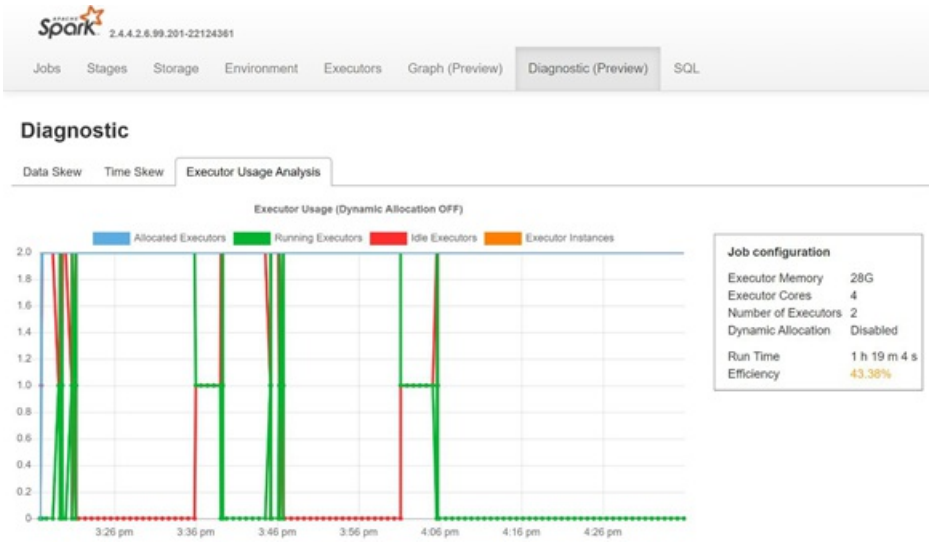
before continuing.

Put the POC into practice

We recommend you execute your POC project with the discipline and rigor of any production project. Run the project according to plan and manage a change request process to prevent uncontrolled growth of the POC's scope.

Here are some examples of high-level tasks:

1. Create a Synapse workspace, Spark pools and dedicated SQL pools, storage accounts, and all Azure resources identified in the POC plan.
2. Load POC dataset:
   - Make data available in Azure by extracting from the source or by creating sample data in Azure. For more information, see:
     - Transferring data to and from Azure
     - Azure Data Box
     - Copy activity performance and scalability guide
     - Data loading strategies for dedicated SQL pool in Azure Synapse Analytics
     - Bulk load data using the COPY statement
   - Test the dedicated connector for the Spark pool and the dedicated SQL pool.
3. Migrate existing code to the Spark pool:
   - If you're migrating from Spark, your migration effort is likely to be straightforward given that the Spark pool leverages the open-source Spark distribution. However, if you're using vendor specific features on top of core Spark features, you'll need to correctly map these features to the Spark pool features.
   - If you're migrating from a non-Spark system, your migration effort will vary based on the complexity involved.
4. Execute the tests:
   - Many tests can be executed in parallel across multiple Spark pool clusters.
   - Record your results in a consumable and readily understandable format.
5. Monitor for troubleshooting and performance. For more information, see:
   - Monitor Apache Spark activities
   - Monitor with web user interfaces - Spark's history server
   - Monitoring resource utilization and query activity in Azure Synapse Analytics
6. Monitor data skewness, time skewness and executor usage percentage by opening the **Diagnostic** tab of Spark's history server.



Interpret the POC results

When you complete all the POC tests, you evaluate the results. Begin by evaluating whether the POC goals were met and the desired outputs were collected. Determine whether more testing is necessary or any questions need addressing.

Next steps

Data lake exploration with dedicated SQL pool in Azure Synapse Analytics

Data lake exploration with serverless SQL pool in Azure Synapse Analytics

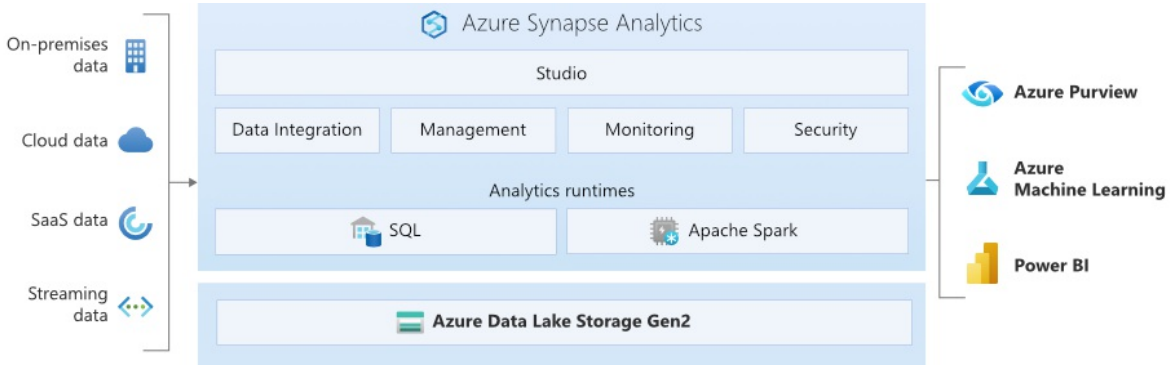
Azure Synapse Analytics frequently asked questions

Azure Synapse implementation success by design

8/19/2022 • 5 minutes to read • Edit Online

The *Azure Synapse implementation success by design* series of articles is designed to help you deliver a successful implementation of Azure Synapse Analytics. It describes a methodology to complement your solution implementation project. It includes suggested checks at strategic points during your project that can help assure a successful implementation. It's important to understand that the methodology shouldn't replace or change your chosen project management methodology (Scrum, Agile, or waterfall). Rather, it suggests validations that can improve the success of your project deployment to a production environment.

Azure Synapse is an enterprise analytics service that accelerates time to insight across data warehouses and big data systems. It brings together the best of SQL technologies used in enterprise data warehousing, Spark technologies used for big data, pipelines for data integration and ETL/ELT, and deep integration with other Azure services, such as Power BI, Azure Cosmos DB, and Azure Machine Learning.



The methodology uses a strategic checkpoint approach to assess and monitor the progress of your project. The goals of these checkpoints are:

- Proactive identification of possible issues and blockers.
- Continuous validation of the solution's fit to the use cases.
- Successful deployment to production.
- Smooth operation and monitoring once in production.

The checkpoints are invoked at four milestones during the project:

1. Project planning
2. Solution development
3. Pre go-live
4. Post go-live

Project planning checkpoint

The project planning checkpoint includes the solution evaluation, project plan evaluation, the solution development environment design evaluation, and the team skill sets evaluation.

Solution evaluation

Evaluate your entire solution with a focus on how it intends to use Azure Synapse. An assessment involves gathering data that will identify the required components of Azure Synapse, the interfaces each will have with other products, a review of the data sources, the data consumers, the roles, and use cases. This assessment will also gather data about the existing environment including detailed specifications from existing data warehouses, big data environments, and integration and data consumption tooling. The assessment will identify which Azure

Synapse components will be implemented and therefore which evaluations and checkpoints should be made throughout the implementation effort. This assessment will also provide additional information to validate the design and implementation against requirements, constraints, and assumptions.

Here's a list of tasks you should complete.

1. Assess your environment to help evaluate the solution design.
2. Make informed technology decisions to implement Azure Synapse and identify the solution components to implement.
3. Evaluate the workspace design.
4. Evaluate the data integration design.
5. Evaluate the dedicated SQL pool design.
6. Evaluate the serverless SQL pool design.
7. Evaluate the Spark pool design.
8. Review the results of each evaluation and respond accordingly.

Project plan evaluation

Evaluate the project plan as it relates to the Azure Synapse requirements that need to be developed. This evaluation isn't about producing a project plan. Rather, the evaluation is about identifying any steps that could lead to blockers or that could impact on the project timeline. Once evaluated, you may need to make adjustments to the project plan.

Here's a list of tasks you should complete.

1. Evaluate the project plan.
2. Evaluate project planning specific to the Azure Synapse components you plan to implement.
3. Review the results of each evaluation and respond accordingly.

Solution development environment design evaluation

Evaluate the environment that's to be used to develop the solution. Establish separate development, test, and production environments. Also, it's important to understand that setting up automated deployment and source code control is essential to a successful and smooth development effort.

Here's a list of tasks you should complete.

1. Evaluate the solution development environment design.
2. Review the results of each evaluation and respond accordingly.

Team skill sets evaluation

Evaluate the project team with a focus on their skill level and readiness to implement the Azure Synapse solution. The success of the project depends on having the correct skill sets and experience. Many different skill sets are required to implement an Azure Synapse solution, so ensure you identify gaps and secure suitable resources that have the required skill sets (or arrange for them to complete training). This evaluation is critical at this stage of your project because a lack of the proper skills can impact on both the timeline and the overall success of the project.

Here's a list of tasks you should complete.

1. Evaluate the team skill sets.
2. Secure skilled resources, or upskill resources to expand their capabilities.
3. Review the results of each evaluation and respond accordingly.

Solution development checkpoint

The solution development checkpoint includes periodic quality checks and additional skill building.

Periodic quality checks

During solution development, you should make periodic checks to validate that the solution is being developed according to recommended practices. Check that the project use cases will be satisfied and that enterprise requirements are being met. For the purposes of this methodology, these checks are called *periodic quality checks*.

Implement the following quality checks:

- Quality checks for workspaces.
- Quality checks for data integration.
- Quality checks for dedicated SQL pools.
- Quality checks for serverless SQL pools.
- Quality checks for Spark pools.

Additional skill building

As the project progresses, identify whether more skill sets are needed. Take the time to determine whether more skill sets could improve the quality of the solution. Supplementing the team with more skill sets can help to avoid project delays and project timeline impacts.

Pre go-live checkpoint

Before deploying your solution to production, we recommend that you perform reviews to assess the preparedness of the solution.

The *pre go-live* checklist provides a final readiness check to successfully deploy to production.

1. Perform the operational readiness review.
2. Perform the user readiness and onboarding plan review.
3. Review the results of each review and respond accordingly.

Post go-live checkpoint

After deploying to production, we recommend that you validate that the solution operates as expected.

The *post go-live* checklist provides a final readiness check to monitor your Azure Synapse solution.

1. Perform the monitoring review.
2. Continually monitor your Azure Synapse solution.

Next steps

In the next article in the *Azure Synapse implementation success by design* series, learn how to assess your environment to help evaluate the solution design and make informed technology decisions to implement Azure Synapse.

Synapse implementation success methodology: Assess environment

8/19/2022 • 16 minutes to read • Edit Online

> **NOTE**
> This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

The first step when implementing Azure Synapse Analytics is to assessment your environment. An assessment provides you with the opportunity to gather all the available information about your existing environment, environmental requirements, project requirements, constraints, timelines, and pain points. This information will form the basis of later evaluations and checkpoint activities. It will prove invaluable when it comes time to validate and compare against the project solution as it's planned, designed, and developed. We recommend that you dedicate a good amount of time to gather all the information and be sure to have necessary discussions with relevant groups. Relevant groups can include project stakeholders, business users, solution designers, and subject matter experts (SMEs) of the existing solution and environment.

The assessment will become a guide to help you evaluate the solution design and make informed technology recommendations to implement Azure Synapse.

Workload assessment

The workload assessment is concerned with the environment, analytical workload roles, ETL/ELT, networking and security, the Azure environment, and data consumption.

Environment

For the environment, evaluate the following points.

- Describe your existing analytical workload:
  - What are the workloads (like data warehouse or big data)?
  - How is this workload helping the business? What are the use case scenarios?
  - What is the business driver for this analytical platform and for potential migration?
  - Gather details about the existing architecture, design, and implementation choices.
  - Gather details about all existing upstream and downstream dependent components and consumers.
- Are you migrating an existing data warehouse (like Microsoft SQL Server, Microsoft Analytics Platform System (APS), Netezza, Snowflake, or Teradata)?
- Are you migrating a big data platform (like Cloudera or Hortonworks)?
- Gather the architecture and dataflow diagrams for the current analytical environment.
- Where are the data sources for your planned analytical workloads located (Azure, other cloud providers, or on-premises)?
- What is the total size of existing datasets (historical and incremental)? What is the current rate of growth of your dataset(s)? What is the projected rate of growth of your datasets for the next 2-5 years?
- Do you have an existing data lake? Gather as much detail as possible about file types (like Parquet or CSV), file sizes, and security configuration.
- Do you have semi-structured or unstructured data to process and analyze?
- Describe the nature of the data processing (batch or real-time processing).

- Do you need interactive data exploration from relational data, data lake, or other sources?
- Do you need real-time data analysis and exploration from operational data sources?
- What are the pain points and limitations in the current environment?
- What source control and DevOps tools are you using today?
- Do you have a use case to build a hybrid (cloud and on-premises) analytical solution, cloud only, or multi-cloud?
- Gather information on the existing cloud environment. Is it a single-cloud provider or a multi-cloud provider?
- Gather plans about the future cloud environment. Will it be a single-cloud provider or a multi-cloud provider?
- What are the RPO/RTO/HA/SLA requirements in the existing environment?
- What are the RPO/RTO/HA/SLA requirements in the planned environment?

Analytical workload roles

For the analytical workload roles, evaluate the following points.

- Describe the different roles (data scientist, data engineer, data analyst, and others).
- Describe the analytical platform access control requirement for these roles.
- Identify the platform owner who's responsible to provision compute resources and grant access.
- Describe how different data roles currently collaborate.
- Are there multiple teams collaborating on the same analytical platform? If so, what's the access control and isolation requirements for each of these teams?
- What are the client tools that end users use to interact with the analytical platform?

ETL/ELT, transformation, and orchestration

For ETL/ELT, transformation, and orchestration, evaluate the following points.

- What tools are you using today for data ingestion (ETL or ELT)?
- Where do these tools exist in the existing environment (on-premises or the cloud)?
- What are your current data load and update requirements (real-time, micro batch, hourly, daily, weekly, or monthly)?
- Describe the transformation requirements for each layer (big data, data lake, data warehouse).
- What is the current programming approach for transforming the data (no-code, low-code, programming like SQL, Python, Scala, C#, or other)?
- What is the preferred planned programming approach to transform the data (no-code, low-code, programming like SQL, Python, Scala, C#, or other)?
- What tools are currently in use for data orchestration to automate the data-driven process?
- Where are the data sources for your existing ETL located (Azure, other cloud provider, or on-premises)?
- What are the existing data consumptions tools (reporting, BI tools, open-source tools) that require integration with the analytical platform?
- What are the planned data consumptions tools (reporting, BI tools, open-source tools) that will require integration with the analytical platform?

Networking and security

For networking and security, evaluate the following points.

- What regulatory requirements do you have for your data?
- If your data contains customer content, payment card industry (PCI), or Health Insurance Portability and Accountability Act of 1996 (HIPAA) data, has your security group certified Azure for this data? If so, for which Azure services?
- Describe your user authorization and authentication requirements.
- Are there security issues that could limit access to data during implementation?

- Is there test data available to use during development and testing?
- Describe the organizational network security requirements on the analytical compute and storage (private network, public network, or firewall restrictions).
- Describe the network security requirements for client tools to access analytical compute and storage (peered network, private endpoint, or other).
- Describe the current network setup between on-premises and Azure (Azure ExpressRoute, site-to-site, or other).

Use the following checklists of possible requirements to guide your assessment.

- Data protection:
  - In-transit encryption
  - Encryption at rest (default keys or customer-managed keys)
  - Data discovery and classification
- Access control:
  - Object-level security
  - Row-level security
  - Column-level security
  - Dynamic data masking
- Authentication:
  - SQL login
  - Azure Active Directory (Azure AD)
  - Multi-factor authentication (MFA)
- Network security:
  - Virtual networks
  - Firewall
  - Azure ExpressRoute
- Threat protection:
  - Threat detection
  - Auditing
  - Vulnerability assessment

For more information, see the Azure Synapse Analytics security white paper.

Azure environment

For the Azure environment, evaluate the following points.

- Are you currently using Azure? Is it used for production workloads?
- If you're using Azure, which services are you using? Which regions are you using?
- Do you use Azure ExpressRoute? What's its bandwidth?
- Do you have budget approval to provision the required Azure services?
- How do you currently provision and manage resources (Azure Resource Manager (ARM) or Terraform)?
- Is your key team familiar with Synapse Analytics? Is any training required?

Data consumption

For data consumption, evaluate the following points.

- Describe how and what tools you currently use to perform activities like ingest, explore, prepare, and data visualization.
- Identify what tools you plan to use to perform activities like ingest, explore, prepare, and data visualization.
- What applications are planned to interact with the analytical platform (Microsoft Power BI, Microsoft Excel,

Microsoft SQL Server Reporting Services, Tableau, or others)?
- Identify all data consumers.
- Identify data export and data sharing requirements.

Azure Synapse services assessment

The Azure Synapse services assessment is concerned with the services within Azure Synapse. Azure Synapse has the following components for compute and data movement:

- **Synapse SQL:** A distributed query system for Transact-SQL (T-SQL) that enables data warehousing and data virtualization scenarios. It also extends T-SQL to address streaming and machine learning (ML) scenarios. Synapse SQL offers both *serverless* and *dedicated* resource models.
- **Serverless SQL pool:** A distributed data processing system, built for large-scale data and computational functions. There's no infrastructure to set up or clusters to maintain. This service is suited for unplanned or burst workloads. Recommended scenarios include quick data exploration on files directly on the data lake, logical data warehouse, and data transformation of raw data.
- **Dedicated SQL pool:** Represents a collection of analytic resources that are provisioned when using Synapse SQL. The size of a dedicated SQL pool (formerly SQL DW) is determined by Data Warehousing Units (DWU). This service is suited for a data warehouse with predictable, high performance continuous workloads over data stored in SQL tables.
- **Apache Spark pool:** Deeply and seamlessly integrates Apache Spark, which is the most popular open source big data engine used for data preparation, data engineering, ETL, and ML.
- **Data integration pipelines:** Azure Synapse contains the same data integration engine and experiences as Azure Data Factory (ADF). They allow you to create rich at-scale ETL pipelines without leaving Azure Synapse.

To help determine the best SQL pool type (dedicated or serverless), evaluate the following points.

- Do you want to build a traditional relational data warehouse by reserving processing power for data stored in SQL tables?
- Do your use cases demand predictable performance?
- Do you want to a build a logical data warehouse on top of a data lake?
- Do you want to query data directly from a data lake?
- Do you want to explore data from a data lake?

The following table compares the two Synapse SQL pool types.

COMPARISON	DEDICATED SQL POOL	SERVERLESS SQL POOL
Value propositions	Fully managed capabilities of a data warehouse. Predictable and high performance for continuous workloads. Optimized for managed (loaded) data.	Easy to get started and explore data lake data. Better total cost of ownership (TCO) for ad hoc and intermittent workloads. Optimized for querying data in a data lake.
Workloads	*Ideal for continuous workloads.* Loading boosts performance, with more complexity. Charging per DWU (when sized well) will be cost-beneficial.	*Ideal for ad hoc or intermittent workloads.* There's no need to load data, so it's easier to start and run. Charging per usage will be cost-beneficial.
Query performance	*Delivers high concurrency and low latency.* Supports rich caching options, including materialized views. There's the ability to choose trade-offs with workload management (WLM).	*Not suited for dashboarding queries.* Millisecond response times aren't expected. It works only on external data.

Dedicated SQL pool assessment

For the dedicated SQL pool assessment, evaluate the following platform points.

- What is the current data warehouse platform (Microsoft SQL Server, Netezza, Teradata, Greenplum, or other)?
- For a migration workload, determine the make and model of your appliance for each environment. Include details of CPUs, GPUs, and memory.
- For an appliance migration, when was the hardware purchased? Has the appliance been fully depreciated? If not, when will depreciation end? And, how much capital expenditure is left?
- Are there any hardware and network architecture diagrams?
- Where are the data sources for your planned data warehouse located (Azure, other cloud provider, or on-premises)?
- What are the data hosting platforms of the data sources for your data warehouse (Microsoft SQL Server, Azure SQL Database, DB2, Oracle, Azure Blob Storage, AWS, Hadoop, or other)?
- Are any of the data sources data warehouses? If so, which ones?
- Identify all ETL, ELT, and data loading scenarios (batch windows, streaming, near real-time). Identify existing service level agreements (SLAs) for each scenario and document the expected SLAs in the new environment.
- What is the current data warehouse size?
- What rate of dataset growth is being targeted for the dedicated SQL pool?
- Describe the environments you're using today (development, test, or production).
- Which tools are currently in place for data movement (ADF, Microsoft SQL Server Integration Services (SSIS), robocopy, Informatica, SFTP, or others)?
- Are you planning to load real-time or near real-time data?

Evaluate the following database points.

- What is the number of objects in each data warehouse (schemas, tables, views, stored procedures, functions)?
- Is it a star schema, snowflake schema or other design?
- What are the largest tables in terms of size and number of records?
- What are the widest tables in terms of the number of columns?
- Is there already a data model designed for your data warehouse? Is it a Kimball, Inmon, or star schema design?
- Are Slowly Changing Dimensions (SCDs) in use? If so, which types?
- Will a semantic layer be implemented by using relational data marts or Analysis Services (tabular or multidimensional), or another product?
- What are the HA/RPO/RTO/data archiving requirements?
- What are the region replication requirements?

Evaluate the following workload characteristics.

- What is the estimated number of concurrent users or jobs accessing the data warehouse during *peak hours*?
- What is the estimated number of concurrent users or jobs accessing the data warehouse during *off peak hours*?
- Is there a period of time when there will be no users or jobs?
- What are your query execution performance expectations for interactive queries?
- What are your data load performance expectations for daily/weekly/monthly data loads or updates?
- What are your query execution expectations for reporting and analytical queries?
- How complex will the most commonly executed queries be?
- What percentage of your total dataset size is your active dataset?
- Approximately what percentage of the workload is anticipated for loading or updating, batch processing or

reporting, interactive query, and analytical processing?

- Identify the data consuming patterns and platforms:
  - Current and planned reporting method and tools.
  - Which application or analytical tools will access the data warehouse?
  - Number of concurrent queries?
  - Average number of active queries at any point in time?
  - What is the nature of data access (interactive, ad hoc, export, or others)?
  - Data roles and complete description of their data requirements.
  - Maximum number of concurrent connections.
- Query performance SLA pattern by:
  - Dashboard users.
  - Batch reporting.
  - ML users.
  - ETL process.
- What are the security requirements for the existing environment and for the new environment (row-level security, column-level security, access control, encryption, and others)?
- Do you have requirements to integrate ML model scoring with T-SQL?

Serverless SQL pool assessment

Synapse Serverless SQL pool supports three major use cases.

- **Basic discovery and exploration:** Quickly reason about the data in various formats (Parquet, CSV, JSON) in your data lake, so you can plan how to extract insights from it.
- **Logical data warehouse:** Provide a relational abstraction on top of raw or disparate data without relocating and transforming data, allowing an always-current view of your data.
- **Data transformation:** Simple, scalable, and performant way to transform data in the lake by using T-SQL, so it can be fed to BI and other tools or loaded into a relational data store (Synapse SQL databases, Azure SQL Database, or others).

Different data roles can benefit from serverless SQL pool:

- **Data engineers** can explore the data lake, transform and prepare data by using this service, and simplify their data transformation pipelines.
- **Data scientists** can quickly reason about the contents and structure of the data in the data lake, thanks to features such as OPENROWSET and automatic schema inference.
- **Data analysts** can explore data and Spark external tables created by data scientists or data engineers by using familiar T-SQL statements or their favorite query tools.
- **BI professionals** can quickly create Power BI reports on top of data in the data lake and Spark tables.

> **NOTE**
>
> The T-SQL language is used in both dedicated SQL pool and the serverless SQL pool, however there are some differences in the set of supported features. For more information about T-SQL features supported in Synapse SQL (dedicated and serverless), see Transact-SQL features supported in Azure Synapse SQL.

For the serverless SQL pool assessment, evaluate the following points.

- Do you have use cases to discover and explore data from a data lake by using relational queries (T-SQL)?
- Do you have use cases to build a logical data warehouse on top of a data lake?
- Identify whether there are use cases to transform data in the data lake without first moving data from the data lake.

- Is your data already in Azure Data Lake Storage (ADLS) or Azure Blob Storage?
- If your data is already in ADLS, do you have a good partition strategy in the data lake?
- Do you have operational data in Azure Cosmos DB? Do you have use cases for real-time analytics on Azure Cosmos DB without impacting transactions?
- Identify the file types in the data lake.
- Identify the query performance SLA. Does your use case demand predictable performance and cost?
- Do you have unplanned or bursty SQL analytical workloads?
- Identify the data consuming pattern and platforms:
  - Current and planned reporting method and tools.
  - Which application or analytical tools will access the serverless SQL pool?
  - Average number of active queries at any point in time.
  - What is the nature of data access (interactive, ad hoc, export, or others)?
  - Data roles and complete description of their data requirements.
  - Maximum number of concurrent connections.
  - Query complexity?
- What are the security requirements (access control, encryption, and others)?
- What is the required T-SQL functionality (stored procedures or functions)?
- Identify the number of queries that will be sent to the serverless SQL pool and the result set size of each query.

> **TIP**
>
> If you're new to serverless SQL pools, we recommend you work through the Build data analytics solutions using Azure Synapse serverless SQL pools learning path.

Spark pool assessment

Spark pools in Azure Synapse enable the following key scenarios.

- **Data engineering/Data preparation:** Apache Spark includes many language features to support preparation and processing of large volumes of data. Preparation and processing can make the data more valuable and allow it to be consumed by other Azure Synapse services. It's enabled through multiple languages (C#, Scala, PySpark, Spark SQL) and by using supplied libraries for processing and connectivity.
- **Machine learning:** Apache Spark comes with MLlib, which is an ML library built on top of Spark that you can use from a Spark pool. Spark pools also include Anaconda, which is a Python distribution that comprises various packages for data science including ML. In addition, Apache Spark on Synapse provides pre-installed libraries for Microsoft Machine Learning, which is a fault-tolerant, elastic, and RESTful ML framework. When combined with built-in support for notebooks, you have a rich environment for creating ML applications.

> **NOTE**
>
> For more information, see Apache Spark in Azure Synapse Analytics.
>
> Also, Azure Synapse is compatible with Linux Foundation Delta Lake. Delta Lake is an open-source storage layer that brings ACID (atomicity, consistency, isolation, and durability) transactions to Apache Spark and big data workloads. For more information, see What is Delta Lake.

For the Spark pool assessment, evaluate the following points.

- Identify the workloads that require data engineering or data preparation.
- Clearly define the types of transformations.
- Identify whether you have unstructured data to process.

- When you're migrating from an existing Spark/Hadoop workload:
  - What is the existing big data platform (Cloudera, Hortonworks, cloud services, or other)?
  - If it's a migration from on-premises, has hardware depreciated or licenses expired? If not, when will depreciation or expiry happen?
  - What is the existing cluster type?
  - What are the required libraries and Spark versions?
  - Is it a Hadoop migration to Spark?
  - What are the current or preferred programming languages?
  - What is the type of workload (big data, ML, or other)?
  - What are the existing and planned client tools and reporting platforms?
  - What are the security requirements?
  - Are there any current pain points and limitations?
- Do you plan to use, or are currently using, Delta Lake?
- How do you manage packages today?
- Identify the required compute cluster types.
- Identify whether cluster customization is required.

> **TIP**
> If you're new to Spark pools, we recommend you work through the Perform data engineering with Azure Synapse Apache Spark Pools learning path.

Next steps

In the next article in the *Azure Synapse success by design* series, learn how to evaluate the Synapse workspace design and validate that it meets guidelines and requirements.

Synapse implementation success methodology: Evaluate workspace design

8/19/2022 • 6 minutes to read • Edit Online

NOTE

This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

Synapse workspace is a unified graphical user experience that stitches together your analytical and data processing engines, data lakes, databases, tables, datasets, and reporting artifacts along with code and process orchestration. Considering the number of technologies and services that are integrated into Synapse workspace, ensure that the key components are included in your design.

Synapse workspace design review

Identify whether your solution design involves one Synapse workspace or multiple workspaces. Determine the drivers of this design. While there might be different reasons, in most cases the reason for multiple workspaces is either security segregation or billing segregation. When determining the number of workspaces and database boundaries, keep in mind that there's a limit of 20 workspaces per subscription.

Identify which elements or services within each workspace need to be shared and with which resources. Resources can include data lakes, integration runtimes (IRs), metadata or configurations, and code. Determine why this particular design was chosen in terms of potential synergies. Ask yourself whether these synergies justify the extra cost and management overhead.

Data lake design review

We recommended that the data lake (if part of your solution) be properly tiered. You should divide your data lake into three major areas that relate to *Bronze*, *Silver*, and *Gold* datasets. Bronze - or the raw layer - might reside on its own separate storage account because it has stricter access controls due to unmasked sensitive data that it might store.

Security design review

Review the security design for the workspace and compare it with the information you gathered during the assessment. Ensure all of the requirements are met, and all of the constraints have been taken into account. For ease of management, we recommended that users be organized into groups with appropriate permissions profiling: you can simplify access control by using security groups that align with roles. That way, network administrators can add or remove users from appropriate security groups to manage access.

Serverless SQL pools and Apache Spark tables store their data in an Azure Data Lake Gen2 (ADLS Gen2) container that's associated with the workspace. User-installed Apache Spark libraries are also managed in this same storage account. To enable these use cases, both users and the workspace managed service identity (MSI) must be added to the **Storage Blob Data Contributor** role of the ADLS Gen2 storage container. Verify this requirement against your security requirements.

Dedicated SQL pools provide a rich set of security features to encrypt and mask sensitive data. Both dedicated and serverless SQL pools enable the full surface area of SQL Server permissions including built-in roles, user-

defined roles, SQL authentication, and Azure Active Directory (Azure AD) authentication. Review the security design for your solution's dedicated SQL pool and serverless SQL pool access and data.

Review the security plan for your data lake and all the ADLS Gen2 storage accounts (and others) that will form part of your Azure Synapse Analytics solution. ADLS Gen2 storage isn't itself a compute engine and so it doesn't have a built-in ability to selectively mask data attributes. You can apply ADLS Gen2 permissions at the storage account or container level by using role-based access control (RBAC) and/or at the folder or file level by using access control lists (ACLs). Review the design carefully and strive to avoid unnecessary complexity.

Here are some points to consider for the security design.

- Make sure Azure AD set up requirements are included in the design.
- Check for cross-tenant scenarios. Such issues may arise because some data is in another Azure tenant, or it needs to move to another tenant, or it needs to be accessed by users from another tenant. Ensure these scenarios are considered in your design.
- What are the roles for each workspace? How will they use the workspace?
- How is the security designed within the workspace?
  - Who can view all scripts, notebooks, and pipelines?
  - Who can execute scripts and pipelines?
  - Who can create/pause/resume SQL and Spark pools?
  - Who can publish changes to the workspace?
  - Who can commit changes to source control?
- Will pipelines access data by using stored credentials or the workspace managed identity?
- Do users have the appropriate access to the data lake to browse the data in Synapse Studio?
- Is the data lake properly secured by using the appropriate combination of RBAC and ACLs?
- Have the SQL pool user permissions been correctly set for each role (data scientist, developer, administrator, business user, and others)?

Networking design review

Here are some points to consider for the network design.

- Is connectivity designed between all the resources?
- What is the networking mechanism to be used (Azure ExpressRoute, public Internet, or private endpoints)?
- Do you need to be able to securely connect to Synapse Studio?
- Has data exfiltration been taken into consideration?
- Do you need to connect to on-premises data sources?
- Do you need to connect to other cloud data sources or compute engines, such as Azure Machine Learning?
- Have Azure networking components, like network security groups (NSGs), been reviewed for proper connectivity and data movement?
- Has integration with the private DNS zones been taken into consideration?
- Do you need to be able to browse the data lake from within Synapse Studio or simply query data in the data lake with serverless SQL or PolyBase?

Finally, identify all of your data consumers and verify that their connectivity is accounted for in the design. Check that network and security outposts allow your service to access required on-premises sources and that its authentication protocols and mechanisms are supported. In some scenarios, you might need to have more than one self-hosted IR or data gateway for SaaS solutions, like Microsoft Power BI.

Monitoring design review

Review the design of the monitoring of the Azure Synapse components to ensure they meet the requirements

and expectations identified during the assessment. Verify that monitoring of resources and data access has been designed, and that it identifies each monitoring requirement. A robust monitoring solution should be put in place as part of the first deployment to production. That way, failures can be identified, diagnosed, and addressed in a timely manner. Aside from the base infrastructure and pipeline runs, data should also be monitored. Depending on the Azure Synapse components in use, identify the monitoring requirements for each component. For example, if Spark pools form part of the solution, monitor the malformed record store.

Here are some points to consider for the monitoring design.

- Who can monitor each resource type (pipelines, pools, and others)?
- How long do database activity logs need to be retained?
- Will workspace and database log retention use Log Analytics or Azure Storage?
- Will alerts be triggered in the event of a pipeline error? If so, who should be notified?
- What threshold level of a SQL pool should trigger an alert? Who should be notified?

Source control design review

By default, a Synapse workspace applies changes directly to the Synapse service by using the built-in publish functionality. You can enable source control integration, which provides many advantages. Advantages include better collaboration, versioning, approvals, and release pipelines to promote changes through to development, test, and production environments. Azure Synapse allows a single source control repository per workspace, which can be either Azure DevOps Git or GitHub.

Here are some points to consider for the source control design.

- If using Azure DevOps Git, is the Synapse workspace and its repository in the same tenant?
- Who will be able to access source control?
- What permissions will each user be granted in source control?
- Has a branching and merging strategy been developed?
- Will release pipelines be developed for deployment to different environments?
- Will an approval process be used for merging and for release pipelines?

> **NOTE**
> The design of the development environment is of critical importance to the success of your project. If a development environment has been designed, it will be evaluated in a separate stage of this methodology.

Next steps

In the next article in the *Azure Synapse success by design* series, learn how to evaluate the data integration design and validate that it meets guidelines and requirements.

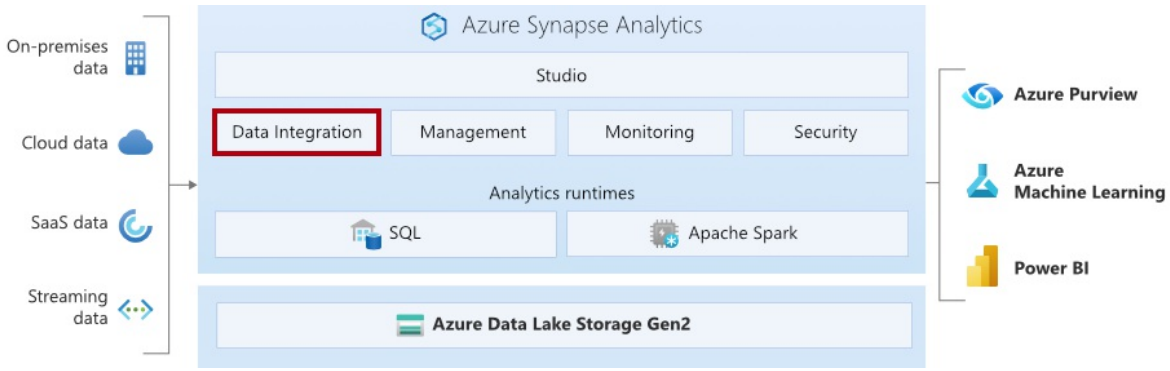
Synapse implementation success methodology: Evaluate data integration design

8/19/2022 • 5 minutes to read • Edit Online

NOTE

This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

Azure Synapse Analytics contains the same data integration engine and experiences as Azure Data Factory (ADF), allowing you to create rich at-scale ETL pipelines without leaving Azure Synapse Analytics.



This article describes how to evaluate the design of the data integration components for your project. Specifically, it helps you to determine whether Azure Synapse pipelines are the best fit for your data integration requirements. Time invested in evaluating the design prior to solution development can help to eliminate unexpected design changes that may impact on your project timeline or cost.

Fit gap analysis

You should perform a thorough fit gap analysis of your data integration strategy. If you choose Azure Synapse pipelines as the data integration tool, review the following points to ensure they're the best fit for your data integration requirements and orchestration. Even if you choose different data integration tools, you should still review the following points to validate that all key design points have been considered and that your chosen tool will support your solution needs. This information should have been captured during your assessment performed earlier in this methodology.

- Review your data sources and destinations (targets):
  - Validate that source and destination stores are supported data stores.
  - If they're not supported, check whether you can use the extensible options.
- Review the triggering points of your data integration and the frequency:
  - Azure Synapse pipelines support schedule, tumbling window, and storage event triggers.
  - Validate the minimum recurrence interval and supported storage events against your requirements.
- Review the required modes of data integration:
  - Scheduled, periodic, and triggered batch processing can be effectively designed in Azure Synapse pipelines.
  - To implement Change Data Capture (CDC) functionality, use third-party products or create a custom solution.

    - To support real-time streaming, use Azure Event Hubs, Azure Event Hubs from Apache Kafka, or Azure IoT Hub.
    - To run Microsoft SQL Server Integration Services (SSIS) packages, you can lift and shift SSIS workloads to the cloud.
- Review the compute design:
    - Does the compute required for the pipelines need to be serverless or provisioned?
    - Azure Synapse pipelines support both modes of integration runtime (IR): serverless or self-hosted on a Windows machine.
    - Validate ports and firewalls and proxy setting when using the self-hosted IR (provisioned).
- Review security requirements, networking and firewall configuration of the environment and compare them to the security, networking and firewall configuration design:
    - Review how the data sources are secured and networked.
    - Review how the target data stores are secured and networked. Azure Synapse pipelines have different data access strategies that provide a secure way to connect data stores via private endpoints or virtual networks.
    - Use Azure Key Vault to store credentials whenever applicable.
    - Use ADF for customer-managed key (CMK) encryption of credentials and store them in the self-hosted IR.
- Review the design for ongoing monitoring of all data integration components.

Architecture considerations

As you review the data integration design, consider the following recommendations and guidelines to ensure that the data integration components of your solution will provide ongoing operational excellence, performance efficiency, reliability, and security.

Operational excellence

For operational excellence, evaluate the following points.

- **Environment:** When planning your environments, segregate them by development/test, user acceptance testing (UAT), and production. Use the folder organizational options to organize your pipelines and datasets by business/ETL jobs to support better maintainability. Use annotations to tag your pipelines so you can easily monitor them. Create reusable pipelines by using parameters, and iteration and conditional activities.
- **Monitoring and alerting:** Synapse workspaces include the Monitor Hub, which has rich monitoring information of each and every pipeline run. It also integrates with Log Analytics for further log analysis and alerting. You should implement these features to provide proactive error notifications. Also, use *Upon Failure* paths to implement customized error handling.
- **Automated deployment and testing:** Azure Synapse pipelines are built into Synapse workspace, so you can take advantage of workspace automation and deployment. Use ARM templates to minimize manual activities when creating Synapse workspaces. Also, integrate Synapse workspaces with Azure DevOps to build code versioning and automate publication.

Performance efficiency

For performance efficiency, evaluate the following points.

- Follow performance guidance and optimization features when working with the copy activity.
- Choose optimized connectors for data transfer instead of generic connectors. For example, use PolyBase instead of bulk insert when moving data from Azure Data Lake Storage Gen2 (ALDS Gen2) to a dedicated SQL pool.
- When creating a new Azure IR, set the region location as auto-resolve or select the same region as the data stores.
- For self-hosted IR, choose the Azure virtual machine (VM) size based on the integration requirements.

- Choose a stable network connection, like Azure ExpressRoute, for fast and consistent bandwidth.

Reliability

When you execute a pipeline by using Azure IR, it's serverless in nature and so it provides resiliency out of the box. There's little for customers to manage. However, when a pipeline runs in a self-hosted IR, we recommend that you run it by using a high availability configuration in Azure VMs. This configuration ensures integration pipelines aren't broken even when a VM goes offline. Also, we recommend that you use Azure ExpressRoute for a fast and reliable network connection between on-premises and Azure.

Security

A secured data platform is one of the key requirements of every organization. You should thoroughly plan security for the entire platform rather than individual components. Here are some security guidelines for Azure Synapse pipeline solutions.

- Secure data movement to the cloud by using Azure Synapse private endpoints.
- Use Azure Active Directory (Azure AD) managed identities for authentication.
- Use Azure role-based access control (RBAC) and Synapse RBAC for authorization.
- Store credentials, secrets, and keys in Azure Key Vault rather than in the pipeline. For more information, see Use Azure Key Vault secrets in pipeline activities.
- Connect to on-premises resources via Azure ExpressRoute or VPN over private endpoints.
- Enable the **Secure output** and **Secure input** options in pipeline activities when parameters store secrets or passwords.

Next steps

In the next article in the *Azure Synapse success by design* series, learn how to evaluate your dedicated SQL pool design to identify issues and validate that it meets guidelines and requirements.

Synapse implementation success methodology: Evaluate dedicated SQL pool design

8/19/2022 • 2 minutes to read • Edit Online

> **NOTE**
> This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

You should evaluate your dedicated SQL pool design to identify issues and validate that it meets guidelines and requirements. By evaluating the design *before solution development begins*, you can avoid blockers and unexpected design changes. That way, you protect the project's timeline and budget.

Synapse SQL has a scale-out architecture that distributes computational data processing across multiple nodes. Compute is separate from storage, which enables you to scale compute independently of the data in your system. For more information, see Dedicated SQL pool (formerly SQL DW) architecture in Azure Synapse Analytics.

Assessment analysis

During the assessment stage, you collected information about how the original system was deployed and details of the structures that were implemented. That information can now help you to identify gaps between what's implemented and what needs to be developed. For example, now's the time to consider the impact of designing round-robin tables instead of hash distributed tables, or the performance benefits of correctly using replicated tables.

Review the target architecture

To successfully deploy a dedicated SQL pool, it's important to adopt an architecture that's aligned with business requirements. For more information, see Data warehousing in Microsoft Azure.

Migration path

A migration project for Azure Synapse is similar to any other database migration. You should consider that there might be differences between the original system and Azure Synapse.

Ensure that you have a clear migration path established for:

- Database objects, scripts, and queries
- Data transfer (export from source and transit to the cloud)
- Initial data load into Azure Synapse
- Logins and users
- Data access control (row-level security)

For more information, see Migrate a data warehouse to a dedicated SQL pool in Azure Synapse Analytics.

Feature gaps

Determine whether the original system depends on features that aren't supported by Azure Synapse.

Unsupported features in dedicated SQL pools include certain data types, like XML and spatial data types, and cursors.

For more information, see:

- Table data types for dedicated SQL pool (formerly SQL DW) in Azure Synapse Analytics
- Transact-SQL features supported in Azure Synapse SQL

Dedicated SQL pool testing

As with any other project, you should conduct tests to ensure that your dedicated SQL pool delivers the required business needs. It's critical to test data quality, data integration, security, and performance.

Next steps

In the next article in the *Azure Synapse success by design* series, learn how to evaluate your Spark pool design to identify issues and validate that it meets guidelines and requirements.

Synapse implementation success methodology: Evaluate serverless SQL pool design

8/19/2022 • 6 minutes to read • Edit Online

NOTE

This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

You should evaluate your serverless SQL pool design to identify issues and validate that it meets guidelines and requirements. By evaluating the design *before solution development begins*, you can avoid blockers and unexpected design changes. That way, you protect the project's timeline and budget.

The architectural separation of storage and compute for modern data, analytical platforms and services has been a trend and frequently used pattern. It provides cost savings and more flexibility allowing independent on-demand scaling of your storage and compute. Synapse SQL serverless extends this pattern by adding the capability to query your data lake data directly. There's no need to worry about compute management when using self-service types of workloads.

Fit gap analysis

When planning to implement SQL serverless pools within Azure Synapse, you first need to ensure serverless pools are the right fit for your workloads. You should consider operational excellence, performance efficiency, reliability, and security.

Operational excellence

For operational excellence, evaluate the following points.

- **Solution development environment:** Within this methodology, there's an evaluation of the solution development environment. Identify how the environments (development, test, and production) are designed to support solution development. Commonly, you'll find a production and non-production environments (for development and test). You should find Synapse workspaces in all of the environments. In most cases, you'll be obliged to segregate your production and development/test users and workloads.
- **Synapse workspace design:** Within this methodology, there's an evaluation of the Synapse workspace design. Identify how the workspaces have been designed for your solution. Become familiar with the design and know whether the solution will use a single workspace or whether multiple workspaces form part of the solution. Know why a single or multiple workspace design was chosen. A multi-workspace design is often chosen to enforce strict security boundaries.
- **Deployment:** SQL serverless is available on-demand with every Synapse workspace, so it doesn't require any special deployment actions. Check regional proximity of the service and that of the Azure Data Lake Storage Gen2 (ADLS Gen2) account that it's connected to.
- **Monitoring:** Check whether built-in monitoring is sufficient and whether any external services need to be put in place to store historical log data. Log data allows analyzing changes in performance and allows you to define alerting or triggered actions for specific circumstances.

Performance efficiency

Unlike traditional database engines, SQL serverless doesn't rely on its own optimized storage layer. For that reason, its performance is heavily dependent on how data is organized in ADLS Gen2. For performance

efficiency, evaluate the following points.

- **Data ingestion:** Review how data is stored in the data lake. File sizes, the number of files, and folder structure all have an impact on performance. Keep in mind that while some file sizes might work for SQL serverless, they may impose issues for efficient processing or consumption by other engines or applications. You'll need to evaluate the data storage design and validate it against all of the data consumers, including SQL serverless and any other data tools that form part of your solution.
- **Data placement:** Evaluate whether your design has unified and defined common patterns for data placement. Ensure that directory branching can support your security requirements. There are a few common patterns that can help you keep your time series data organized. Whatever your choice, ensure that it also works with other engines and workloads. Also, validate whether it can help partition auto-discovery for Spark applications and external tables.
- **Data formats:** In most cases, SQL serverless will offer the best performance and better compatibility feature-wise by using a Parquet format. Verify your performance and compatibility requirements, because while Parquet improves performance - thanks to better compression and reduction of IO (by reading only required columns needed for analysis) - it requires more compute resources. Also, because some source systems don't natively support Parquet as an export format, it could lead to more transformation steps in your pipelines and/or dependencies in your overall architecture.
- **Exploration:** Every industry is different. In many cases, however, there are common data access patterns found in the most frequent-run queries. Patterns typically involve filtering, and aggregations by dates, categories, or geographic regions. Identify your most common filtering criteria, and relate them to how much data is read/discarded by the most frequent-run queries. Validate whether the information on the data lake is organized to favor your exploration requirements and expectations. For the queries identified in your design and in your assessment, see whether you can eliminate unnecessary partitions in your OPENROWSET path parameter, or - if there are external tables - whether creating more indexes can help.

Reliability

For reliability, evaluate the following points.

- **Availability:** Validate any availability requirements that were identified during the assessment stage. While there aren't any specific SLAs for SQL serverless, there's a 30-minute timeout for query execution. Identify the longest running queries from your assessment and validate them against your serverless SQL design. A 30-minute timeout could break the expectations for your workload and appear as a service problem.
- **Consistency:** SQL serverless is designed primarily for read workloads. So, validate whether all consistency checks have been performed during the data lake data provisioning and formation process. Keep abreast of new capabilities, like Delta Lake open-source storage layer, which provides support for ACID (atomicity, consistency, isolation, and durability) guarantees for transactions. This capability allows you to implement effective lambda or kappa architectures to support both streaming and batch use cases. Be sure to evaluate your design for opportunities to apply new capabilities but not at the expense of your project's timeline or cost.
- **Backup:** Review any disaster recovery requirements that were identified during the assessment. Validate them against your SQL serverless design for recovery. SQL serverless itself doesn't have its own storage layer and that would require handling snapshots and backup copies of your data. The data store accessed by serverless SQL is external (ADLS Gen2). Review the recovery design in your project for these datasets.

Security

Organization of your data is important for building flexible security foundations. In most cases, different processes and users will require different permissions and access to specific sub areas of your data lake or logical data warehouse.

For security, evaluate the following points.

- **Data storage:** Using the information gathered during the assessment stage, identify whether typical *Raw*, *Stage*, and *Curated* data lake areas need to be placed on the same storage account instead of independent

storage accounts. The latter might result in more flexibility in terms of roles and permissions. It can also add more input/output operations per second (IOPS) capacity that might be needed if your architecture must support heavy and simultaneous read/write workloads (like real-time or IoT scenarios). Validate whether you need to segregate further by keeping your sandboxed and master data areas on separate storage accounts. Most users won't need to update or delete data, so they don't need write permissions to the data lake, except for sandboxed and private areas.

- From your assessment information, identify whether any requirements rely on security features like Always Encrypted, Dynamic data masking or Row-level security. Validate the availability of these features in specific scenarios, like when used with the OPENROWSET function. Anticipate potential workarounds that may be required.
- From your assessment information, identify what would be the best authentication methods. Consider Azure Active Directory (Azure AD) service principals, shared access signature (SAS), and when and how authentication pass-through can be used and integrated in the exploration tool of choice of the customer. Evaluate the design and validate that the best authentication method as part of the design.

Other considerations

Review your design and check whether you have put in place best practices and recommendations. Give special attention to filter optimization and collation to ensure that predicate pushdown works properly.

Next steps

In the next article in the *Azure Synapse success by design* series, learn how to evaluate your Spark pool design to identify issues and validate that it meets guidelines and requirements.

Synapse implementation success methodology: Evaluate Spark pool design

8/19/2022 • 4 minutes to read • Edit Online

NOTE

This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

You should evaluate your Apache Spark pool design to identify issues and validate that it meets guidelines and requirements. By evaluating the design *before solution development begins*, you can avoid blockers and unexpected design changes. That way, you protect the project's timeline and budget.

Apache Spark in Synapse brings the Apache Spark parallel data processing to Azure Synapse Analytics. This evaluation provides guidance on when Apache Spark in Azure Synapse is - or isn't - the right fit for your workload. It describes points to consider when you're evaluating your solution design elements that incorporate Spark pools.

Fit gap analysis

When planning to implement Spark pools with Azure Synapse, first ensure they're the best fit for your workload.

Consider the following points.

- Does your workload require data engineering/data preparation?
  - Apache Spark works best for workloads that require:
    - Data cleaning.
    - Transforming semi-structured data, like XML into relational.
    - Complex free-text transformation, like fuzzy matching or natural language processing (NLP).
    - Data preparation for machine learning (ML).
- Does your workload for data engineering/data preparation involve complex or simple transformations? And, are you looking for a low-code/no-code approach?
  - For simple transformations, like removing columns, changing column data types, or joining datasets, consider creating an Azure Synapse pipeline by using a data flow activity.
  - Data flow activities provide a low-code/no-code approach to prepare your data.
- Does your workload require ML on big data?
  - Apache Spark works well for large datasets that will be used for ML. If you're using small datasets, consider using Azure Machine Learning as the compute service.
- Do you plan to perform data exploration or ad hoc query analysis on big data?
  - Apache Spark in Azure Synapse provides Python/Scala/SQL/.NET-based data exploration. However, if you need a full Transact-SQL (T-SQL) experience, consider using a serverless SQL pool.
- Do you have a current Spark/Hadoop workload and do you need a unified big data platform?
  - Azure Synapse provides a unified analytical platform for working with big data. There are Spark and SQL serverless pools for ad hoc queries, and the dedicated SQL pool for reporting and serving data.
  - Moving from a Spark/Hadoop workload from on-premises (or another cloud environment) may involve some refactoring that you should take into consideration.
  - If you're looking for a lift-and-shift approach of your Apache big data environment from on-premises

to the cloud, and you need to meet a strict data engineering service level agreement (SLA), consider using Azure HDInsight.

Architecture considerations

To ensure that your Apache Spark pool meets your requirements for operational excellence, performance, reliability, and security, there are key areas to validate in your architecture.

Operational excellence

For operational excellence, evaluate the following points.

- **Environment:** When configuring your environment, design your Spark pool to take advantage of features such as autoscale and dynamic allocation. Also, to reduce costs, consider enabling the automatic pause feature.
- **Package management:** Determine whether required Apache Spark libraries will be used at a workspace, pool, or session level. For more information, see Manage libraries for Apache Spark in Azure Synapse Analytics.
- **Monitoring:** Apache Spark in Azure Synapse provides built-in monitoring of Spark pools and applications with the creation of each spark session. Also consider implementing application monitoring with Azure Log Analytics or Prometheus and Grafana, which you can use to visualize metrics and logs.

Performance efficiency

For performance efficiency, evaluate the following points.

- **File size and file type:** File size and the number of files have an impact on performance. Design the architecture to ensure that the file types are conducive to native ingestion with Apache Spark. Also, lean toward fewer large files instead of many small files.
- **Partitioning:** Identify whether partitioning at the folder and/or file level will be implemented for your workload. *Folder partitions* limit the amount of data to search and read. *File partitions* reduce the amount of data to be searched inside the file - but only apply to specific file formats that should be considered in the initial architecture.

Reliability

For reliability, evaluate the following points.

- **Availability:** Spark pools have a start time of three to four minutes. It could take longer if there are many libraries to install. When designing batch vs. streaming workloads, identify the SLA for executing the job from your assessment information and determine which architecture best meets your needs. Also, take into consideration that each job execution creates a new Spark pool cluster.
- **Checkpointing:** Apache Spark streaming has a built-in checkpointing mechanism. Checkpointing allows your stream to recover from the last processed entry should there be a failure on a node in your pool.

Security

For security, evaluate the following points.

- **Data access:** Data access must be considered for the Azure Data Lake Storage (ADLS) account that's attached to the Synapse workspace. In addition, determine the security levels required to access any data that isn't within the Azure Synapse environment. Refer to the information you collected during the assessment stage.
- **Networking:** Review the networking information and requirements gathered during your assessment. If the design involves a managed virtual network with Azure Synapse, consider the implications this requirement will have on Apache Spark in Azure Synapse. One implication is the inability to use Spark SQL while accessing data.

Next steps

In the next article in the *Azure Synapse success by design* series, learn how to evaluate your modern data warehouse project plan before the project starts.

For more information on best practices, see Apache Spark for Azure Synapse Guidance.

Synapse implementation success methodology: Evaluate project plan

8/19/2022 • 2 minutes to read • Edit Online

> **NOTE**
> This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

In the lifecycle of the project, the most important and extensive planning is done *before implementation*. This article describes how to conduct a high-level review of your project plan. The aim is to ensure it contains critical artifacts and information to deliver a successful solution. It includes checklists of items that you should complete and approve before the project starts.

A detailed review should follow the high-level project plan review. The detailed review should focus on the specific Azure Synapse components identified during the assessment stage.

Evaluate the project plan

Work through the following two high-level checklists, taking care to verify that each task aligns with the information gathered during the assessment stage.

First, ensure that your project plan defines the following points.

- **The core resource team:** Assemble a group of key people that have expertise crucial to the project.
- **Scope:** Document how the project scope will be defined, verified, measured, and how the work breakdown will be defined and assigned.
- **Schedule:** Define the time duration required to complete the project.
- **Cost:** Estimate costs for internal and external resources, including infrastructure, hardware, and software.

Second, having defined and assigned the work breakdown, prepare the following artifacts.

- **Migration plan:** Document the plan to migrate from your current system to Azure Synapse. Incorporate tasks for executing the migration within the project plan scope and schedule.
- **Success criteria:** Define the critical success criteria for stakeholders (or the project sponsor), including go and no-go criteria.
- **Quality assurance:** Define how to conduct code reviews, and the development, staging, and production promotion approval processes.
- **Test plan:** Define test cases, success criteria for unit, integration, user testing, and metrics to validate all deliverables. Incorporate tasks for developing and executing the test plans within the project plan scope and schedule.

Evaluate project plan detailed tasks

Once the high-level project plan review is complete and approved, the next step is to drill down into each component of the project plan.

Identify the project plan components that address each aspect of Azure Synapse as it's intended for use in your solution. Also, validate that the project plan accounts for all the effort and resources required to develop, test,

deploy, and operate your solution by evaluating:

- The workspace project plan.
- The data integration project plan.
- The dedicated SQL pool project plan.
- The serverless SQL pool project plan.
- The Spark pool project plan.

Next steps

In the next article in the *Azure Synapse success by design* series, learn how to evaluate the environments for your modern data warehouse project to support development, testing, and production.

Synapse implementation success methodology: Evaluate solution development environment design

8/19/2022 • 2 minutes to read • Edit Online

> **NOTE**
> This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

Solution development and the environment within which it's performed is key to the success of your project. Regardless of your selected project methodology (like waterfall, Agile, or Scrum), you should set up multiple environments to support development, testing, and production. You should also define clear processes for promoting changes between environments.

Setting up a modern data warehouse environment for both production and pre-production use can be complex. Keep in mind that one of the key design decisions is automation. Automation helps increase productivity while minimizing the risk of errors. Further, your environments should support future agile development, including the addition of new workloads, like data science or real-time. During the design review, produce a solution development environment design that will support your solution not only for the current project but also for ongoing support and development of your solution.

Solution development environment design

The environment design should include the production environment, which hosts the production solution, and at least one non-production environment. Most environments contain two non-production environments: one for development and another for testing, Quality Assurance (QA), and User Acceptance Testing (UAT). Typically, environments are hosted in separate Azure subscriptions. Consider creating a production subscription, and a non-production subscription. This separation will provide a clear security boundary and delineation between production and non-production.

Ideally, you should establish three environments.

- **Development:** The environment within which your data and analytics solutions are built. Determine whether to provide sandboxes for developers. Sandboxes can allow developers to make and test their changes in isolation, while a shared development environment will host integrated changes from the entire development team.
- **Test/QA/UAT:** The production-like environment for testing deployments prior to their release to production.
- **Production:** The final production environment.

Synapse workspaces

For each Synapse workspace in your solution, the environment should include a production workspace and at least one non-production workspace for development and test/QA/UAT. Use the same name for all pools and artifacts across environments. Consistent naming will ease the promotion of workspaces to other environments.

Promoting a workspace to another workspace is a two-part process:

1. Use an Azure Resource Manager template (ARM template) to create or update workspace resources.
2. Migrate artifacts like SQL scripts, notebooks, Spark job definitions, pipelines, datasets, and data flows by using Azure Synapse continuous integration and delivery (CI/CD) tools in Azure DevOps or on GitHub.

Azure DevOps or GitHub

Ensure that integration with Azure DevOps or GitHub is properly set up. Design a repeatable process that releases changes across development, Test/QA/UAT, and production environments.

> **IMPORTANT**
>
> We recommend that sensitive configuration data always be stored securely in Azure Key Vault. Use Azure Key Vault to maintain a central, secure location for sensitive configuration data, like database connection strings. That way, appropriate services can access configuration data from within each environment.

Next steps

In the next article in the *Azure Synapse success by design* series, learn how to evaluate your team of skilled resources that will implement your Azure Synapse solution.

Synapse implementation success methodology: Evaluate team skill sets

8/19/2022 • 7 minutes to read • Edit Online

> **NOTE**
> This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

Solution development requires a team comprising individuals with many different skills. It's important for the success of your solution that your team has the necessary skills to successfully complete their assigned tasks. This evaluation takes an honest and critical look at the skill level of your project resources, and it provides you with a list of roles that are often required during the implementation of an Azure Synapse solution. Your team needs to possess relevant experience and skills to complete their assigned project tasks within the expected time frame.

Microsoft learning level definitions

This article uses the Microsoft standard level definitions for describing learning levels.

LEVEL	DESCRIPTION
100	Assumes little or no expertise with the topic, and covers topic concepts, functions, features, and benefits.
200	Assumes 100-level knowledge and provides specific details about the topic.
300	*Advanced material.* Assumes 200-level knowledge, in-depth understanding of features in a real-world environment, and strong coding skills. Provides a detailed technical overview of a subset of product/technology features, covering architecture, performance, migration, deployment, and development.
400	*Expert material.* Assumes a deep level of technical knowledge and experience, and a detailed, thorough understanding of the topic. Provides expert-to-expert interaction and coverage of specialized topics.

Roles, resources, and readiness

Successfully delivering an Azure Synapse solution involves many different roles and skill sets. This topic describes roles commonly required to implement a successful project. Not all of these roles will be required for all projects, and not all of these roles will be required for the entire duration of the project. However, these roles will be required to complete some critical project tasks. You should evaluate the skill level of the individuals executing tasks to ensure their success in completing their job.

Refer to your project plan and verify that these resources and roles were identified. Also, check to see if your project plan identifies other resources and roles. In many cases, you may find that individuals belong to more

than one role. For example, the Azure administrator could also be your Azure network administrator. It's also possible that a role in your organization is split between multiple individuals. For example, the Synapse administrator doesn't get involved in Synapse SQL security. In this case, adjust your evaluation accordingly.

Evaluate the following points.

- Identify the roles that will be required by your solution implementation.
- Identify the specific individuals in your project that will fulfill each role.
- Identify the specific project tasks that will be performed by each individual.
- Assign a learning level to each individual for their tasks and roles.

Typically, a successful implementation requires that each individual has at least a level-300 proficiency for the tasks they'll perform. It's highly recommended that individuals at level-200 (or below) be provided with guidance and instruction to raise their level of understanding prior to beginning their project tasks. In this case, involve a level-300 (or above) individual to mentor and review. It's recommended that you adjust the project plan timeline and effort estimates to factor in learning new skills.

> **NOTE**
> We recommend you align your roles with the built-in roles. There are two sets of built-in roles: RBAC roles for Azure Synapse and RBAC roles built into Azure. These two sets of built-in roles and permissions are independent.

Azure administrator

The *Azure administrator* manages administrative aspects of Azure. They're responsible for subscriptions, region identification, resource groups, monitoring, and portal access. They also provision resources, like resource groups, storage accounts, Azure Data Factory (ADF), Microsoft Purview, and more.

Security administrator

The *security administrator* must have local knowledge of the existing security landscape and requirements. This role collaborates with the Synapse administrator, Synapse database administrator, Synapse Spark administrator, and other roles to set up security requirements. The security administrator could also be an Azure Active Directory (Azure AD) administrator.

Network administrator

The *network administrator* must have local knowledge of the existing networking landscape and requirements. This role requires Azure networking skills and Synapse networking skills.

Synapse administrator

The *Synapse administrator* is responsible for the administration of the overall Azure Synapse environment. This role is responsible for the availability and scale of workspace resources, data lake administration, analytics runtimes, and workspace administration and monitoring. This role works closely with all other roles to ensure access to Azure Synapse, the availability of analytics services, and sufficient scale. Other responsibilities include:

- Provision Synapse workspaces.
- Set up Azure Synapse networking and security requirements.
- Monitor Synapse workspace activity.

Synapse database administrator

The *Synapse database administrator* is responsible for the design, implementation, maintenance, and operational aspects of the SQL pools (serverless and dedicated). This role is responsible for the overall availability, consistent performance, and optimizations of the SQL pools. This role is also responsible for managing the security of the data in the databases, granting privileges over the data, and granting or denying user access. Other responsibilities include:

- Perform various dedicated SQL pool administration functions, like provisioning, scale, pauses, resumes, restores, workload management, monitoring, and others.
- Perform various dedicated SQL pool administration functions, like securing, monitoring, and others.
- Set up SQL pool database security.
- Performance tuning and troubleshooting.

Synapse Spark administrator

The *Synapse Spark administrator* is responsible for the design, implementation, maintenance, and operational aspects of the Spark pools. This role is responsible for the overall availability, consistent performance, and optimizations of the Spark pools. This role is also responsible for managing the security of the data, granting privileges over the data, and granting or denying user access. Other responsibilities include:

- Perform various dedicated Spark pool administration functions, like provisioning, monitoring, and others.
- Set up Spark pool data security.
- Notebook troubleshooting and performance.
- Pipeline Spark execution troubleshooting and performance.

Synapse SQL pool database developer

The *Synapse pool database developer* is responsible for database design and development. For dedicated SQL pools, responsibilities include table structure and indexing, developing database objects, and schema design. For serverless SQL pools, responsibilities include external tables, views, and schema design. Other responsibilities include:

- Logical and physical database design.
- Table design, including distribution, indexing, and partitioning.
- Programming object design and development, including stored procedures and functions.
- Design and development of other performance optimizations, including materialized views, workload management, and more.
- Design and implementation of data protection, including data encryption.
- Design and implementation of access control, including object-level security, row-level security, column-level security, dynamic data masking, and Synapse role-based access control.
- Monitoring, auditing, performance tuning and troubleshooting.

Spark developer

The *Spark developer* is responsible for creating notebooks and executing Spark processing by using Spark pools.

Data integration administrator

The *Data integration administrator* is responsible for setting up and securing data integration by using Synapse pipelines, ADF, or third-party integration tools, and for performing all configuration and security functions to support the data integration tools.

For Synapse pipelines and ADF, other responsibilities include setting up the integration runtime (IR), self-hosted integration runtime (SHIR), and/or SSIS integration runtime (SSIS-IR). Knowledge of virtual machine provisioning - on-premises or in Azure - may be required.

Data integration developer

The *Data integration developer* is responsible for developing ETL/ELT and other data integration processes by using the solution's selected data integration tools.

Data consumption tools administrator

The *Data consumption tools administrator* is responsible for the data consumption tools. Tools can include Microsoft Power BI, Microsoft Excel, Tableau, and others. The administrator of each tool will need to set up

permissions to grant access to data in Azure Synapse.

Data engineer

The *Data engineer* role is responsible for implementing data-related artifacts, including data ingestion pipelines, cleansing and transformation activities, and data stores for analytical workloads. It involves using a wide range of data platform technologies, including relational and non-relational databases, file stores, and data streams.

Data engineers are responsible for ensuring that the privacy of data is maintained within the cloud, and spanning from on-premises to the cloud data stores. They also own the management and monitoring of data stores and data pipelines to ensure that data loads perform as expected.

Data scientist

The *Data scientist* derives value and insights from data. Data scientists find innovative ways to work with data and help teams achieve a rapid return on investment (ROI) on analytics efforts. They work with data curation and advanced search, matching, and recommendation algorithms. Data scientists need access to the highest quality data and substantial amounts of computing resources to extract deep insights.

Data analyst

The *Data analyst* enables businesses to maximize the value of their data assets. They transform raw data into relevant insights based on identified business requirements. Data analysts are responsible for designing and building scalable data models, cleaning, and transforming data, and presenting advanced analytics in reports and visualizations.

Azure DevOps engineer

The *Azure DevOps engineer* is responsible for designing and implementing strategies for collaboration, code, infrastructure, source control, security, compliance, continuous integration, testing, delivery, and monitoring of an Azure Synapse project.

Learning resources and certifications

If you're interested to learn about Microsoft Certifications that may help assess your team's readiness, browse the available certifications for Azure Synapse Analytics.

To complete online, self-paced training, browse the available learning paths and modules for Azure Synapse Analytics.

Next steps

In the next article in the *Azure Synapse success by design* series, learn how to perform an operational readiness review to evaluate your solution for its preparedness to provide optimal services to users.

Synapse implementation success methodology: Perform operational readiness review

8/19/2022 • 5 minutes to read • Edit Online

> **NOTE**
> This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

Once you build an Azure Synapse Analytics solution and it's ready to deploy, it's important to ensure the operational readiness of that solution. Performing an operational readiness review evaluates the solution for its preparedness to provide optimal services to users. Organizations that invest time and resources in assessing operational readiness before launch have a much higher rate of success. It's also important to conduct an operational readiness review periodically post deployment - perhaps annually - to ensure there isn't any drift from operational expectations.

Process and focus areas

Process and focus areas include service operational goals, solution readiness, security, monitoring, high availability (HA) and disaster recovery (DR).

Service operational goals

Document service expectations from the customer's point of view, and get buy-in from the business on these service expectations. Make any necessary modifications to meet business goals and objectives of the service.

The service level agreement (SLA) of each Azure service varies based on the service. For example, Microsoft guarantees a specific monthly uptime percentage. For more information, see SLA for Azure Synapse Analytics. Ensure these SLAs align with your own business SLAs and document any gaps. It's also important to define any operational level agreements (OLAs) between different teams and ensure that they align with the SLAs.

Solution readiness

It's important to review solution readiness by using the following points.

- Describe the entire solution architecture calling out critical functionalities of different components and how they interact with each other.
- Document scalability aspects of your solution. Include specific details about the effort involved in scaling and the impact of it on business. Consider whether it can respond to sudden surges of user activity. Bear in mind that Azure Synapse provides functionality for scaling with minimal downtime.
- Document any single points of failure in your solution, along with how to recover should such failures occur. Include the impact of such failures on dependent services in order to minimize the impact.
- Document all dependent services on the solution and their impact.

Security

Data security and privacy are non-negotiable. Azure Synapse implements a multi-layered security architecture for end-to-end protection of your data. Review security readiness by using the following points.

- **Authentication:** Ensure Azure Active Directory (Azure AD) authentication is used whenever possible. If non-Azure AD authentication is used, ensure strong password mechanisms are in place and that passwords are rotated on a regular basis. For more information, see Password Guidance. Ensure monitoring is in place to

detect suspicious actions related to user authentication. Consider using Azure Identity Protection to automate the detection and remediation of identity-based risks.

- **Access control:** Ensure proper access controls are in place following the principle of least privilege. Use security features available with Azure services to strengthen the security of your solution. For example, Azure Synapse provides granular security features, including row-level security (RLS), column-level security, and dynamic data masking. For more information, see Azure Synapse Analytics security white paper: Access control.
- **Threat protection:** Ensure proper threat detection mechanisms are place to prevent, detect, and respond to threats. Azure Synapse provides SQL Auditing, SQL Threat Detection, and Vulnerability Assessment to audit, protect, and monitor databases. For more information, see Azure Synapse Analytics security white paper: Threat detection.

For more information, see the Azure Synapse Analytics security white paper.

Monitoring

Set and document expectations for monitoring readiness with your business. These expectations should describe:

- How to monitor the entire user experience, and whether it includes monitoring of a single-user experience.
- The specific metrics of each service to monitor.
- How and who to notify about poor user experience.
- Details of proactive health checks.
- Any mechanisms that are in place that automate actions in response to incidents, for example, raising tickets automatically.

Consider using Azure Monitor to collect, analyze, and act on telemetry data from your Azure and on-premises environments. Azure Monitor helps you maximize performance and availability of your applications by proactively identify problems in seconds.

List all the important metrics to monitor for each service in your solution along with their acceptable thresholds. For example, the following list includes important metrics to monitor for a dedicated SQL pool:

- `DWULimit`
- `DWUUsed`
- `AdaptiveCacheHitPercent`
- `AdaptiveCacheUsedPercent`
- `LocalTempDBUsedPercent`
- `ActiveQueries`
- `QueuedQueries`

Consider using Azure Service Health to notify you about Azure service incidents and planned maintenance. That way, you can take action to mitigate downtime. You can set up customizable cloud alerts and use a personalized dashboard to analyze health issues, monitor the impact to your cloud resources, get guidance and support, and share details and updates.

Lastly, ensure proper notifications are set up to notify appropriate people when incidents occur. Incidents could be proactive, such as when a certain metric exceeds a threshold, or reactive, such as a failure of a component or service. For more information, see Overview of alerts in Microsoft Azure.

High availability

Define and document *recovery time objective (RTO)* and *recovery point objective (RPO)* for your solution. RTO is how soon the service will be available to users, and RPO is how much data loss would occur in the event of a failover.

Each of the Azure services publishes a set of guidelines and metrics on the expected high availability (HA) of the service. Ensure these HA metrics align with your business expectations. when they don't align, customizations may be necessary to meet your HA requirements. For example, Azure Synapse dedicated SQL pool supports an eight-hour RPO with automatic restore points. If that RPO isn't sufficient, you can set up user-defined restore points with an appropriate frequency to meet your RPO needs. For more information, see Backup and restore in Azure Synapse dedicated SQL pool.

Disaster recovery

Define and document a detailed process for disaster recovery (DR) scenarios. DR scenarios can include a failover process, communication mechanisms, escalation process, war room setup, and others. Also document the process for identifying the causes of outages and the steps to take to recover from disasters.

Use the built-in DR mechanisms available with Azure services for building your DR process. For example, Azure Synapse performs a standard geo-backup of SQL dedicated pools once every day to a paired data center. You can use a geo-backup to recover from a disaster at the primary location. You can also set up Azure Data Lake Storage (ADLS) to copy data to another Azure region that's hundreds of miles apart. If there's a disaster at the primary location, a failover can be initiated to transform the secondary storage location into the primary storage location. For more information, see Disaster recovery and storage account failover.

Next steps

In the next article in the *Azure Synapse success by design* series, learn how to perform monitoring of your Azure Synapse solution.

Synapse implementation success methodology: Perform user readiness and onboarding plan review

8/19/2022 • 2 minutes to read • Edit Online

> **NOTE**
> This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

Training technical people, like administrators and developers, is important to deliver success. Don't overlook that you must also extend training to include end users. Review the use cases and roles identified during the assessment stage, project planning, and solution development to ensure that *everyone* is readied for success.

Evaluate your project plan and prepare an onboarding plan for each group of users, including:

- Big data analytics users.
- Structured data analytics users.
- Users of each of your identified data consumption tools.
- Operations support.
- Help desk and user support.

Onboarding and readiness

It's unrealistic to expect that users figure out how to use Azure Synapse, even when they have experience with similar technologies. So, plan to reach out to your users to ensure a smooth transition for them to the new environment. Specifically, ensure that:

- Users understand what Azure Synapse does and how it does it.
- Users understand how to use the Azure Synapse service or the platform that uses it.
- Onboarding of users is a consistent and continuous process.
- Users see and understand the value of the new environment.

The onboarding of users starts with explanatory sessions or technical workshops. It also involves giving them access to the platform. The onboarding process can span several months depending on the complexity of the solution, and it should set the right tone for future interactions with the Azure Synapse platform and services.

Take tracking steps and make sure users are capable of completing a set of core tasks that will form part of their daily operations. These tasks will be specific to different groups of user groups, roles, and use cases. Be sure to identify:

- The core actions users need to be able to perform.
- The steps users must take to perform each action.

Focus your efforts on the tasks that users struggle the most with. Be sure to provide straightforward instructions and processes, yet be mindful of how long it takes for users to complete specific tasks. It's always a good idea to request qualitative feedback to better track user readiness and improve the onboarding experience.

Next steps

In the next article in the *Azure Synapse success by design* series, learn how to monitor your Azure Synapse solution.

Synapse implementation success methodology: Perform monitoring review

8/19/2022 • 2 minutes to read • Edit Online

> **NOTE**
> This article forms part of the *Azure Synapse implementation success by design* series of articles. For an overview of the series, see Azure Synapse implementation success by design.

Monitoring is a key part of the operationalization of any Azure solution. This article provides guidance on reviewing and configuring the monitoring of your Azure Synapse Analytics environment. Key to this activity is the identification of what needs to be monitored and who needs to review the monitoring results.

Using your solution requirements and other data collected during the assessment stage and solution development, build a list of important behaviors and activities that need to be monitored in your production environment. As you build this list, identify the groups of users that will need access to monitoring information and build the procedures they can follow to respond to monitoring results.

You can use Azure Monitor to provide base-level infrastructure metrics, alerts, and logs for most Azure services. Azure diagnostic logs are emitted by a resource to provide rich, frequent data about the operation of that resource. Azure Synapse can write diagnostic logs in Azure Monitor.

For more information, see Use Azure Monitor with your Azure Synapse Analytics workspace.

Monitor dedicated SQL pools

You can monitor a dedicated SQL pool by using Azure Monitor, altering, dynamic management views (DMVs), and Log Analytics.

- **Alerts:** You can set up alerts that send you an email or call a webhook when a certain metric reaches a predefined threshold. For example, you can receive an alert email when the database size grows too large. For more information, see Create alerts for Azure SQL Database and Azure Synapse Analytics using the Azure portal.
- **DMVs:** You can use DMVs to monitor workloads to help investigate query executions in SQL pools.
- **Log Analytics:** Log Analytics is a tool in the Azure portal that you can use to edit and run log queries from data collected by Azure Monitor. For more information, see Monitor workload - Azure portal.

Monitor serverless SQL pools

You can monitor a serverless SQL pool by monitoring your SQL requests in Synapse Studio. That way, you can keep an eye on the status of running requests and review details of historical requests.

Monitor Spark pools

You can monitor your Apache Spark applications in Synapse Studio. That way, you can keep an eye on the latest status, issues, and progress.

You can enable the Synapse Studio connector that's built in to Log Analytics. You can then collect and send Apache Spark application metrics and logs to your Log Analytics workspace. You can also use an Azure Monitor workbook to visualize the metrics and logs. For more information, see Monitor Apache Spark applications with

Azure Log Analytics.

Monitor pipelines

Azure Synapse allows you to create complex pipelines that automate and integrate your data movement, data transformation, and compute activities. You can author and monitor pipelines by using Synapse Studio to keep an eye on the latest status, issues, and progress of your pipelines. For more information, see Use Synapse Studio to monitor your workspace pipeline runs.

Next steps

For more information about this article, check out the following resources:

- Synapse implementation success methodology
- Use Azure Monitor with your Azure Synapse Analytics workspace

Migrate a data warehouse to a dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 3 minutes to read • Edit Online

The following sections provide an overview of what's involved with migrating an existing data warehouse solution to an Azure Synapse Analytics dedicated SQL pool.

Overview

Before you begin your migration, you should verify that Azure Synapse Analytics is the best solution for your workload. Azure Synapse Analytics is a distributed system designed to perform analytics on large data. Migrating to Azure Synapse Analytics requires some design changes that aren't difficult to understand but that might take some time to implement. If your business requires an enterprise-class data warehouse, the benefits are worth the effort. However, if you don't need the power of Azure Synapse Analytics, it's more cost-effective to use SQL Server or Azure SQL Database.

Consider using Azure Synapse Analytics when you:

- Have one or more terabytes of data.
- Plan to run analytics on substantial amounts of data.
- Need the ability to scale compute and storage.
- Want to save on costs by pausing compute resources when you don't need them.

Rather than Azure Synapse Analytics, consider other options for operational online transaction processing (OLTP) workloads that have:

- High frequency reads and writes.
- Large numbers of singleton selects.
- High volumes of single row inserts.
- Row-by-row processing needs.
- Incompatible formats (for example, JSON and XML).

Pre-migration

After you decide to migrate an existing solution to Azure Synapse Analytics, you need to plan your migration before you get started. A primary goal of planning is to ensure that your data, table schemas, and code are compatible with Azure Synapse Analytics. There are some compatibility differences between your current system and Azure Synapse Analytics that you'll need to work around. In addition, migrating large amounts of data to Azure takes time. Careful planning will speed up the process of getting your data to Azure.

Another key goal of planning is to adjust your design to ensure that your solution takes full advantage of the high query performance that Azure Synapse Analytics is designed to provide. Designing data warehouses for scale introduces unique design patterns, so traditional approaches aren't always the best. While some design adjustments can be made after migration, making changes earlier in the process will save you time later.

Migrate

Performing a successful migration requires you to migrate your table schemas, code, and data. For more detailed guidance on these topics, see the following articles:

- Consider table design
- Consider code change
- Migrate your data
- Consider workload management

More resources

The Customer Advisory Team has some great Azure Synapse Analytics (formerly Azure SQL Data Warehouse) guidance published as blog posts. For more information on migration, see Migrating data to Azure SQL Data Warehouse in practice.

For more information specifically about migrations from Netezza or Teradata to Azure Synapse Analytics, start at the first step of a seven-article sequence on migrations:

- Netezza to Azure Synapse Analytics migrations
- Teradata to Azure Synapse Analytics migrations

Migration assets from real-world engagements

For more assistance with completing this migration scenario, see the following resources. They were developed in support of a real-world migration project engagement.

TITLE/LINK	DESCRIPTION
Data Workload Assessment Model and Tool	This tool provides suggested "best fit" target platforms, cloud readiness, and application or database remediation level for a given workload. It offers simple, one-click calculation and report generation that helps to accelerate large estate assessments by providing an automated and uniform target platform decision process.
Handling data encoding issues while loading data to Azure Synapse Analytics	This blog post provides insight on some of the data encoding issues you might encounter while using PolyBase to load data to SQL Data Warehouse. This article also provides some options that you can use to overcome such issues and load the data successfully.
Getting table sizes in Azure Synapse Analytics dedicated SQL pool	One of the key tasks that an architect must perform is to get metrics about a new environment post-migration. Examples include collecting load times from on-premises to the cloud and collecting PolyBase load times. One of the most important tasks is to determine the storage size in SQL Data Warehouse compared to the customer's current platform.

The Data SQL Engineering team developed these resources. This team's core charter is to unblock and accelerate complex modernization for data platform migration projects to Microsoft's Azure data platform.

Videos

Watch how Walgreens migrated its retail inventory system with about 100 TB of data from Netezza to Azure Synapse Analytics in record time.

> **TIP**
> For more information on Synapse migrations, see Azure Synapse Analytics migration guides.

Design and performance for Teradata migrations

8/19/2022 • 19 minutes to read • Edit Online

This article is part one of a seven-part series that provides guidance on how to migrate from Teradata to Azure Synapse Analytics. The focus of this article is best practices for design and performance.

Overview

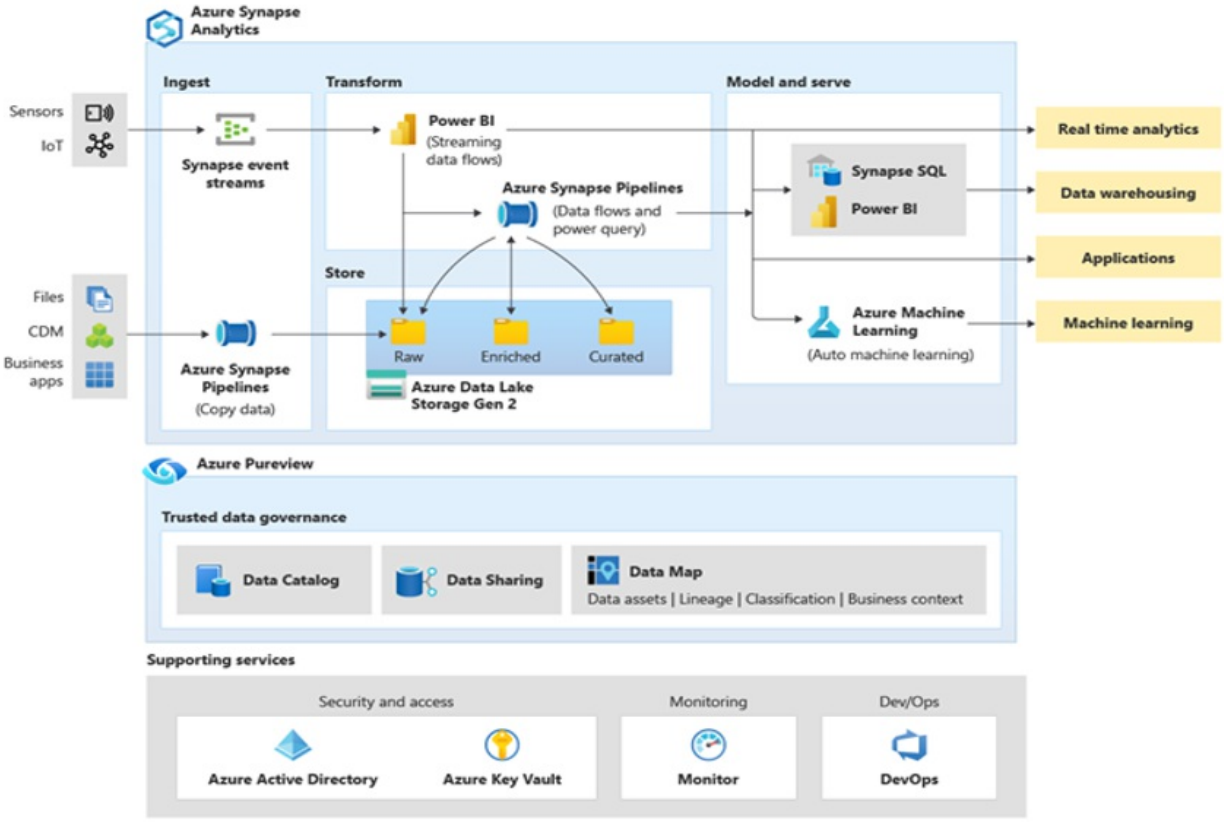
Many existing users of Teradata data warehouse systems want to take advantage of the innovations provided by modern cloud environments. Infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS) cloud environments let you delegate tasks like infrastructure maintenance and platform development to the cloud provider.

> **TIP**
> More than just a database—the Azure environment includes a comprehensive set of capabilities and tools.

Although Teradata and Azure Synapse Analytics are both SQL databases that use massively parallel processing (MPP) techniques to achieve high query performance on exceptionally large data volumes, there are some basic differences in approach:

- Legacy Teradata systems are often installed on-premises and use proprietary hardware, while Azure Synapse is cloud-based and uses Azure Storage and compute resources.
- Because storage and compute resources are separate in the Azure environment and have elastic scaling capability, those resources can be scaled upwards or downwards independently.
- You can pause or resize Azure Synapse as needed to reduce resource utilization and cost.
- Upgrading a Teradata configuration is a major task involving extra physical hardware and potentially lengthy database reconfiguration or reload.

Microsoft Azure is a globally available, highly secure, scalable cloud environment that includes Azure Synapse and an ecosystem of supporting tools and capabilities. The next diagram summarizes the Azure Synapse ecosystem.



Azure Synapse provides best-of-breed relational database performance by using techniques such as MPP and multiple levels of automated caching for frequently used data. You can see the results of these techniques in independent benchmarks such as the one run recently by GigaOm, which compares Azure Synapse to other popular cloud data warehouse offerings. Customers who migrate to the Azure Synapse environment see many benefits, including:

- Improved performance and price/performance.
- Increased agility and shorter time to value.
- Faster server deployment and application development.
- Elastic scalability—only pay for actual usage.
- Improved security/compliance.
- Reduced storage and disaster recovery costs.
- Lower overall TCO, better cost control, and streamlined operational expenditure (OPEX).

To maximize these benefits, migrate new or existing data and applications to the Azure Synapse platform. In many organizations, migration includes moving an existing data warehouse from a legacy on-premises platform, such as Teradata, to Azure Synapse. At a high level, the migration process includes these steps:

Preparation

- Define scope—what is to be migrated.
- Build inventory of data and processes for migration.
- Define data model changes (if any).
- Define source data extract mechanism.
- Identify the appropriate Azure and third-party tools and features to be used.
- Train staff early on the new platform.

- Set up the Azure target platform.

Migration □

- Start small and simple.
- Automate wherever possible.
- Leverage Azure built-in tools and features to reduce migration effort.
- Migrate metadata for tables and views.
- Migrate historical data to be maintained.
- Migrate or refactor stored procedures and business processes.
- Migrate or refactor ETL/ELT incremental load processes.

Post migration

- Monitor and document all stages of the process.
- Use the experience gained to build a template for future migrations.
- Re-engineer the data model if required (using new platform performance and scalability).
- Test applications and query tools.
- Benchmark and optimize query performance.

This article provides general information and guidelines for performance optimization when migrating a data warehouse from an existing Netezza environment to Azure Synapse. The goal of performance optimization is to achieve the same or better data warehouse performance in Azure Synapse after schema migration.

Design considerations

Migration scope

When you're preparing to migrate from a Teradata environment, consider the following migration choices.

Choose the workload for the initial migration

Typically, legacy Teradata environments have evolved over time to encompass multiple subject areas and mixed workloads. When you're deciding where to start on a migration project, choose an area where you'll be able to:

- Prove the viability of migrating to Azure Synapse by quickly delivering the benefits of the new environment.
- Allow your in-house technical staff to gain relevant experience with the processes and tools that they'll use when they migrate other areas.
- Create a template for further migrations that's specific to the source Teradata environment and the current tools and processes that are already in place.

A good candidate for an initial migration from a Teradata environment supports the preceding items, and:

- Implements a BI/Analytics workload rather than an online transaction processing (OLTP) workload.
- Has a data model, such as a star or snowflake schema, that can be migrated with minimal modification.

> **TIP**
> Create an inventory of objects that need to be migrated, and document the migration process.

The volume of migrated data in an initial migration should be large enough to demonstrate the capabilities and benefits of the Azure Synapse environment but not too large to quickly demonstrate value. A size in the 1-10 terabyte range is typical.

For your initial migration project, minimize the risk, effort, and migration time so you can quickly see the benefits of the Azure cloud environment, confine the scope of the migration to just the data marts, such as the OLAP DB part of a Teradata warehouse. Both the lift-and-shift and phased migration approaches limit the scope of the initial migration to just the data marts and don't address broader migration aspects, such as ETL migration and historical data migration. However, you can address those aspects in later phases of the project once the migrated data mart layer is backfilled with data and the required build processes.

Lift and shift migration vs. Phased approach

In general, there are two types of migration regardless of the purpose and scope of the planned migration: lift and shift as-is and a phased approach that incorporates changes.

Lift and shift

In a lift and shift migration, an existing data model, like a star schema, is migrated unchanged to the new Azure Synapse platform. This approach minimizes risk and migration time by reducing the work needed to realize the benefits of moving to the Azure cloud environment. Lift and shift migration is a good fit for these scenarios:

- You have an existing Teradata environment with a single data mart to migrate, or
- You have an existing Teradata environment with data that's already in a well-designed star or snowflake schema, or
- You're under time and cost pressures to move to a modern cloud environment.

> **TIP**
> Lift and shift is a good starting point, even if subsequent phases implement changes to the data model.

Phased approach that incorporates changes

If a legacy data warehouse has evolved over a long period of time, you might need to re-engineer it to maintain the required performance levels. You might also have to re-engineer to support new data like Internet of Things (IoT) streams. As part of the re-engineering process, migrate to Azure Synapse to get the benefits of a scalable cloud environment. Migration can also include a change in the underlying data model, such as a move from an Inmon model to a data vault.

Microsoft recommends moving your existing data model as-is to Azure (optionally using a VM Teradata instance in Azure) and using the performance and flexibility of the Azure environment to apply the re-engineering changes. That way, you can use Azure's capabilities to make the changes without impacting the existing source system.

Use an Azure VM Teradata instance as part of a migration

When migrating from an on-premises Teradata environment, you can leverage cloud storage and elastic scalability in Azure to create a Teradata instance within a VM. This approach collocates the Teradata instance with the target Azure Synapse environment. You can use standard Teradata utilities, such as Teradata Parallel Data Transporter, to efficiently move the subset of Teradata tables being migrated onto the VM instance. Then, all further migration tasks can occur within the Azure environment. This approach has several benefits:

- After the initial replication of data, the source system isn't affected by the migration tasks.
- Familiar Teradata interfaces, tools, and utilities are available within the Azure environment.
- The Azure environment sidesteps any potential issues with network bandwidth availability between the on-premises source system and the cloud target system.
- Tools like Azure Data Factory can call utilities like Teradata Parallel Transporter to efficiently and rapidly

migrate data.

- You can orchestrate and control the migration process entirely within the Azure environment.

> **TIP**
> Use Azure VMs to create a temporary Teradata instance to speed up migration and minimize impact on the source system.

Use Azure Data Factory to implement a metadata-driven migration

You can automate and orchestrate the migration process by using the capabilities of the Azure environment. This approach minimizes the performance hit on the existing Netezza environment, which may already be running close to capacity.

Azure Data Factory is a cloud-based data integration service that supports creating data-driven workflows in the cloud that orchestrate and automate data movement and data transformation. You can use Data Factory to create and schedule data-driven workflows (pipelines) that ingest data from disparate data stores. Data Factory can process and transform data by using compute services such as Azure HDInsight Hadoop, Spark, Azure Data Lake Analytics, and Azure Machine Learning.

When you're planning to use Data Factory facilities to manage the migration process, create metadata that lists all the data tables to be migrated and their location.

Design differences between Teradata and Azure Synapse

As mentioned earlier, there are some basic differences in approach between Teradata and Azure Synapse Analytics databases and these differences are discussed next.

Multiple databases vs. a single database and schemas

The Netezza environment often contains multiple separate databases. For instance, there could be separate databases for: data ingestion and staging tables, core warehouse tables, and data marts (sometimes referred to as the semantic layer). ETL or ELT pipeline processes might implement cross-database joins and move data between the separate databases.

In contrast, the Azure Synapse environment contains a single database and uses schemas to separate tables into logically separate groups. We recommend that you use a series of schemas within the target Azure Synapse database to mimic the separate databases migrated from the Teradata environment. If the Teradata environment already uses schemas, you may need to use a new naming convention when you move the existing Teradata tables and views to the new environment. For example, you could concatenate the existing Teradata schema and table names into the new Azure Synapse table name, and use schema names in the new environment to maintain the original separate database names. If schema consolidation naming has dots, Azure Synapse Spark might have issues. Although you can use SQL views on top of the underlying tables to maintain the logical structures, there are potential downsides to that approach:

- Views in Azure Synapse are read-only, so any updates to the data must take place on the underlying base tables.
- There may already be one or more layers of views in existence and adding an extra layer of views could affect performance and supportability because nested views are difficult to troubleshoot.

> **TIP**
> Combine multiple databases into a single database within Azure Synapse and use schema names to logically separate the tables.

Table considerations

When you migrate tables between different environments, typically only the raw data and the metadata that describes it physically migrate. Other database elements from the source system, such as indexes, usually aren't migrated because they might be unnecessary or implemented differently in the new environment. Performance optimizations in the source environment, such as indexes, indicate where you might add performance optimization in the new environment. For example, if a table within the source Teradata environment has a non-unique secondary index (NUSI), that suggests that a non-clustered index should be created within Azure Synapse. Other native performance optimization techniques like table replication may be more applicable than straight like-for-like index creation.

> **TIP**
> Existing indexes indicate candidates for indexing in the migrated warehouse.

High availability for the database

Teradata supports data replication across nodes via the `FALLBACK` option, which replicates table rows that reside physically on a given node to another node within the system. This approach guarantees that data won't be lost if there's a node failure and provides the basis for failover scenarios.

The goal of the high availability architecture in Azure Synapse Analytics is to guarantee that your database is up and running 99.9% of the time, without worrying about the impact of maintenance operations and outages. For more information on the SLA, see SLA for Azure Synapse Analytics. Azure automatically handles critical servicing tasks such as patching, backups, and Windows and SQL upgrades. Azure also automatically handles unplanned events such as failures in the underlying hardware, software, or network.

Data storage in Azure Synapse is automatically backed up with snapshots. These snapshots are a built-in feature of the service that creates restore points. You don't have to enable this capability. Users can't currently delete automatic restore points that the service uses to maintain service level agreements (SLAs) for recovery.

Azure Synapse Dedicated SQL pool takes snapshots of the data warehouse throughout the day to create restore points that are available for seven days. This retention period can't be changed. Azure Synapse supports an eight-hour recovery point objective (RPO). You can restore your data warehouse in the primary region from any one of the snapshots taken in the past seven days. If you require more granular backups, you can use another user-defined option.

Unsupported Teradata table types

Teradata supports special table types for time-series and temporal data. The syntax and some of the functions for these table types aren't directly supported in Azure Synapse. However, you can migrate the data into a standard table in Azure Synapse by mapping to appropriate data types and indexing or partitioning the date/time column.

> **TIP**
> Standard tables in Azure Synapse can support migrated Teradata time-series and temporal data.

Teradata implements temporal query functionality by using query rewriting to add additional filters within a temporal query to limit the applicable date range. If you plan to migrate this functionality from the source Teradata environment, then add the additional filtering into the relevant temporal queries.

The Azure environment supports time series insights for complex analytics on time-series data at scale. This functionality is aimed at IoT data analysis applications.

SQL DML syntax differences

SQL Data Manipulation Language (DML) syntax differences exist between Teradata SQL and Azure Synapse T-SQL:

- `QUALIFY`: Teradata supports the `QUALIFY` operator. For example:

  ```
  SELECT col1
  FROM tab1
  WHERE col1='XYZ'
  QUALIFY ROW_NUMBER () OVER (PARTITION by
  col1 ORDER BY col1) = 1;
  ```

  The equivalent Azure Synapse syntax is:

  ```
  SELECT * FROM (
  SELECT col1, ROW_NUMBER () OVER (PARTITION by col1 ORDER BY col1) rn
  FROM tab1 WHERE col1='XYZ'
  ) WHERE rn = 1;
  ```

- Date arithmetic: Azure Synapse has operators such as `DATEADD` and `DATEDIFF`, which can be used on `DATE` or `DATETIME` fields. Teradata supports direct subtraction on dates such as `SELECT DATE1 - DATE2 FROM...`

- `GROUP BY`: for the `GROUP BY` ordinal, explicitly provide a T-SQL column name.

- `LIKE ANY`: Teradata supports `LIKE ANY` syntax such as:

  ```
  SELECT * FROM CUSTOMER
  WHERE POSTCODE LIKE ANY
  ('CV1%', 'CV2%', 'CV3%');
  ```

  The equivalent in Azure Synapse syntax is:

  ```
  SELECT * FROM CUSTOMER
  WHERE
  (POSTCODE LIKE 'CV1%') OR (POSTCODE LIKE 'CV2%') OR (POSTCODE LIKE 'CV3%');
  ```

- Depending on system settings, character comparisons in Teradata may be case insensitive by default. In Azure Synapse, character comparisons are always case sensitive.

Functions, stored procedures, triggers, and sequences

When migrating a data warehouse from a mature environment like Teradata, you probably need to migrate elements other than simple tables and views. Examples include functions, stored procedures, triggers, and sequences. Check whether tools within the Azure environment can replace the functionality of functions, stored procedures, and sequences because it's usually more efficient to use built-in Azure tools than to recode those elements for Azure Synapse.

As part of your preparation phase, create an inventory of objects that need to be migrated, define a method for handling them, and allocate appropriate resources in your migration plan.

Data integration partners offer tools and services that can automate the migration of functions, stored procedures, and sequences.

The following sections further discuss the migration of functions, stored procedures, and sequences.

Functions

As with most database products, Teradata supports system and user-defined functions within a SQL implementation. When you migrate a legacy database platform to Azure Synapse, common system functions can usually be migrated without change. Some system functions might have a slightly different syntax, but any required changes can be automated.

For Teradata system functions or arbitrary user-defined functions that have no equivalent in Azure Synapse, recode those functions using a target environment language. Azure Synapse uses the Transact-SQL language to implement user-defined functions.

Stored procedures

Most modern database products support storing procedures within the database. Teradata provides the SPL language for this purpose. A stored procedure typically contains both SQL statements and procedural logic, and returns data or a status.

Azure Synapse supports stored procedures using T-SQL, so you need to recode any migrated stored procedures in that language.

Triggers

Azure Synapse doesn't support trigger creation, but trigger creation can be implemented using Azure Data Factory.

Sequences

Azure Synapse handles sequences in a similar way to Teradata, and you can implement sequences using IDENTITY columns or SQL code that generates the next sequence number in a series. A sequence provides unique numeric values that you can use as surrogate key values for primary keys.

Extract metadata and data from a Teradata environment

Data Definition Language (DDL) generation

The ANSI SQL standard defines the basic syntax for Data Definition Language (DDL) commands. Some DDL commands, such as `CREATE TABLE` and `CREATE VIEW`, are common to both Teradata and Azure Synapse but also provide implementation-specific features such as indexing, table distribution, and partitioning options.

You can edit existing Teradata `CREATE TABLE` and `CREATE VIEW` scripts to achieve equivalent definitions in Azure Synapse. To do so, you might need to use modified data types and remove or modify Teradata-specific clauses such as `FALLBACK`.

However, all the information that specifies the current definitions of tables and views within the existing Teradata environment is maintained within system catalog tables. These tables are the best source of this information, as it's guaranteed to be up to date and complete. User-maintained documentation may not be in sync with the current table definitions.

Within the Teradata environment, system catalog tables specify the current table and view definition. Unlike user-maintained documentation, system catalog information is always complete and in sync with current table definitions. By using views into the catalog such as `DBC.ColumnsV`, you can access system catalog information to generate `CREATE TABLE` DDL statements that create equivalent tables in Azure Synapse.

> **TIP**
> Use existing Teradata metadata to automate the generation of `CREATE TABLE` and `CREATE VIEW` DDL for Azure Synapse.

You can also use third-party migration and ETL tools that process system catalog information to achieve similar results.

Data extraction from Teradata

You can extract raw table data from Teradata tables to flat delimited files, such as CSV files, using standard Teradata utilities like Basic Teradata Query (BTEQ), Teradata FastExport, or Teradata Parallel Transporter (TPT). Use TPT to extract table data as efficiently as possible. TPT uses multiple parallel FastExport streams to achieve the highest throughput.

> **TIP**
> Use Teradata Parallel Transporter for the most efficient data extract.

Call TPT directly from Azure Data Factory. This is the recommended approach for data migration of Teradata on-premises instances and Teradata instances that run within a VM in the Azure environment.

Extracted data files should contain delimited text in CSV, Optimized Row Columnar (ORC), or Parquet format.

For more information about migrating data and ETL from a Teradata environment, see Data migration, ETL, and load for Teradata migrations.

Performance recommendations for Teradata migrations

The goal of performance optimization is same or better data warehouse performance after migration to Azure Synapse.

> **TIP**
> Prioritize familiarity with the tuning options in Azure Synapse at the start of a migration.

Differences in performance tuning approach

This section highlights low-level performance tuning implementation differences between Teradata and Azure Synapse.

Data distribution options

For performance, Azure Synapse was designed with multi-node architecture and uses parallel processing. To optimize individual table performance in Azure Synapse, you can define a data distribution option in `CREATE TABLE` statements using the `DISTRIBUTION` statement. For example, you can specify a hash-distributed table, which distributes table rows across compute nodes by using a deterministic hash function. The aim is to reduce the amount of data moved between processing nodes when executing a query.

For large table to large table joins, hash distribute one or, ideally, both tables on one of the join columns—which has a wide range of values to help ensure an even distribution. Perform join processing locally because the data rows that will be joined are collocated on the same processing node.

Azure Synapse also supports local joins between a small table and a large table through small table replication. For instance, consider a small dimension table and a large fact table within a star schema model. Azure Synapse can replicate the smaller dimension table across all nodes to ensure that the value of any join key for the large table has a matching, locally available dimension row. The overhead of dimension table replication is relatively low for a small dimension table. For large dimension tables, a hash distribution approach is more appropriate. For more information on data distribution options, see Design guidance for using replicated tables and Guidance for designing distributed tables.

Data indexing

Azure Synapse supports several user-definable indexing options that are different from the indexing options implemented in Teradata. For more information about the different indexing options in Azure Synapse, see Indexes on dedicated SQL pool tables.

Existing indexes within the source Teradata environment provide a useful indication of data usage and the candidate columns for indexing in the Azure Synapse environment.

Data partitioning

In an enterprise data warehouse, fact tables can contain billions of rows. Partitioning optimizes the maintenance and query performance of these tables by splitting them into separate parts to reduce the amount of data

processed. In Azure Synapse, the `CREATE TABLE` statement defines the partitioning specification for a table. Only partition very large tables and ensure each partition contains at least 60 million rows.

You can only use one field per table for partitioning. That field is often a date field because many queries are filtered by date or date range. It's possible to change the partitioning of a table after initial load by using the `CREATE TABLE AS` (CTAS) statement to recreate the table with a new distribution. For a detailed discussion of partitioning in Azure Synapse, see Partitioning tables in dedicated SQL pool.

Data table statistics

You should ensure that statistics on data tables are up to date by building in a statistics step to ETL/ELT jobs.

PolyBase or COPY INTO for data loading

PolyBase supports efficient loading of large amounts of data to a data warehouse by using parallel loading streams. For more information, see PolyBase data loading strategy.

COPY INTO also supports high-throughput data ingestion, and:

- Data retrieval from all files within a folder and subfolders.
- Data retrieval from multiple locations in the same storage account. You can specify multiple locations by using comma separated paths.
- Azure Data Lake Storage (ADLS) and Azure Blob Storage.
- CSV, PARQUET, and ORC file formats.

Workload management

Running mixed workloads can pose resource challenges on busy systems. A successful workload management scheme effectively manages resources, ensures highly efficient resource utilization, and maximizes return on investment (ROI). Workload classification, workload importance, and workload isolation give more control over how workload utilizes system resources.

The workload management guide describes the techniques to analyze the workload, manage and monitor workload importance, and the steps to convert a resource class to a workload group. Use the Azure portal and T-SQL queries on DMVs to monitor the workload to ensure that the applicable resources are efficiently utilized.

Next steps

To learn about ETL and load for Teradata migration, see the next article in this series: Data migration, ETL, and load for Teradata migrations.

Data migration, ETL, and load for Teradata migrations

8/19/2022 • 15 minutes to read • Edit Online

This article is part two of a seven-part series that provides guidance on how to migrate from Teradata to Azure Synapse Analytics. The focus of this article is best practices for ETL and load migration.

Data migration considerations

Initial decisions for data migration from Teradata

When migrating a Teradata data warehouse, you need to ask some basic data-related questions. For example:

- Should unused table structures be migrated?
- What's the best migration approach to minimize risk and user impact?
- When migrating data marts: stay physical or go virtual?

The next sections discuss these points within the context of migration from Teradata.

Migrate unused tables?

> **TIP**
> In legacy systems, it's not unusual for tables to become redundant over time—these don't need to be migrated in most cases.

It makes sense to only migrate tables that are in use in the existing system. Tables that aren't active can be archived rather than migrated, so that the data is available if necessary in future. It's best to use system metadata and log files rather than documentation to determine which tables are in use, because documentation can be out of date.

If enabled, Teradata system catalog tables and logs contain information that can determine when a given table was last accessed—which can in turn be used to decide whether a table is a candidate for migration.

Here's an example query on `DBC.Tables` that provides the date of last access and last modification:

```
SELECT TableName, CreatorName, CreateTimeStamp, LastAlterName,
LastAlterTimeStamp, AccessCount, LastAccessTimeStamp
FROM DBC.Tables t
WHERE DataBaseName = 'databasename'
```

If logging is enabled and the log history is accessible, other information, such as SQL query text, is available in table DBQLogTbl and associated logging tables. For more information, see Teradata log history.

What is the best migration approach to minimize risk and impact on users?

This question comes up frequently because companies may want to lower the impact of changes on the data warehouse data model to improve agility. Companies often see an opportunity to further modernize or transform their data during an ETL migration. This approach carries a higher risk because it changes multiple factors simultaneously, making it difficult to compare the outcomes of the old system versus the new. Making data model changes here could also affect upstream or downstream ETL jobs to other systems. Because of that risk, it's better to redesign on this scale after the data warehouse migration.

Even if a data model is intentionally changed as part of the overall migration, it's good practice to migrate the existing model as-is to Azure Synapse, rather than do any re-engineering on the new platform. This approach minimizes the effect on existing production systems, while benefiting from the performance and elastic scalability of the Azure platform for one-off re-engineering tasks.

When migrating from Teradata, consider creating a Teradata environment in a VM within Azure as a stepping-stone in the migration process.

> **TIP**
> Migrate the existing model as-is initially, even if a change to the data model is planned in the future.

Use a VM Teradata instance as part of a migration

One optional approach for migrating from an on-premises Teradata environment is to leverage the Azure environment to create a Teradata instance in a VM within Azure, collocated with the target Azure Synapse environment. This is possible because Azure provides cheap cloud storage and elastic scalability.

With this approach, standard Teradata utilities, such as Teradata Parallel Data Transporter—or third-party data replication tools, such as Attunity Replicate—can be used to efficiently move the subset of Teradata tables that need to be migrated to the VM instance. Then, all migration tasks can take place within the Azure environment. This approach has several benefits:

- After the initial replication of data, migration tasks don't impact the source system.
- The Azure environment has familiar Teradata interfaces, tools, and utilities.
- The Azure environment provides network bandwidth availability between the on-premises source system and the cloud target system.
- Tools like Azure Data Factory can efficiently call utilities like Teradata Parallel Transporter to migrate data quickly and easily.
- The migration process is orchestrated and controlled entirely within the Azure environment.

When migrating data marts: stay physical or go virtual?

> **TIP**
> Virtualizing data marts can save on storage and processing resources.

In legacy Teradata data warehouse environments, it's common practice to create several data marts that are structured to provide good performance for ad hoc self-service queries and reports for a given department or business function within an organization. As such, a data mart typically consists of a subset of the data warehouse and contains aggregated versions of the data in a form that enables users to easily query that data with fast response times via user-friendly query tools such as Microsoft Power BI, Tableau, or MicroStrategy. This form is typically a dimensional data model. One use of data marts is to expose the data in a usable form, even if the underlying warehouse data model is something different, such as a data vault.

You can use separate data marts for individual business units within an organization to implement robust data security regimes, by only allowing users to access specific data marts that are relevant to them, and eliminating, obfuscating, or anonymizing sensitive data.

If these data marts are implemented as physical tables, they'll require additional storage resources to store them, and additional processing to build and refresh them regularly. Also, the data in the mart will only be as up to date as the last refresh operation, and so may be unsuitable for highly volatile data dashboards.

> **TIP**
> The performance and scalability of Azure Synapse enables virtualization without sacrificing performance.

With the advent of relatively low-cost scalable MPP architectures, such as Azure Synapse, and the inherent performance characteristics of such architectures, it may be that you can provide data mart functionality without having to instantiate the mart as a set of physical tables. This is achieved by effectively virtualizing the data marts via SQL views onto the main data warehouse, or via a virtualization layer using features such as views in Azure or the visualization products of Microsoft partners. This approach simplifies or eliminates the need for additional storage and aggregation processing and reduces the overall number of database objects to be migrated.

There's another potential benefit to this approach. By implementing the aggregation and join logic within a virtualization layer, and presenting external reporting tools via a virtualized view, the processing required to create these views is "pushed down" into the data warehouse, which is generally the best place to run joins, aggregations, and other related operations on large data volumes.

The primary drivers for choosing a virtual data mart implementation over a physical data mart are:

- More agility: a virtual data mart is easier to change than physical tables and the associated ETL processes.
- Lower total cost of ownership: a virtualized implementation requires fewer data stores and copies of data.
- Elimination of ETL jobs to migrate and simplify data warehouse architecture in a virtualized environment.
- Performance: although physical data marts have historically been more performant, virtualization products now implement intelligent caching techniques to mitigate.

Data migration from Teradata

Understand your data

Part of migration planning is understanding in detail the volume of data that needs to be migrated since that can impact decisions about the migration approach. Use system metadata to determine the physical space taken up by the "raw data" within the tables to be migrated. In this context, "raw data" means the amount of space used by the data rows within a table, excluding overheads such as indexes and compression. This is especially true for the largest fact tables since these will typically comprise more than 95% of the data.

You can get an accurate number for the volume of data to be migrated for a given table by extracting a representative sample of the data—for example, one million rows—to an uncompressed delimited flat ASCII data file. Then, use the size of that file to get an average raw data size per row of that table. Finally, multiply that average size by the total number of rows in the full table to give a raw data size for the table. Use that raw data size in your planning.

ETL migration considerations

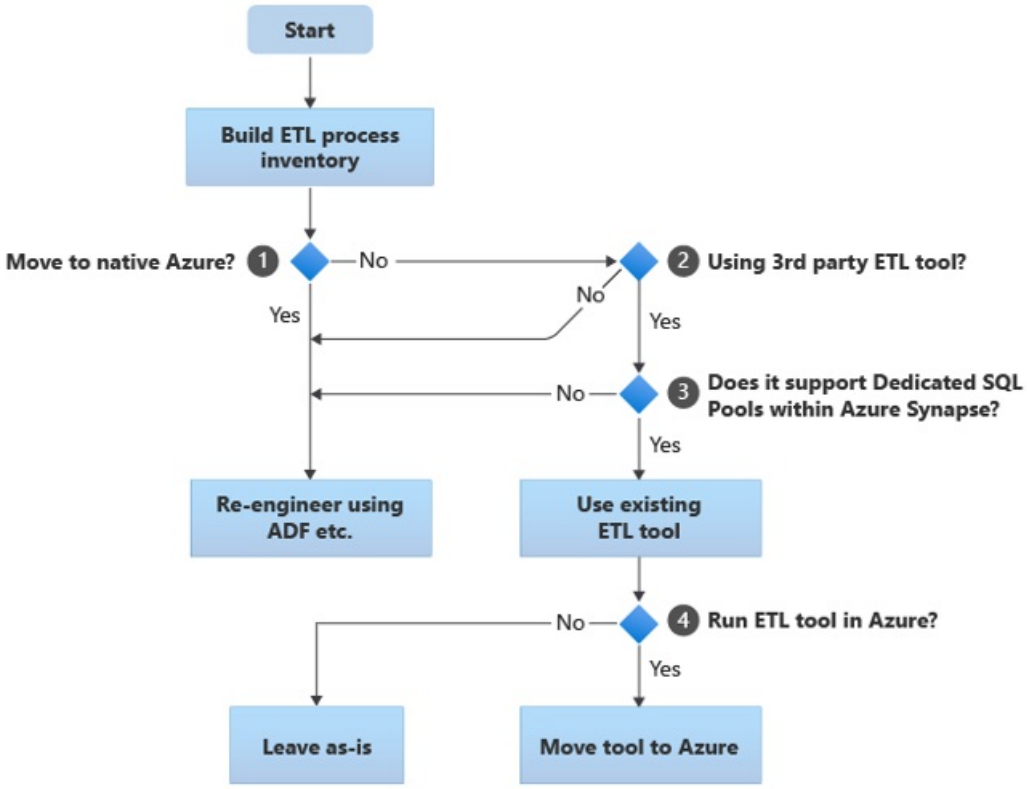
Initial decisions regarding Teradata ETL migration

> **TIP**
> Plan the approach to ETL migration ahead of time and leverage Azure facilities where appropriate.

For ETL/ELT processing, legacy Teradata data warehouses may use custom-built scripts using Teradata utilities such as BTEQ and Teradata Parallel Transporter (TPT), or third-party ETL tools such as Informatica or Ab Initio. Sometimes, Teradata data warehouses use a combination of ETL and ELT approaches that's evolved over time. When planning a migration to Azure Synapse, you need to determine the best way to implement the required ETL/ELT processing in the new environment while minimizing the cost and risk involved. To learn more about

ETL and ELT processing, see ELT vs ETL design approach.

The following sections discuss migration options and make recommendations for various use cases. This flowchart summarizes one approach:



The first step is always to build an inventory of ETL/ELT processes that need to be migrated. As with other steps, it's possible that the standard "built-in" Azure features make it unnecessary to migrate some existing processes. For planning purposes, it's important to understand the scale of the migration to be performed.

In the preceding flowchart, decision 1 relates to a high-level decision about whether to migrate to a totally Azure-native environment. If you're moving to a totally Azure-native environment, we recommend that you re-engineer the ETL processing using Pipelines and activities in Azure Data Factory or Azure Synapse Pipelines. If you're not moving to a totally Azure-native environment, then decision 2 is whether an existing third-party ETL tool is already in use.

In the Teradata environment, some or all ETL processing may be performed by custom scripts using Teradata-specific utilities like BTEQ and TPT. In this case, your approach should be to re-engineer using Data Factory.

> **TIP**
> Leverage investment in existing third-party tools to reduce cost and risk.

If a third-party ETL tool is already in use, and especially if there's a large investment in skills or several existing workflows and schedules use that tool, then decision 3 is whether the tool can efficiently support Azure Synapse as a target environment. Ideally, the tool will include "native" connectors that can leverage Azure facilities like PolyBase or COPY INTO, for the most efficient data loading. There's a way to call an external process, such as PolyBase or `COPY INTO`, and pass in the appropriate parameters. In this case, leverage existing skills and workflows, with Azure Synapse as the new target environment.

If you decide to retain an existing third-party ETL tool, there may be benefits to running that tool within the Azure environment (rather than on an existing on-premises ETL server) and having Azure Data Factory handle the overall orchestration of the existing workflows. One particular benefit is that less data needs to be downloaded from Azure, processed, and then uploaded back into Azure. So, decision 4 is whether to leave the

existing tool running as-is or move it into the Azure environment to achieve cost, performance, and scalability benefits.

Re-engineer existing Teradata-specific scripts

If some or all the existing Teradata warehouse ETL/ELT processing is handled by custom scripts that utilize Teradata-specific utilities, such as BTEQ, MLOAD, or TPT, these scripts need to be recoded for the new Azure Synapse environment. Similarly, if ETL processes were implemented using stored procedures in Teradata, then these will also have to be recoded.

> **TIP**
> The inventory of ETL tasks to be migrated should include scripts and stored procedures.

Some elements of the ETL process are easy to migrate, for example by simple bulk data load into a staging table from an external file. It may even be possible to automate those parts of the process, for example, by using PolyBase instead of fast load or MLOAD. If the exported files are Parquet, you can use a native Parquet reader, which is a faster option than PolyBase. Other parts of the process that contain arbitrary complex SQL and/or stored procedures will take more time to re-engineer.

One way of testing Teradata SQL for compatibility with Azure Synapse is to capture some representative SQL statements from Teradata logs, then prefix those queries with `EXPLAIN`, and then—assuming a like-for-like migrated data model in Azure Synapse—run those `EXPLAIN` statements in Azure Synapse. Any incompatible SQL will generate an error, and the error information can determine the scale of the recoding task.

Microsoft partners offer tools and services to migrate Teradata SQL and stored procedures to Azure Synapse.

Use third-party ETL tools

As described in the previous section, in many cases the existing legacy data warehouse system will already be populated and maintained by third-party ETL products. For a list of Microsoft data integration partners for Azure Synapse, see Data integration partners.

Data loading from Teradata

Choices available when loading data from Teradata

> **TIP**
> Third-party tools can simplify and automate the migration process and therefore reduce risk.

When it comes to migrating data from a Teradata data warehouse, there are some basic questions associated with data loading that need to be resolved. You'll need to decide how the data will be physically moved from the existing on-premises Teradata environment into Azure Synapse in the cloud, and which tools will be used to perform the transfer and load. Consider the following questions, which are discussed in the next sections.

- Will you extract the data to files, or move it directly via a network connection?
- Will you orchestrate the process from the source system, or from the Azure target environment?
- Which tools will you use to automate and manage the process?

Transfer data via files or network connection?

> **TIP**
> Understand the data volumes to be migrated and the available network bandwidth since these factors influence the migration approach decision.

Once the database tables to be migrated have been created in Azure Synapse, you can move the data to populate those tables out of the legacy Teradata system and into the new environment. There are two basic approaches:

- **File extract**: extract the data from the Teradata tables to flat files, normally in CSV format, via BTEQ, Fast Export, or Teradata Parallel Transporter (TPT). Use TPT whenever possible since it's the most efficient in terms of data throughput.

  This approach requires space to land the extracted data files. The space could be local to the Teradata source database (if sufficient storage is available), or remote in Azure Blob Storage. The best performance is achieved when a file is written locally, since that avoids network overhead.

  To minimize the storage and network transfer requirements, it's good practice to compress the extracted data files using a utility like gzip.

  Once extracted, the flat files can either be moved into Azure Blob Storage (collocated with the target Azure Synapse instance) or loaded directly into Azure Synapse using PolyBase or COPY INTO. The method for physically moving data from local on-premises storage to the Azure cloud environment depends on the amount of data and the available network bandwidth.

  Microsoft provides different options to move large volumes of data, including AZCopy for moving files across the network into Azure Storage, Azure ExpressRoute for moving bulk data over a private network connection, and Azure Data Box where the files are moved to a physical storage device that's then shipped to an Azure data center for loading. For more information, see data transfer.

- **Direct extract and load across network**: the target Azure environment sends a data extract request, normally via a SQL command, to the legacy Teradata system to extract the data. The results are sent across the network and loaded directly into Azure Synapse, with no need to land the data into intermediate files. The limiting factor in this scenario is normally the bandwidth of the network connection between the Teradata database and the Azure environment. For very large data volumes this approach may not be practical.

There's also a hybrid approach that uses both methods. For example, you can use the direct network extract approach for smaller dimension tables and samples of the larger fact tables to quickly provide a test environment in Azure Synapse. For the large volume historical fact tables, you can use the file extract and transfer approach using Azure Data Box.

Orchestrate from Teradata or Azure?

The recommended approach when moving to Azure Synapse is to orchestrate the data extract and loading from the Azure environment using Azure Synapse Pipelines or Azure Data Factory, as well as associated utilities, such as PolyBase or COPY INTO, for most efficient data loading. This approach leverages the Azure capabilities and provides an easy method to build reusable data loading pipelines.

Other benefits of this approach include reduced impact on the Teradata system during the data load process since the management and loading process is running in Azure, and the ability to automate the process by using metadata-driven data load pipelines.

Which tools can be used?

The task of data transformation and movement is the basic function of all ETL products. If one of these products is already in use in the existing Teradata environment, then using the existing ETL tool may simplify data migration from Teradata to Azure Synapse. This approach assumes that the ETL tool supports Azure Synapse as

a target environment. For more information on tools that support Azure Synapse, see Data integration partners.

If you're using an ETL tool, consider running that tool within the Azure environment to benefit from Azure cloud performance, scalability, and cost, and free up resources in the Teradata data center. Another benefit is reduced data movement between the cloud and on-premises environments.

Summary

To summarize, our recommendations for migrating data and associated ETL processes from Teradata to Azure Synapse are:

- Plan ahead to ensure a successful migration exercise.
- Build a detailed inventory of data and processes to be migrated as soon as possible.
- Use system metadata and log files to get an accurate understanding of data and process usage. Don't rely on documentation since it may be out of date.
- Understand the data volumes to be migrated, and the network bandwidth between the on-premises data center and Azure cloud environments.
- Consider using a Teradata instance in an Azure VM as a stepping stone to offload migration from the legacy Teradata environment.
- Leverage standard "built-in" Azure features to minimize the migration workload.
- Identify and understand the most efficient tools for data extraction and loading in both Teradata and Azure environments. Use the appropriate tools in each phase of the process.
- Use Azure facilities, such as Azure Synapse Pipelines or Azure Data Factory, to orchestrate and automate the migration process while minimizing impact on the Teradata system.

Next steps

To learn more about security access operations, see the next article in this series: Security, access, and operations for Teradata migrations.

Security, access, and operations for Teradata migrations

8/19/2022 • 16 minutes to read • Edit Online

This article is part three of a seven-part series that provides guidance on how to migrate from Teradata to Azure Synapse Analytics. The focus of this article is best practices for security access operations.

Security considerations

This article discusses connection methods for existing legacy Teradata environments and how they can be migrated to Azure Synapse Analytics with minimal risk and user impact.

This article assumes that there's a requirement to migrate the existing methods of connection and user/role/permission structure as-is. If not, use the Azure portal to create and manage a new security regime.

For more information on the Azure Synapse security options, see Security whitepaper.

Connection and authentication

Teradata authorization options

> **TIP**
> Authentication in both Teradata and Azure Synapse can be "in database" or through external methods.

Teradata supports several mechanisms for connection and authorization. Valid mechanism values are:

- **TD1**, which selects Teradata 1 as the authentication mechanism. Username and password are required.
- **TD2**, which selects Teradata 2 as the authentication mechanism. Username and password are required.
- **TDNEGO**, which selects one of the authentication mechanisms automatically based on the policy, without user involvement.
- **LDAP**, which selects Lightweight Directory Access Protocol (LDAP) as the authentication mechanism. The application provides the username and password.
- **KRB5**, which selects Kerberos (KRB5) on Windows clients working with Windows servers. To sign in using KRB5, the user needs to supply a domain, username, and password. The domain is specified by setting the username to `MyUserName@MyDomain`.
- **NTLM**, which selects NTLM on Windows clients working with Windows servers. The application provides the username and password.

Kerberos (KRB5), Kerberos Compatibility (KRB5C), NT LAN Manager (NTLM), and NT LAN Manager Compatibility (NTLMC) are for Windows only.

Azure Synapse authorization options

Azure Synapse supports two basic options for connection and authorization:

- **SQL authentication**: SQL authentication is via a database connection that includes a database identifier, user ID, and password plus other optional parameters. This is functionally equivalent to Teradata TD1, TD2 and default connections.
- **Azure Active Directory (Azure AD) authentication**: with Azure AD authentication, you can centrally

manage the identities of database users and other Microsoft services in one central location. Central ID management provides a single place to manage SQL Data Warehouse users and simplifies permission management. Azure AD can also support connections to LDAP and Kerberos services—for example, Azure AD can be used to connect to existing LDAP directories if these are to remain in place after migration of the database.

Users, roles, and permissions

Overview

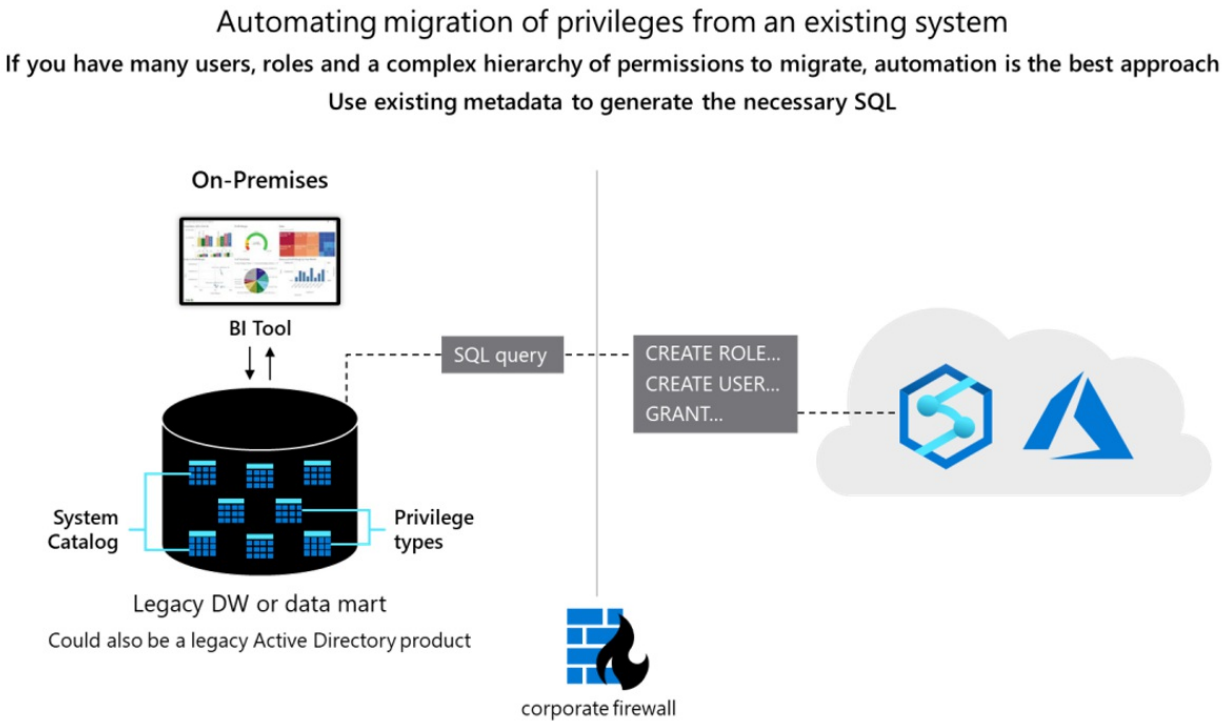
> **TIP**
> High-level planning is essential for a successful migration project.

Both Teradata and Azure Synapse implement database access control via a combination of users, roles, and permissions. Both use standard SQL `CREATE USER` and `CREATE ROLE` statements to define users and roles, and `GRANT` and `REVOKE` statements to assign or remove permissions to those users and/or roles.

> **TIP**
> Automation of migration processes is recommended to reduce elapsed time and scope for errors.

Conceptually the two databases are similar, and it might be possible to automate the migration of existing user IDs, roles, and permissions to some degree. Migrate such data by extracting the existing legacy user and role information from the Teradata system catalog tables and generating matching equivalent `CREATE USER` and `CREATE ROLE` statements to be run in Azure Synapse to recreate the same user/role hierarchy.

After data extraction, use Teradata system catalog tables to generate equivalent `GRANT` statements to assign permissions (where an equivalent one exists). The following diagram shows how to use existing metadata to generate the necessary SQL.



Users and roles

> **TIP**
> Migration of a data warehouse requires more than just tables, views, and SQL statements.

The information about current users and roles in a Teradata system is found in the system catalog tables `DBC.USERS` (or `DBC.DATABASES`) and `DBC.ROLEMEMBERS`. Query these tables (if the user has `SELECT` access to those tables) to obtain current lists of users and roles defined within the system. The following are examples of queries to do this for individual users:

```
/***SQL to find all users***/
SELECT
DatabaseName AS UserName
FROM DBC.Databases
WHERE dbkind = 'u';

/***SQL to find all roles***/
SELECT A.ROLENAME, A.GRANTEE, A.GRANTOR,
  A.DefaultRole,
  A.WithAdmin,
  B.DATABASENAME,
  B.TABLENAME,
  B.COLUMNNAME,
  B.GRANTORNAME,
  B.AccessRight
FROM DBC.ROLEMEMBERS A
JOIN DBC.ALLROLERIGHTS B
ON A.ROLENAME = B.ROLENAME
GROUP BY 1,2,3,4,5,6,7
ORDER BY 2,1,6;
```

These examples modify `SELECT` statements to produce a result set, which is a series of `CREATE USER` and `CREATE ROLE` statements, by including the appropriate text as a literal within the `SELECT` statement.

There's no way to retrieve existing passwords, so you need to implement a scheme for allocating new initial passwords on Azure Synapse.

Permissions

> **TIP**
> There are equivalent Azure Synapse permissions for basic database operations such as DML and DDL.

In a Teradata system, the system tables `DBC.ALLRIGHTS` and `DBC.ALLROLERIGHTS` hold the access rights for users and roles. Query these tables (if the user has `SELECT` access to those tables) to obtain current lists of access rights defined within the system. The following are examples of queries for individual users:

```
/**SQL for AccessRights held by a USER***/
SELECT UserName, DatabaseName,TableName,ColumnName,
CASE WHEN Abbv.AccessRight IS NOT NULL THEN Abbv.Description ELSE
ALRTS.AccessRight
END AS AccessRight, GrantAuthority, GrantorName, AllnessFlag, CreatorName, CreateTimeStamp
FROM DBC.ALLRIGHTS ALRTS LEFT OUTER JOIN AccessRightsAbbv Abbv
ON ALRTS.AccessRight = Abbv.AccessRight
WHERE UserName='UserXYZ'
Order By 2,3,4,5;

/**SQL for AccessRights held by a ROLE***/
SELECT RoleName, DatabaseName,TableName,ColumnName,
CASE WHEN Abbv.AccessRight IS NOT NULL THEN Abbv.Description ELSE
ALRTS.AccessRight
END AS AccessRight, GrantorName, CreateTimeStamp
FROM DBC.ALLROLERIGHTS ALRTS LEFT OUTER JOIN AccessRightsAbbv
Abbv
ON ALRTS.AccessRight = Abbv.AccessRight
WHERE RoleName='BI_DEVELOPER'
Order By 2,3,4,5;
```

Modify these example `SELECT` statements to produce a result set that's a series of `GRANT` statements by including the appropriate text as a literal within the `SELECT` statement.

Use the table `AccessRightsAbbv` to look up the full text of the access right, as the join key is an abbreviated 'type' field. See the following table for a list of Teradata access rights and their equivalent in Azure Synapse.

TERADATA PERMISSION NAME	TERADATA TYPE	AZURE SYNAPSE EQUIVALENT
ABORT SESSION	AS	KILL DATABASE CONNECTION
ALTER EXTERNAL PROCEDURE	AE	[4]
ALTER FUNCTION	AF	ALTER FUNCTION
ALTER PROCEDURE	AP	ALTER PROCEDURE
CHECKPOINT	CP	CHECKPOINT
CREATE AUTHORIZATION	CA	CREATE LOGIN
CREATE DATABASE	CD	CREATE DATABASE
CREATE EXTERNAL PROCEDURE	CE	[4]
CREATE FUNCTION	CF	CREATE FUNCTION
CREATE GLOP	GC	[3]
CREATE MACRO	CM	CREATE PROCEDURE [2]
CREATE OWNER PROCEDURE	OP	CREATE PROCEDURE
CREATE PROCEDURE	PC	CREATE PROCEDURE
CREATE PROFILE	CO	CREATE LOGIN [1]

TERADATA PERMISSION NAME	TERADATA TYPE	AZURE SYNAPSE EQUIVALENT
CREATE ROLE	CR	CREATE ROLE
DROP DATABASE	DD	DROP DATABASE
DROP FUNCTION	DF	DROP FUNCTION
DROP GLOP	GD	[3]
DROP MACRO	DM	DROP PROCEDURE [2]
DROP PROCEDURE	PD	DELETE PROCEDURE
DROP PROFILE	DO	DROP LOGIN [1]
DROP ROLE	DR	DELETE ROLE
DROP TABLE	DT	DROP TABLE
DROP TRIGGER	DG	[3]
DROP USER	DU	DROP USER
DROP VIEW	DV	DROP VIEW
DUMP	DP	[4]
EXECUTE	E	EXECUTE
EXECUTE FUNCTION	EF	EXECUTE
EXECUTE PROCEDURE	PE	EXECUTE
GLOP MEMBER	GM	[3]
INDEX	IX	CREATE INDEX
INSERT	I	INSERT
MONRESOURCE	MR	[5]
MONSESSION	MS	[5]
OVERRIDE DUMP CONSTRAINT	OA	[4]
OVERRIDE RESTORE CONSTRAINT	OR	[4]
REFERENCES	RF	REFERENCES
REPLCONTROL	RO	[5]

TERADATA PERMISSION NAME	TERADATA TYPE	AZURE SYNAPSE EQUIVALENT
RESTORE	RS	[4]
SELECT	R	SELECT
SETRESRATE	SR	[5]
SETSESSRATE	SS	[5]
SHOW	SH	[3]
UPDATE	U	UPDATE

`AccessRightsAbbv` table notes:

1. Teradata `PROFILE` is functionally equivalent to `LOGIN` in Azure Synapse.
2. The following table summarizes the differences between macros and stored procedures in Teradata. In Azure Synapse, procedures provide the functionality described in the table.

MACRO	STORED PROCEDURE
Contains SQL	Contains SQL
May contain BTEQ dot commands	Contains comprehensive SPL
May receive parameter values passed to it	May receive parameter values passed to it
May retrieve one or more rows	Must use a cursor to retrieve more than one row
Stored in DBC PERM space	Stored in DATABASE or USER PERM
Returns rows to the client	May return one or more values to client as parameters

3. `SHOW`, `GLOP`, and `TRIGGER` have no direct equivalent in Azure Synapse.
4. These features are managed automatically by the system in Azure Synapse. See Operational considerations.
5. In Azure Synapse, these features are handled outside of the database.

For more information about access rights in Azure Synapse, see to Azure Synapse Analytics security permissions.

Operational considerations

> **TIP**
> Operational tasks are necessary to keep any data warehouse operating efficiently.

This section discusses how to implement typical Teradata operational tasks in Azure Synapse with minimal risk and impact to users.

As with all data warehouse products, once in production there are ongoing management tasks that are necessary to keep the system running efficiently and to provide data for monitoring and auditing. Resource utilization and capacity planning for future growth also falls into this category, as does backup/restore of data.

While conceptually the management and operations tasks for different data warehouses are similar, the individual implementations may differ. In general, modern cloud-based products such as Azure Synapse tend to incorporate a more automated and "system managed" approach (as opposed to a more "manual" approach in legacy data warehouses such as Teradata).

The following sections compare Teradata and Azure Synapse options for various operational tasks.

Housekeeping tasks

> **TIP**
> Housekeeping tasks keep a production warehouse operating efficiently and optimize use of resources such as storage.

In most legacy data warehouse environments, there's a requirement to perform regular "housekeeping" tasks such as reclaiming disk storage space that can be freed up by removing old versions of updated or deleted rows, or reorganizing data log files or index blocks for efficiency. Collecting statistics is also a potentially time-consuming task. Collecting statistics is required after a bulk data ingest to provide the query optimizer with up-to-date data to base generation of query execution plans.

Teradata recommends collecting statistics as follows:

- Collect statistics on unpopulated tables to set up the interval histogram used in internal processing. This initial collection makes subsequent statistics collections faster. Make sure to recollect statistics after data is added.
- Collect prototype phase statistics for newly populated tables.
- Collect production phase statistics after a significant percentage of change to the table or partition (~10% of rows). For high volumes of nonunique values, such as dates or timestamps, it may be advantageous to recollect at 7%.
- Collect production phase statistics after you've created users and applied real world query loads to the database (up to about three months of querying).
- Collect statistics in the first few weeks after an upgrade or migration during periods of low CPU utilization.

Statistics collection can be managed manually using Automated Statistics Management open APIs or automatically using the Teradata Viewpoint Stats Manager portlet.

> **TIP**
> Automate and monitor housekeeping tasks in Azure.

Teradata Database contains many log tables in the Data Dictionary that accumulate data, either automatically or after certain features are enabled. Because log data grows over time, purge older information to avoid using up permanent space. There are options to automate the maintenance of these logs available. The Teradata dictionary tables that require maintenance are discussed next.

Dictionary tables to maintain

Reset accumulators and peak values using the `DBC.AMPUsage` view and the `ClearPeakDisk` macro provided with the software:

- `DBC.Acctg`: resource usage by account/user
- `DBC.DataBaseSpace`: database and table space accounting

Teradata automatically maintains these tables, but good practices can reduce their size:

- `DBC.AccessRights`: user rights on objects
- `DBC.RoleGrants`: role rights on objects
- `DBC.Roles`: defined roles
- `DBC.Accounts`: account codes by user

Archive these logging tables (if desired) and purge information 60-90 days old. Retention depends on customer requirements:

- `DBC.SW_Event_Log`: database console log
- `DBC.ResUsage`: resource monitoring tables
- `DBC.EventLog`: session logon/logoff history
- `DBC.AccLogTbl`: logged user/object events
- `DBC.DBQL tables`: logged user/SQL activity
- `.NETSecPolicyLogTbl`: logs dynamic security policy audit trails
- `.NETSecPolicyLogRuleTbl`: controls when and how dynamic security policy is logged

Purge these tables when the associated removable media is expired and overwritten:

- `DBC.RCEvent`: archive/recovery events
- `DBC.RCConfiguration`: archive/recovery config
- `DBC.RCMedia`: VolSerial for archive/recovery

Azure Synapse has an option to automatically create statistics so that they can be used as needed. Perform defragmentation of indexes and data blocks manually, on a scheduled basis, or automatically. Leveraging native built-in Azure capabilities can reduce the effort required in a migration exercise.

Monitoring and auditing

> **TIP**
> Over time, several different tools have been implemented to allow monitoring and logging of Teradata systems.

Teradata provides several tools to monitor the operation including Teradata Viewpoint and Ecosystem Manager. For logging query history, the Database Query Log (DBQL) is a Teradata database feature that provides a series of predefined tables that can store historical records of queries and their duration, performance, and target activity based on user-defined rules.

Database administrators can use Teradata Viewpoint to determine system status, trends, and individual query status. By observing trends in system usage, system administrators are better able to plan project implementations, batch jobs, and maintenance to avoid peak periods of use. Business users can use Teradata Viewpoint to quickly access the status of reports and queries and drill down into details.

> **TIP**
> The Azure portal provides a UI to manage monitoring and auditing tasks for all Azure data and processes.

Similarly, Azure Synapse provides a rich monitoring experience within the Azure portal to provide insights into your data warehouse workload. The Azure portal is the recommended tool when monitoring your data warehouse as it provides configurable retention periods, alerts, recommendations, and customizable charts and dashboards for metrics and logs.

The portal also enables integration with other Azure monitoring services such as Operations Management Suite (OMS) and Azure Monitor (logs) to provide a holistic monitoring experience for not only the data warehouse but also the entire Azure analytics platform for an integrated monitoring experience.

> **TIP**
> Low-level and system-wide metrics are automatically logged in Azure Synapse.

Resource utilization statistics for Azure Synapse are automatically logged within the system. The metrics for each query include usage statistics for CPU, memory, cache, I/O, and temporary workspace, as well as connectivity information like failed connection attempts.

Azure Synapse provides a set of Dynamic Management Views (DMVs). These views are useful when actively troubleshooting and identifying performance bottlenecks with your workload.

For more information, see Azure Synapse operations and management options.

High Availability (HA) and Disaster Recovery (DR)

Teradata implements features such as `FALLBACK`, Archive Restore Copy utility (ARC), and Data Stream Architecture (DSA) to provide protection against data loss and high availability (HA) via replication and archive of data. Disaster Recovery (DR) options include Dual Active Solution, DR as a service, or a replacement system depending on the recovery time requirement.

> **TIP**
> Azure Synapse creates snapshots automatically to ensure fast recovery times.

Azure Synapse uses database snapshots to provide high availability of the warehouse. A data warehouse snapshot creates a restore point that can be used to recover or copy a data warehouse to a previous state. Since Azure Synapse is a distributed system, a data warehouse snapshot consists of many files that are in Azure Storage. Snapshots capture incremental changes from the data stored in your data warehouse.

Azure Synapse automatically takes snapshots throughout the day creating restore points that are available for seven days. This retention period can't be changed. Azure Synapse supports an eight-hour recovery point objective (RPO). A data warehouse can be restored in the primary region from any one of the snapshots taken in the past seven days.

> **TIP**
> Use user-defined snapshots to define a recovery point before key updates.

User-defined restore points are also supported, allowing manual triggering of snapshots to create restore points of a data warehouse before and after large modifications. This capability ensures that restore points are logically consistent, which provides additional data protection in case of any workload interruptions or user errors for a

desired RPO of less than 8 hours.

> **TIP**
> Microsoft Azure provides automatic backups to a separate geographical location to enable DR.

As well as the snapshots described previously, Azure Synapse also performs as standard a geo-backup once per day to a paired data center. The RPO for a geo-restore is 24 hours. You can restore the geo-backup to a server in any other region where Azure Synapse is supported. A geo-backup ensures that a data warehouse can be restored in case the restore points in the primary region aren't available.

Workload management

> **TIP**
> In a production data warehouse, there are typically mixed workloads with different resource usage characteristics running concurrently.

A workload is a class of database requests with common traits whose access to the database can be managed with a set of rules. Workloads are useful for:

- Setting different access priorities for different types of requests.
- Monitoring resource usage patterns, performance tuning, and capacity planning.
- Limiting the number of requests or sessions that can run at the same time.

In a Teradata system, workload management is the act of managing workload performance by monitoring system activity and acting when pre-defined limits are reached. Workload management uses rules, and each rule applies only to some database requests. However, the collection of all rules applies to all active work on the platform. Teradata Active System Management (TASM) performs full workload management in a Teradata Database.

In Azure Synapse, resource classes are pre-determined resource limits that govern compute resources and concurrency for query execution. Resource classes can help you manage your workload by setting limits on the number of queries that run concurrently and on the compute resources assigned to each query. There's a trade-off between memory and concurrency.

Azure Synapse automatically logs resource utilization statistics. Metrics include usage statistics for CPU, memory, cache, I/O, and temporary workspace for each query. Azure Synapse also logs connectivity information, such as failed connection attempts.

> **TIP**
> Low-level and system-wide metrics are automatically logged within Azure.

Azure Synapse supports these basic workload management concepts:

- **Workload classification**: you can assign a request to a workload group to set importance levels.
- **Workload importance**: you can influence the order in which a request gets access to resources. By default, queries are released from the queue on a first-in, first-out basis as resources become available. Workload importance allows higher priority queries to receive resources immediately regardless of the queue.
- **Workload isolation**: you can reserve resources for a workload group, assign maximum and minimum

usage for varying resources, limit the resources a group of requests can consume can, and set a timeout value to automatically kill runaway queries.

Running mixed workloads can pose resource challenges on busy systems. A successful workload management scheme effectively manages resources, ensures highly efficient resource utilization, and maximizes return on investment (ROI). The workload classification, workload importance, and workload isolation gives more control over how workload utilizes system resources.

The workload management guide describes the techniques to analyze the workload, manage and monitor workload importance](../../sql-data-warehouse/sql-data-warehouse-how-to-manage-and-monitor-workload-importance.md), and the steps to convert a resource class to a workload group. Use the Azure portal and T-SQL queries on DMVs to monitor the workload to ensure that the applicable resources are efficiently utilized. Azure Synapse provides a set of Dynamic Management Views (DMVs) for monitoring all aspects of workload management. These views are useful when actively troubleshooting and identifying performance bottlenecks in your workload.

This information can also be used for capacity planning, determining the resources required for additional users or application workload. This also applies to planning scale up/scale downs of compute resources for cost-effective support of "peaky" workloads.

For more information on workload management in Azure Synapse, see Workload management with resource classes.

Scale compute resources

> **TIP**
> A major benefit of Azure is the ability to independently scale up and down compute resources on demand to handle peaky workloads cost-effectively.

The architecture of Azure Synapse separates storage and compute, allowing each to scale independently. As a result, compute resources can be scaled to meet performance demands independent of data storage. You can also pause and resume compute resources. A natural benefit of this architecture is that billing for compute and storage is separate. If a data warehouse isn't in use, you can save on compute costs by pausing compute.

Compute resources can be scaled up or scaled back by adjusting the data warehouse units setting for the data warehouse. Loading and query performance will increase linearly as you add more data warehouse units.

Adding more compute nodes adds more compute power and ability to leverage more parallel processing. As the number of compute nodes increases, the number of distributions per compute node decreases, providing more compute power and parallel processing for queries. Similarly, decreasing data warehouse units reduces the number of compute nodes, which reduces the compute resources for queries.

Next steps

To learn more about visualization and reporting, see the next article in this series: Visualization and reporting for Teradata migrations.

Visualization and reporting for Teradata migrations

8/19/2022 • 20 minutes to read • Edit Online

This article is part four of a seven-part series that provides guidance on how to migrate from Teradata to Azure Synapse Analytics. The focus of this article is best practices for visualization and reporting.

Access Azure Synapse Analytics using Microsoft and third-party BI tools

Organizations access data warehouses and data marts using a range of business intelligence (BI) tools and applications. Some examples of BI products are:

- Microsoft BI tools, such as Power BI.
- Office applications, such as Microsoft Excel spreadsheets.
- Third-party BI tools from different vendors.
- Custom analytics applications with embedded BI tool functionality.
- Operational applications that support on-demand BI by running queries and reports on a BI platform that in turn queries data in a data warehouse or data mart.
- Interactive data science development tools, such as Azure Synapse Spark Notebooks, Azure Machine Learning, RStudio, and Jupyter Notebooks.

If you migrate visualization and reporting as part of your data warehouse migration, all existing queries, reports, and dashboards generated by BI products need to run in the new environment. Your BI products must yield the same results on Azure Synapse as they did in your legacy data warehouse environment.

For consistent results after migration, all BI tools and application dependencies must work after you've migrated your data warehouse schema and data to Azure Synapse. The dependencies include less visible aspects, such as access and security. When you address access and security, ensure that you migrate:

- Authentication so users can sign into the data warehouse and data mart databases on Azure Synapse.
- All users to Azure Synapse.
- All user groups to Azure Synapse.
- All roles to Azure Synapse.
- All authorization privileges governing access control to Azure Synapse.
- User, role, and privilege assignments to mirror what you had in your existing data warehouse before migration. For example:
    - Database object privileges assigned to roles
    - Roles assigned to user groups
    - Users assigned to user groups and/or roles

Access and security are important considerations for data access in the migrated system and are discussed in more detail in Security, access, and operations for Teradata migrations.

TIP

Existing users, user groups, roles, and assignments of access security privileges need to be migrated first for migration of reports and visualizations to succeed.

Migrate all required data to ensure that the reports and dashboards that query data in the legacy environment produce the same results in Azure Synapse.

Business users will expect a seamless migration, with no surprises that destroy their confidence in the migrated system on Azure Synapse. Take care to allay any fears that your users might have through good communication. Your users will expect that:

- Table structure remains the same when directly referred to in queries.
- Table and column names remain the same when directly referred to in queries. For instance, calculated fields defined on columns in BI tools shouldn't fail when aggregate reports are produced.
- Historical analysis remains the same.
- Data types remain the same, if possible.
- Query behavior remains the same.
- ODBC/JDBC drivers are tested to ensure that query behavior remains the same.

TIP

Communication and business user involvement are critical to success.

If BI tools query views in the underlying data warehouse or data mart database, will those views still work after the migration? Some views might not work if there are proprietary SQL extensions specific to your legacy data warehouse DBMS that have no equivalent in Azure Synapse. If so, you need to know about those incompatibilities and find a way to resolve them.

TIP

Views and SQL queries using proprietary SQL query extensions are likely to result in incompatibilities that impact BI reports and dashboards.

Other issues, like the behavior of `NULL` values or data type variations across DBMS platforms, need to be tested to ensure that even slight differences don't exist in calculation results. Minimize those issues and take all necessary steps to shield business users from being affected by them. Depending on your legacy data warehouse environment, third-party tools that can help hide the differences between the legacy and new environments so that BI tools and applications run unchanged.

Testing is critical to visualization and report migration. You need a test suite and agreed-on test data to run and rerun tests in both environments. A test harness is also useful, and a few are mentioned in this guide. Also, it's important to involve business users in the testing aspect of the migration to keep confidence high and to keep them engaged and part of the project.

TIP

Use repeatable tests to ensure reports, dashboards, and other visualizations migrate successfully.

You might be thinking about switching BI tools, for example to migrate to Power BI. The temptation is to make such changes at the same time you're migrating your schema, data, ETL processing, and more. However, to minimize risk, it's better to migrate to Azure Synapse first and get everything working before undertaking further modernization.

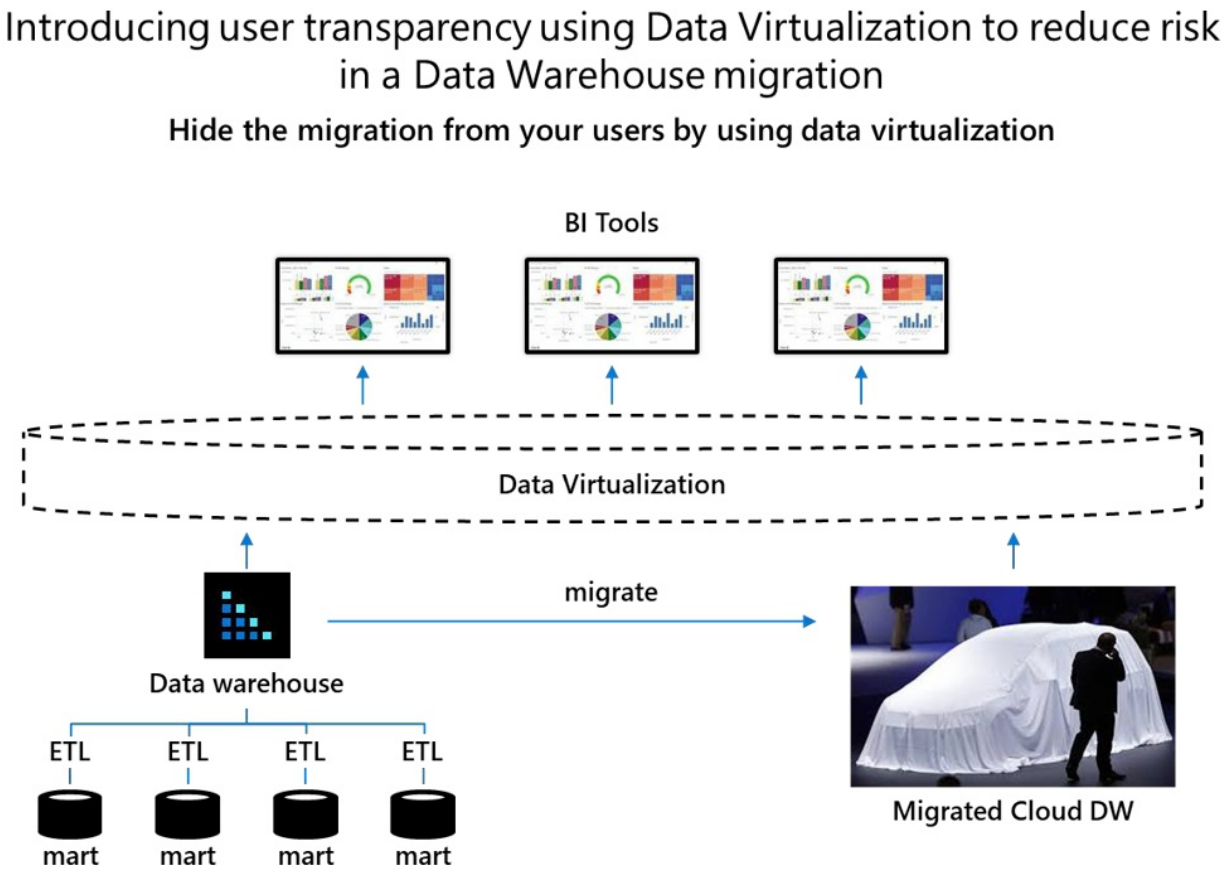
If your existing BI tools run on-premises, ensure they can connect to Azure Synapse through your firewall so you're able to run comparisons against both environments. Alternatively, if the vendor of your existing BI tools offers their product on Azure, you can try it there. The same applies for applications running on-premises that embed BI or call your BI server on demand, for example by requesting a "headless report" with XML or JSON data.

There's a lot to think about here, so let's take a closer look.

Use data virtualization to minimize the impact of migration on BI tools and reports

During migration, you might be tempted to fulfill long-term requirements like opening business requests, adding missing data, or implementing new features. However, such changes can affect BI tool access to your data warehouse, especially if the change involves structural changes to your data model. If you want to adopt an agile data modeling technique or implement structural changes, do so *after* migration.

One way to minimize the effect of schema changes or other structural changes on your BI tools is to introduce data virtualization between the BI tools and your data warehouse and data marts. The following diagram shows how data virtualization can hide a migration from users.



Data virtualization breaks the dependency between business users utilizing self-service BI tools and the physical schema of the underlying data warehouse and data marts that are being migrated.

> **TIP**
> Data virtualization allows you to shield business users from structural changes during migration so they remain unaware of those changes. Structural changes include schema alterations that tune your data model for Azure Synapse.

With data virtualization, any schema alterations made during a migration to Azure Synapse, for example to optimize performance, can be hidden from business users because they only have access to virtual tables in the data virtualization layer. And, if you make structural changes, you only need to update the mappings between the data warehouse or data marts and any virtual tables. With data virtualization, users remain unaware of structural changes. Microsoft partners provide data virtualization software.

Identify high-priority reports to migrate first

A key question when migrating your existing reports and dashboards to Azure Synapse is which ones to migrate first. Several factors might drive that decision, such as:

- Usage
- Business value
- Ease of migration
- Data migration strategy

The following sections discuss these factors.

Whatever your decision, it must involve your business users because they produce the reports, dashboards, and other visualizations, and make business decisions based on insights from those items. Everyone benefits when you can:

- Migrate reports and dashboards seamlessly,
- Migrate reports and dashboards with minimal effort, and
- Point your BI tool(s) at Azure Synapse instead of your legacy data warehouse system, and get like-for-like reports, dashboards, and other visualizations.

Migrate reports based on usage

Usage is often an indicator of business value. Unused reports and dashboards clearly don't contribute to business decisions or offer current value. If you don't have a way to find out which reports and dashboards are unused, you can use one of the several BI tools that provide usage statistics.

If your legacy data warehouse has been up and running for years, there's a good chance you have hundreds, if not thousands, of reports in existence. It's worth compiling an inventory of reports and dashboards and identifying their business purpose and usage statistics.

For unused reports, determine whether to decommission them to reduce your migration effort. A key question when deciding whether to decommission an unused report is whether the report is unused because people don't know it exists, because it offers no business value, or because it's been superseded by another report.

Migrate reports based on business value

Usage alone isn't always a good indicator of business value. You might want to consider the extent to which a report's insights contribute to business value. One way to do that is to evaluate the profitability of every business decision that relies on the report and the extent of the reliance. However, that information is unlikely to be readily available in most organizations.

Another way to evaluate business value is to look at the alignment of a report with business strategy. The business strategy set by your executive typically lays out strategic business objectives (SBOs), key performance

indicators (KPIs), KPI targets that need to be achieved, and who is accountable for achieving them. You can classify a report by which SBOs the report contributes to, such as fraud reduction, improved customer engagement, and optimized business operations. Then, you can prioritize for migration the reports and dashboards that are associated with high-priority objectives. In this way, the initial migration can deliver business value in a strategic area.

Another way to evaluate business value is to classify reports and dashboards as operational, tactical, or strategic to identify at which business level they're used. SBOs require contributions at all these levels. By knowing which reports and dashboards are used, at what level, and what objectives they're associated with, you're able to focus the initial migration on high-priority business value. You can use the following *business strategy objective* table to evaluate reports and dashboards.

LEVEL	REPORT / DASHBOARD NAME	BUSINESS PURPOSE	DEPARTMENT USED	USAGE FREQUENCY	BUSINESS PRIORITY
Strategic					
Tactical					
Operational					

Metadata discovery tools like Azure Data Catalog let business users tag and rate data sources to enrich the metadata for those data sources to assist with their discovery and classification. You can use the metadata for a report or dashboard to help you understand its business value. Without such tools, understanding the contribution of reports and dashboards to business value is likely to be a time-consuming task, whether you're migrating or not.

Migrate reports based on data migration strategy

If your migration strategy is based on migrating data marts first, then the order of data mart migration will affect which reports and dashboards are migrated first. If your strategy is based on business value, the order in which you migrate data marts to Azure Synapse will reflect business priorities. Metadata discovery tools can help you implement your strategy by showing you which data mart tables supply data for which reports.

> **TIP**
> Your data migration strategy affects which reports and visualizations get migrated first.

Migration incompatibility issues that can affect reports and visualizations

BI tools produce reports, dashboards, and other visualizations by issuing SQL queries that access physical tables and/or views in your data warehouse or data mart. When you migrate your legacy data warehouse to Azure Synapse, several factors can affect the ease of migration of reports, dashboards, and other visualizations. Those factors include:

- Schema incompatibilities between the environments.
- SQL incompatibilities between the environments.

Schema incompatibilities

During a migration, schema incompatibilities in the data warehouse or data mart tables that supply data for reports, dashboards, and other visualizations can be:

- Non-standard table types in your legacy data warehouse DBMS that don't have an equivalent in Azure

Synapse.

- Data types in your legacy data warehouse DBMS that don't have an equivalent in Azure Synapse.

In most cases, there's a workaround to the incompatibilities. For example, you can migrate the data in an unsupported table type into a standard table with appropriate data types and indexed or partitioned on a date/time column. Similarly, it might be possible to represent unsupported data types in another type of column and perform calculations in Azure Synapse to achieve the same results.

> **TIP**
> Schema incompatibilities include legacy warehouse DBMS table types and data types that are unsupported on Azure Synapse.

To identify the reports affected by schema incompatibilities, run queries against the system catalog of your legacy data warehouse to identify the tables with unsupported data types. Then, you can use metadata from your BI tool to identify the reports that access data in those tables. For more information about how to identify object type incompatibilities, see Unsupported Teradata database object types.

> **TIP**
> Query the system catalog of your legacy warehouse DBMS to identify schema incompatibilities with Azure Synapse.

The effect of schema incompatibilities on reports, dashboards, and other visualizations might be less than you think because many BI tools don't support the less generic data types. As a result, your legacy data warehouse might already have views that `CAST` unsupported data types to more generic types.

SQL incompatibilities

During a migration, SQL incompatibilities are likely to affect any report, dashboard, or other visualization in an application or tool that:

- Accesses legacy data warehouse DBMS views that include proprietary SQL functions that have no equivalent in Azure Synapse.
- Issues SQL queries that include proprietary SQL functions, specific to the SQL dialect of your legacy environment, that have no equivalent in Azure Synapse.

Gauge the impact of SQL incompatibilities on your reporting portfolio

Your reporting portfolio might include embedded query services, reports, dashboards, and other visualizations. Don't rely on the documentation associated with those items to gauge the effect of SQL incompatibilities on the migration of your reporting portfolio to Azure Synapse. You need to use a more precise way to assess the effect of SQL incompatibilities.

Use EXPLAIN statements to find SQL incompatibilities

You can find SQL incompatibilities by reviewing the logs of recent SQL activity in your legacy Teradata data warehouse. Use a script to extract a representative set of SQL statements to a file. Then, prefix each SQL statement with an `EXPLAIN` statement, and run those `EXPLAIN` statements in Azure Synapse. Any SQL statements containing proprietary unsupported SQL extensions will be rejected by Azure Synapse when the `EXPLAIN` statements are executed. This approach lets you assess the extent of SQL incompatibilities.

Metadata from your legacy data warehouse DBMS can also help you identify incompatible views. As before, capture a representative set of SQL statements from the applicable logs, prefix each SQL statement with an `EXPLAIN` statement, and run those `EXPLAIN` statements in Azure Synapse to identify views with incompatible SQL.

TIP

Gauge the impact of SQL incompatibilities by harvesting your DBMS log files and running `EXPLAIN` statements.

Test report and dashboard migration to Azure Synapse Analytics

A key element of data warehouse migration is testing of reports and dashboards in Azure Synapse to verify the migration has worked. Define a series of tests and a set of required outcomes for each test that you will run to verify success. Test and compare the reports and dashboards across your existing and migrated data warehouse systems to:

- Identify whether any schema changes that were made during migration affected the ability of reports to run, report results, or the corresponding report visualizations. An example of a schema change is if you mapped an incompatible data type to an equivalent data type that's supported in Azure Synapse.
- Verify that all users are migrated.
- Verify that all roles are migrated, and users are assigned to those roles.
- Verify that all data access security privileges are migrated to ensure access control list (ACL) migration.
- Ensure consistent results for all known queries, reports, and dashboards.
- Ensure that data and ETL migration is complete and error-free.
- Ensure that data privacy is upheld.
- Test performance and scalability.
- Test analytical functionality.

TIP

Test and tune performance to minimize compute costs.

For information about how to migrate users, user groups, roles, and privileges, see Security, access, and operations for Teradata migrations.

Automate testing as much as possible to make each test repeatable and to support a consistent approach to evaluating test results. Automation works well for known regular reports, and can be managed via Azure Synapse pipelines or Azure Data Factory orchestration. If you already have a suite of test queries in place for regression testing, you can use the existing testing tools to automate post migration testing.

TIP

Best practice is to build an automated test suite to make tests repeatable.

Ad-hoc analysis and reporting are more challenging and require compilation of a set of tests to verify that the same reports and dashboards from before and after migration are consistent. If you find inconsistencies, then your ability to compare metadata lineage across the original and migrated systems during migration testing becomes crucial. That comparison can highlight differences and pinpoint where inconsistencies originated, when detection by other means is difficult.

> **TIP**
> Leverage tools that compare metadata lineage to verify results.

Analyze lineage to understand dependencies between reports, dashboards, and data

Your understanding of lineage is a critical factor in the successful migration of reports and dashboards. Lineage is metadata that shows the journey of migrated data so you can track its path from a report or dashboard all the way back to the data source. Lineage shows how data has traveled from point to point, its location in the data warehouse and/or data mart, and which reports and dashboards use it. Lineage can help you understand what happens to data as it travels through different data stores, such as files and databases, different ETL pipelines, and into reports. When business users have access to data lineage, it improves trust, instills confidence, and supports informed business decisions.

> **TIP**
> Your ability to access metadata and data lineage from reports all the way back to a data source is critical for verifying that migrated reports work correctly.

In multi-vendor data warehouse environments, business analysts in BI teams might map out data lineage. For example, if you use different vendors for ETL, data warehouse, and reporting, and each vendor has its own metadata repository, then figuring out where a specific data element in a report came from can be challenging and time-consuming.

> **TIP**
> Tools that automate the collection of metadata and show end-to-end lineage in a multi-vendor environment are valuable during a migration.

To migrate seamlessly from a legacy data warehouse to Azure Synapse, use end-to-end data lineage to prove like-for-like migration when you're comparing the reports and dashboards generated by each environment. To show the end-to-end data journey, you'll need to capture and integrate metadata from several tools. Having access to tools that support automated metadata discovery and data lineage, helps you identify duplicate reports or ETL processes, and find reports that rely on obsolete, questionable, or non-existent data sources. You can use that information to reduce the number of reports and ETL processes that you migrate.

You can also compare the end-to-end lineage of a report in Azure Synapse to the end-to-end lineage of the same report in your legacy environment to check for differences that might have inadvertently occurred during migration. This type of comparison is exceptionally useful when you need to test and verify migration success.

Data lineage visualization not only reduces time, effort, and error in the migration process, but also enables faster migration.

By using automated metadata discovery and data lineage tools that compare lineage, you can verify that a report in Azure Synapse that's produced from migrated data is produced in the same way in your legacy environment. This capability also helps you determine:

- What data needs to be migrated to ensure successful report and dashboard execution in Azure Synapse.
- What transformations have been and should be performed to ensure successful execution in Azure Synapse.

- How to reduce report duplication.

Automated metadata discovery and data lineage tools substantially simplify the migration process because they help businesses become more aware of their data assets and to know what needs to be migrated to Azure Synapse to achieve a solid reporting environment.

Several ETL tools provide end-to-end lineage capability, so check whether your existing ETL tool has that capability if you plan to use it with Azure Synapse. Azure Synapse pipelines or Data Factory both support the ability to view lineage in mapping flows. Microsoft partners also provide automated metadata discovery, data lineage, and lineage comparison tools.

Migrate BI tool semantic layers to Azure Synapse Analytics

Some BI tools have what is known as a semantic metadata layer. That layer simplifies business user access to the underlying physical data structures in a data warehouse or data mart database. The semantic metadata layer simplifies access by providing high-level objects like dimensions, measures, hierarchies, calculated metrics, and joins. The high-level objects use business terms that are familiar to business analysts, and map to physical data structures in your data warehouse or data mart.

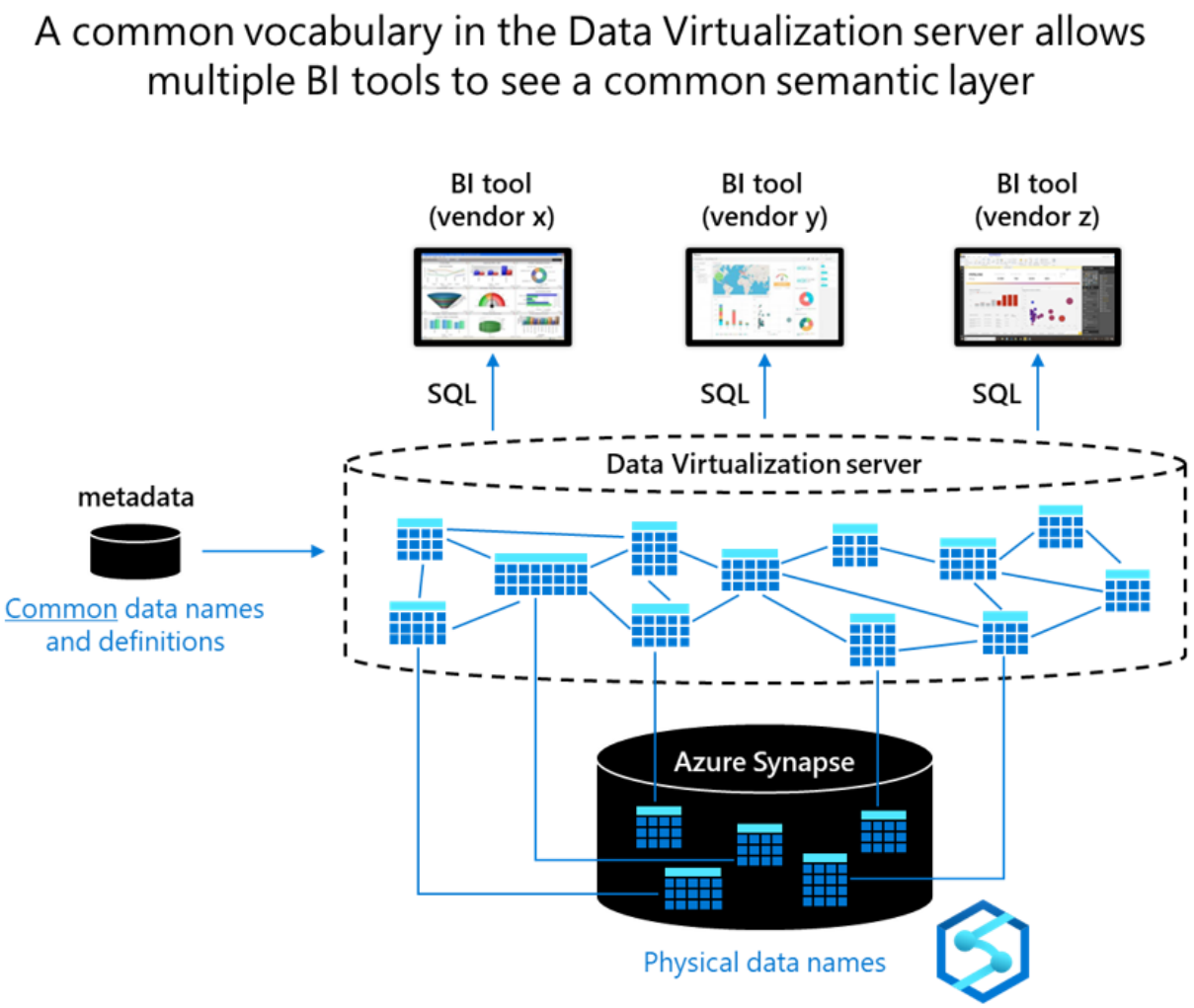
> **TIP**
> Some BI tools have semantic layers that simplify business user access to physical data structures in your data warehouse or data mart.

In a data warehouse migration, changes to column names or table names may be forced upon you. For example, in Teradata, table names can have a "#". In Azure Synapse, the "#" is only allowed as a prefix to a table name to indicate a temporary table. In Teradata, TEMPORARY TABLES do not necessarily have a "#" in the name, but in Synapse they must. You may need to do some rework to change table mappings in such cases.

To achieve consistency across multiple BI tools, create a universal semantic layer by using a data virtualization server that sits between BI tools and applications and Azure Synapse. In the data virtualization server, use common data names for high-level objects like dimensions, measures, hierarchies, and joins. That way you configure everything, including calculated fields, joins, and mappings, only once instead of in every tool. Then, point all BI tools at the data virtualization server.

> **TIP**
> Use data virtualization to create a common semantic layer to guarantee consistency across all BI tools in an Azure Synapse environment.

With data virtualization, you get consistency across all BI tools and break the dependency between BI tools and applications and the underlying physical data structures in Azure Synapse. Microsoft partners can help you achieve consistency in Azure. The following diagram shows how a common vocabulary in the data virtualization server lets multiple BI tools see a common semantic layer.



Conclusions

In a lift and shift data warehouse migration, most reports, dashboards, and other visualizations should migrate easily.

During a migration from a legacy environment, you might find that data in the legacy data warehouse or data mart tables is stored in unsupported data types. Or, you may find legacy data warehouse views that include proprietary SQL with no equivalent in Azure Synapse. If so, you'll need to resolve those issues to ensure a successful migration to Azure Synapse.

Don't rely on user-maintained documentation to identify where issues are located. Instead, use `EXPLAIN` statements because they're a quick, pragmatic way to identify SQL incompatibilities. Rework the incompatible SQL statements to achieve equivalent functionality in Azure Synapse. Also, use automated metadata discovery and lineage tools to understand dependencies, find duplicate reports, and identify invalid reports that rely on obsolete, questionable, or non-existent data sources. Use lineage tools to compare lineage to verify that reports running in your legacy data warehouse environment are produced identically in Azure Synapse.

Don't migrate reports that you no longer use. BI tool usage data can help you determine which reports aren't in use. For the reports, dashboards, and other visualizations that you do want to migrate, migrate all users, user groups, roles, and privileges. If you're using business value to drive your report migration strategy, associate reports with strategic business objectives and priorities to help identify the contribution of report insights to specific objectives. If you're migrating data mart by data mart, use metadata to identify which reports are dependent on which tables and views, so you can make an informed decision about which data marts to migrate first.

> **TIP**
> Identify incompatibilities early to gauge the extent of the migration effort. Migrate your users, group roles, and privilege assignments. Only migrate the reports and visualizations that are used and are contributing to business value.

Structural changes to the data model of your data warehouse or data mart can occur during a migration. Consider using data virtualization to shield BI tools and applications from structural changes. With data virtualization, you can use a common vocabulary to define a common semantic layer. The common semantic layer guarantees consistent common data names, definitions, metrics, hierarchies, and joins across all BI tools and applications in the new Azure Synapse environment.

Next steps

To learn more about minimizing SQL issues, see the next article in this series: Minimizing SQL issues for Teradata migrations.

Minimize SQL issues for Teradata migrations

8/19/2022 • 17 minutes to read • Edit Online

This article is part five of a seven-part series that provides guidance on how to migrate from Teradata to Azure Synapse Analytics. The focus of this article is best practices for minimizing SQL issues.

Overview

Characteristics of Teradata environments

> **TIP**
> Teradata pioneered large scale SQL databases using MPP in the 1980s.

In 1984, Teradata initially released their database product. It introduced massively parallel processing (MPP) techniques to enable data processing at a scale more efficiently than the existing mainframe technologies available at the time. Since then, the product has evolved and has many installations among large financial institutions, telecommunications, and retail companies. The original implementation used proprietary hardware and was channel attached to mainframes—typically IBM or IBM-compatible processors.

While more recent announcements have included network connectivity and the availability of the Teradata technology stack in the cloud (including Azure), most existing installations are on premises, so many users are considering migrating some or all their Teradata data to Azure Synapse Analytics to gain the benefits of a move to a modern cloud environment.

> **TIP**
> Many existing Teradata installations are data warehouses using a dimensional data model.

Teradata technology is often used to implement a data warehouse, supporting complex analytic queries on large data volumes using SQL. Dimensional data models—star or snowflake schemas—are common, as is the implementation of data marts for individual departments.

This combination of SQL and dimensional data models simplifies migration to Azure Synapse, since the basic concepts and SQL skills are transferable. The recommended approach is to migrate the existing data model as-is to reduce risk and time taken. Even if the eventual intention is to make changes to the data model (for example, moving to a data vault model), perform an initial as-is migration and then make changes within the Azure cloud environment, leveraging the performance, elastic scalability, and cost advantages there.

While the SQL language has been standardized, individual vendors have in some cases implemented proprietary extensions. This document highlights potential SQL differences you may encounter while migrating from a legacy Teradata environment, and provides workarounds.

Use an Azure VM Teradata instance as part of a migration

> **TIP**
> Use an Azure VM to create a temporary Teradata instance to speed up migration and minimize impact on the source system.

Leverage the Azure environment when running a migration from an on-premises Teradata environment. Azure

provides affordable cloud storage and elastic scalability to create a Teradata instance within a VM in Azure, collocated with the target Azure Synapse environment.

With this approach, standard Teradata utilities such as Teradata Parallel Data Transporter (or third-party data replication tools such as Attunity Replicate) can be used to efficiently move the subset of Teradata tables that are to be migrated onto the VM instance, and then all migration tasks can take place within the Azure environment. This approach has several benefits:

- After the initial replication of data, the source system isn't impacted by the migration tasks.
- The familiar Teradata interfaces, tools, and utilities are available within the Azure environment.
- Once in the Azure environment there are no potential issues with network bandwidth availability between the on-premises source system and the cloud target system.
- Tools such as Azure Data Factory can efficiently call utilities such as Teradata Parallel Transporter to migrate data quickly and easily.
- The migration process is orchestrated and controlled entirely within the Azure environment.

Use Azure Data Factory to implement a metadata-driven migration

> **TIP**
> Automate the migration process by using Azure Data Factory capabilities.

Automate and orchestrate the migration process by making use of the capabilities in the Azure environment. This approach also minimizes the migration's impact on the existing Teradata environment, which may already be running close to full capacity.

Azure Data Factory is a cloud-based data integration service that allows creation of data-driven workflows in the cloud for orchestrating and automating data movement and data transformation. Using Data Factory, you can create and schedule data-driven workflows—called pipelines—that can ingest data from disparate data stores. It can process and transform data by using compute services such as Azure HDInsight Hadoop, Spark, Azure Data Lake Analytics, and Azure Machine Learning.

By creating metadata to list the data tables to be migrated and their location, you can use the Data Factory facilities to manage and automate parts of the migration process. You can also use Azure Synapse Pipelines.

SQL DDL differences between Teradata and Azure Synapse

SQL Data Definition Language (DDL)

> **TIP**
> SQL DDL commands `CREATE TABLE` and `CREATE VIEW` have standard core elements but are also used to define implementation-specific options.

The ANSI SQL standard defines the basic syntax for DDL commands such as `CREATE TABLE` and `CREATE VIEW`. These commands are used within both Teradata and Azure Synapse, but they've also been extended to allow definition of implementation-specific features such as indexing, table distribution, and partitioning options.

The following sections discuss Teradata-specific options to consider during a migration to Azure Synapse.

Table considerations

> **TIP**
> Use existing indexes to give an indication of candidates for indexing in the migrated warehouse.

When migrating tables between different technologies, only the raw data and its descriptive metadata get physically moved between the two environments. Other database elements from the source system, such as indexes and log files, aren't directly migrated as these may not be needed or may be implemented differently within the new target environment. For example, there's no equivalent of the `MULTISET` option within Teradata's `CREATE TABLE` syntax.

It's important to understand where performance optimizations—such as indexes—were used in the source environment. This indicates where performance optimization can be added in the new target environment. For example, if a non-unique secondary index (NUSI) has been created in the source Teradata environment, this might indicate that a non-clustered index should be created in the migrated Azure Synapse database. Other native performance optimization techniques, such as table replication, may be more applicable than a straight "like-for-like" index creation.

Unsupported Teradata table types

> **TIP**
> Standard tables within Azure Synapse can support migrated Teradata time-series and temporal tables.

Teradata includes support for special table types for time-series and temporal data. The syntax and some of the functions for these table types aren't directly supported within Azure Synapse, but the data can be migrated into a standard table with appropriate data types and indexing or partitioning on the date/time column.

Teradata implements the temporal query functionality via query rewriting to add additional filters within a temporal query to limit the applicable date range. If this functionality is currently in use within the source Teradata environment and is to be migrated, then this additional filtering will need to be added into the relevant temporal queries.

The Azure environment also includes specific features for complex analytics on time-series data at scale called time series insights—this is aimed at IoT data analysis applications and may be more appropriate for this use-case.

Unsupported Teradata data types

> **TIP**
> Assess the impact of unsupported data types as part of the preparation phase.

Most Teradata data types have a direct equivalent in Azure Synapse. The following table shows the Teradata data types that are unsupported in Azure Synapse together with the recommended mapping. In the table, Teradata column type is the type that's stored within the system catalog—for example, in `DBC.ColumnsV`.

TERADATA COLUMN TYPE	TERADATA DATA TYPE	AZURE SYNAPSE DATA TYPE
++	TD_ANYTYPE	Not supported in Azure Synapse
A1	ARRAY	Not supported in Azure Synapse
AN	ARRAY	Not supported in Azure Synapse

TERADATA COLUMN TYPE	TERADATA DATA TYPE	AZURE SYNAPSE DATA TYPE
AT	TIME	TIME
BF	BYTE	BINARY
BO	BLOB	BLOB data type isn't directly supported but can be replaced with BINARY.
BV	VARBYTE	BINARY
CF	VARCHAR	CHAR
CO	CLOB	CLOB data type isn't directly supported but can be replaced with VARCHAR.
CV	VARCHAR	VARCHAR
D	DECIMAL	DECIMAL
DA	DATE	DATE
DH	INTERVAL DAY TO HOUR	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).
DM	INTERVAL DAY TO MINUTE	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).
DS	INTERVAL DAY TO SECOND	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).
DT	DATASET	DATASET data type is supported in Azure Synapse.
DY	INTERVAL DAY	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).
F	FLOAT	FLOAT
HM	INTERVAL HOUR TO MINUTE	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).

TERADATA COLUMN TYPE	TERADATA DATA TYPE	AZURE SYNAPSE DATA TYPE
HR	INTERVAL HOUR	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).
HS	INTERVAL HOUR TO SECOND	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).
I1	BYTEINT	TINYINT
I2	SMALLINT	SMALLINT
I8	BIGINT	BIGINT
I	INTEGER	INT
JN	JSON	JSON data type isn't currently directly supported within Azure Synapse, but JSON data can be stored in a VARCHAR field.
MI	INTERVAL MINUTE	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).
MO	INTERVAL MONTH	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).
MS	INTERVAL MINUTE TO SECOND	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).
N	NUMBER	NUMERIC
PD	PERIOD(DATE)	Can be converted to VARCHAR or split into two separate dates

TERADATA COLUMN TYPE	TERADATA DATA TYPE	AZURE SYNAPSE DATA TYPE
PM	PERIOD (TIMESTAMP WITH TIME ZONE)	Can be converted to VARCHAR or split into two separate timestamps (DATETIMEOFFSET)
PS	PERIOD(TIMESTAMP)	Can be converted to VARCHAR or split into two separate timestamps (DATETIMEOFFSET)
PT	PERIOD(TIME)	Can be converted to VARCHAR or split into two separate times
PZ	PERIOD (TIME WITH TIME ZONE)	Can be converted to VARCHAR or split into two separate times but WITH TIME ZONE isn't supported for TIME
SC	INTERVAL SECOND	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).
SZ	TIMESTAMP WITH TIME ZONE	DATETIMEOFFSET
TS	TIMESTAMP	DATETIME or DATETIME2
TZ	TIME WITH TIME ZONE	TIME WITH TIME ZONE isn't supported because TIME is stored using "wall clock" time only without a time zone offset.
XM	XML	XML data type isn't currently directly supported within Azure Synapse, but XML data can be stored in a VARCHAR field.
YM	INTERVAL YEAR TO MONTH	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).
YR	INTERVAL YEAR	INTERVAL data types aren't supported in Azure Synapse, but date calculations can be done with the date comparison functions (for example, DATEDIFF and DATEADD).

Use the metadata from the Teradata catalog tables to determine whether any of these data types are to be migrated and allow for this in the migration plan. For example, use a SQL query like this one to find any occurrences of unsupported data types that need attention.

```
SELECT
ColumnType, CASE
WHEN ColumnType = '++' THEN 'TD_ANYTYPE'
WHEN ColumnType = 'A1' THEN 'ARRAY' WHEN
ColumnType = 'AN' THEN 'ARRAY' WHEN
ColumnType = 'BO' THEN 'BLOB'
WHEN ColumnType = 'CO' THEN 'CLOB'
WHEN ColumnType = 'DH' THEN 'INTERVAL DAY TO HOUR' WHEN
ColumnType = 'DM' THEN 'INTERVAL DAY TO MINUTE' WHEN
ColumnType = 'DS' THEN 'INTERVAL DAY TO SECOND' WHEN
ColumnType = 'DT' THEN 'DATASET'
WHEN ColumnType = 'DY' THEN 'INTERVAL DAY'
WHEN ColumnType = 'HM' THEN 'INTERVAL HOUR TO MINUTE' WHEN
ColumnType = 'HR' THEN 'INTERVAL HOUR'
WHEN ColumnType = 'HS' THEN 'INTERVAL HOUR TO SECOND' WHEN
ColumnType = 'JN' THEN 'JSON'
WHEN ColumnType = 'MI' THEN 'INTERVAL MINUTE' WHEN
ColumnType = 'MO' THEN 'INTERVAL MONTH'
WHEN ColumnType = 'MS' THEN 'INTERVAL MINUTE TO SECOND' WHEN
ColumnType = 'PD' THEN 'PERIOD(DATE)'
WHEN ColumnType = 'PM' THEN 'PERIOD (TIMESTAMP WITH TIME ZONE)'
WHEN ColumnType = 'PS' THEN 'PERIOD(TIMESTAMP)' WHEN
ColumnType = 'PT' THEN 'PERIOD(TIME)'
WHEN ColumnType = 'PZ' THEN 'PERIOD (TIME WITH TIME ZONE)' WHEN
ColumnType = 'SC' THEN 'INTERVAL SECOND'
WHEN ColumnType = 'SZ' THEN 'TIMESTAMP WITH TIME ZONE' WHEN
ColumnType = 'XM' THEN 'XML'
WHEN ColumnType = 'YM' THEN 'INTERVAL YEAR TO MONTH' WHEN
ColumnType = 'YR' THEN 'INTERVAL YEAR'
END AS Data_Type,
COUNT (*) AS Data_Type_Count FROM
DBC.ColumnsV
WHERE DatabaseName IN ('UserDB1', 'UserDB2', 'UserDB3') -- select databases to be migrated
GROUP BY 1,2
ORDER BY 1;
```

TIP

Third-party tools and services can automate data mapping tasks.

There are third-party vendors who offer tools and services to automate migration, including the mapping of data types. If a third-party ETL tool such as Informatica or Talend is already in use in the Teradata environment, those tools can implement any required data transformations.

Data Definition Language (DDL) generation

TIP

Use existing Teradata metadata to automate the generation of `CREATE TABLE` and `CREATE VIEW DDL` for Azure Synapse.

Edit existing Teradata `CREATE TABLE` and `CREATE VIEW` scripts to create the equivalent definitions with modified data types as described previously if necessary. Typically, this involves removing extra Teradata-specific clauses such as `FALLBACK` or `MULTISET`.

However, all the information that specifies the current definitions of tables and views within the existing Teradata environment is maintained within system catalog tables. This is the best source of this information as it's guaranteed to be up to date and complete. Be aware that user-maintained documentation may not be in sync with the current table definitions.

Access this information via views onto the catalog such as `DBC.ColumnsV` and generate the equivalent

`CREATE TABLE` DDL statements for the equivalent tables in Azure Synapse.

> **TIP**
> Third-party tools and services can automate data mapping tasks.

There are Microsoft partners who offer tools and services to automate migration, including data-type mapping. Also, if a third-party ETL tool such as Informatica or Talend is already in use in the Teradata environment, that tool can implement any required data transformations.

SQL DML differences between Teradata and Azure Synapse

SQL Data Manipulation Language (DML)

> **TIP**
> SQL DML commands `SELECT`, `INSERT`, and `UPDATE` have standard core elements but may also implement different syntax options.

The ANSI SQL standard defines the basic syntax for DML commands such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. Both Teradata and Azure Synapse use these commands, but in some cases there are implementation differences.

The following sections discuss the Teradata-specific DML commands that you should consider during a migration to Azure Synapse.

SQL DML syntax differences

Be aware of these differences in SQL Data Manipulation Language (DML) syntax between Teradata SQL and Azure Synapse (T-SQL) when migrating:

- `QUALIFY`: Teradata supports the `QUALIFY` operator. For example:

  ```
  SELECT col1
  FROM tab1
  WHERE col1='XYZ'
  QUALIFY ROW_NUMBER () OVER (PARTITION by
  col1 ORDER BY col1) = 1;
  ```

  The equivalent Azure Synapse syntax is:

  ```
  SELECT * FROM (
  SELECT col1, ROW_NUMBER () OVER (PARTITION by col1 ORDER BY col1) rn
  FROM tab1 WHERE col1='XYZ'
  ) WHERE rn = 1;
  ```

- Date arithmetic: Azure Synapse has operators such as `DATEADD` and `DATEDIFF` which can be used on `DATE` or `DATETIME` fields. Teradata supports direct subtraction on dates such as `SELECT DATE1 - DATE2 FROM...`
- In `GROUP BY` ordinal, explicitly provide the T-SQL column name.
- `LIKE ANY`: Teradata supports `LIKE ANY` syntax such as:

```
SELECT * FROM CUSTOMER
WHERE POSTCODE LIKE ANY
('CV1%', 'CV2%', 'CV3%');
```

The equivalent in Azure Synapse syntax is:

```
SELECT * FROM CUSTOMER
WHERE
(POSTCODE LIKE 'CV1%') OR (POSTCODE LIKE 'CV2%') OR (POSTCODE LIKE 'CV3%');
```

- Depending on system settings, character comparisons in Teradata may be case insensitive by default. In Azure Synapse, character comparisons are always case sensitive.

Use EXPLAIN to validate legacy SQL

> **TIP**
> Use real queries from the existing system query logs to find potential migration issues.

One way of testing legacy Teradata SQL for compatibility with Azure Synapse is to capture some representative SQL statements from the legacy system query logs, prefix those queries with EXPLAIN, and (assuming a "like-for-like" migrated data model in Azure Synapse with the same table and column names) run those `EXPLAIN` statements in Azure Synapse. Any incompatible SQL will throw an error—use this information to determine the scale of the recoding task. This approach doesn't require that data is loaded into the Azure environment, only that the relevant tables and views have been created.

Functions, stored procedures, triggers, and sequences

> **TIP**
> As part of the preparation phase, assess the number and type of non-data objects being migrated.

When migrating from a mature legacy data warehouse environment such as Teradata, there are often elements other than simple tables and views that need to be migrated to the new target environment. Examples of this include functions, stored procedures, triggers, and sequences.

As part of the preparation phase, create an inventory of the objects that need to be migrated and define the methods for handling them. Then assign an appropriate allocation of resources in the project plan.

There may be facilities in the Azure environment that replace the functionality implemented as either functions or stored procedures in the Teradata environment. In this case, it's often more efficient to use the built-in Azure facilities rather than recoding the Teradata functions.

> **TIP**
> Third-party products and services can automate migration of non-data elements.

Microsoft partners offer tools and services that can automate the migration.

See the following sections for more information on each of these elements.

Functions

As with most database products, Teradata supports system functions and user-defined functions within the SQL implementation. When migrating to another database platform such as Azure Synapse, common system functions are available and can be migrated without change. Some system functions may have slightly different

syntax, but the required changes can be automated. System functions where there's no equivalent, such as arbitrary user-defined functions, may need to be recoded using the languages available in the target environment. Azure Synapse uses the popular Transact-SQL language to implement user-defined functions.

Stored procedures

Most modern database products allow for procedures to be stored within the database. Teradata provides the SPL language for this purpose. A stored procedure typically contains SQL statements and some procedural logic, and may return data or a status.

The dedicated SQL pools of Azure Synapse Analytics also support stored procedures using T-SQL, so if you must migrate stored procedures, recode them accordingly.

Triggers

Azure Synapse doesn't support the creation of triggers, but you can implement them within Azure Data Factory.

Sequences

Azure Synapse sequences are handled in a similar way to Teradata, using IDENTITY to create surrogate keys or managed identity.

Teradata to T-SQL mapping

This table shows the Teradata to T-SQL compliant with Azure Synapse SQL data type mapping:

TERADATA DATA TYPE	AZURE SYNAPSE SQL DATA TYPE
bigint	bigint
bool	bit
boolean	bit
byteint	tinyint
char [(*p*)]	char [(*p*)]
char varying [(*p*)]	varchar [(*p*)]
character [(*p*)]	char [(*p*)]
character varying [(*p*)]	varchar [(*p*)]
date	date
datetime	datetime
dec [(*p*[,*s*])]	decimal [(*p*[,*s*])]
decimal [(*p*[,*s*])]	decimal [(*p*[,*s*])]
double	float(53)
double precision	float(53)
float [(*p*)]	float [(*p*)]
float4	float(53)

TERADATA DATA TYPE	AZURE SYNAPSE SQL DATA TYPE
float8	float(53)
int	int
int1	tinyint
int2	smallint
int4	int
int8	bigint
integer	integer
interval	*Not supported*
national char varying [(*p*)]	nvarchar [(*p*)]
national character [(*p*)]	nchar [(*p*)]
national character varying [(*p*)]	nvarchar [(*p*)]
nchar [(*p*)]	nchar [(*p*)]
numeric [(*p*[,*s*])]	numeric [(*p*[,*s*])
nvarchar [(*p*)]	nvarchar [(*p*)]
real	real
smallint	smallint
time	time
time with time zone	datetimeoffset
time without time zone	time
timespan	*Not supported*
timestamp	datetime2
timetz	datetimeoffset
varchar [(*p*)]	varchar [(*p*)]

Summary

Typical existing legacy Teradata installations are implemented in a way that makes migration to Azure Synapse easy. They use SQL for analytical queries on large data volumes, and are in some form of dimensional data

model. These factors make them good candidates for migration to Azure Synapse.

To minimize the task of migrating the actual SQL code, follow these recommendations:

- Initial migration of the data warehouse should be as-is to minimize risk and time taken, even if the eventual final environment will incorporate a different data model such as data vault.
- Consider using a Teradata instance in an Azure VM as a stepping stone as part of the migration process.
- Understand the differences between Teradata SQL implementation and Azure Synapse.
- Use metadata and query logs from the existing Teradata implementation to assess the impact of the differences and plan an approach to mitigate.
- Automate the process wherever possible to minimize errors, risk, and time for the migration.
- Consider using specialist Microsoft partners and services to streamline the migration.

Next steps

To learn more about Microsoft and third-party tools, see the next article in this series: Tools for Teradata data warehouse migration to Azure Synapse Analytics.

Tools for Teradata data warehouse migration to Azure Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

This article is part six of a seven-part series that provides guidance on how to migrate from Teradata to Azure Synapse Analytics. The focus of this article is best practices for Microsoft and third-party tools.

Data warehouse migration tools

By migrating your existing data warehouse to Azure Synapse, you benefit from:

- A globally secure, scalable, low-cost, cloud-native, pay-as-you-use analytical database.
- The rich Microsoft analytical ecosystem that exists on Azure. This ecosystem consists of technologies to help modernize your data warehouse once it's migrated and extend your analytical capabilities to drive new value.

Several tools from both Microsoft and third-party partners can help you migrate your existing data warehouse to Azure Synapse. This article discusses the following types of tools:

- Microsoft data and database migration tools.
- Third-party data warehouse automation tools to automate and document the migration to Azure Synapse.
- Third-party data warehouse migration tools to migrate schema and data to Azure Synapse.
- Third-party tools to bridge the SQL differences between your existing data warehouse DBMS and Azure Synapse.

Microsoft data migration tools

Microsoft offers several tools to help you migrate your existing data warehouse to Azure Synapse, such as:

- Azure Data Factory.
- Microsoft services for physical data transfer.
- Microsoft services for data ingestion.

The next sections discuss these tools in more detail.

Microsoft Azure Data Factory

Data Factory is a fully managed, pay-as-you-use, hybrid data integration service for highly scalable ETL and ELT processing. It uses Apache Spark to process and analyze data in parallel and in-memory to maximize throughput.

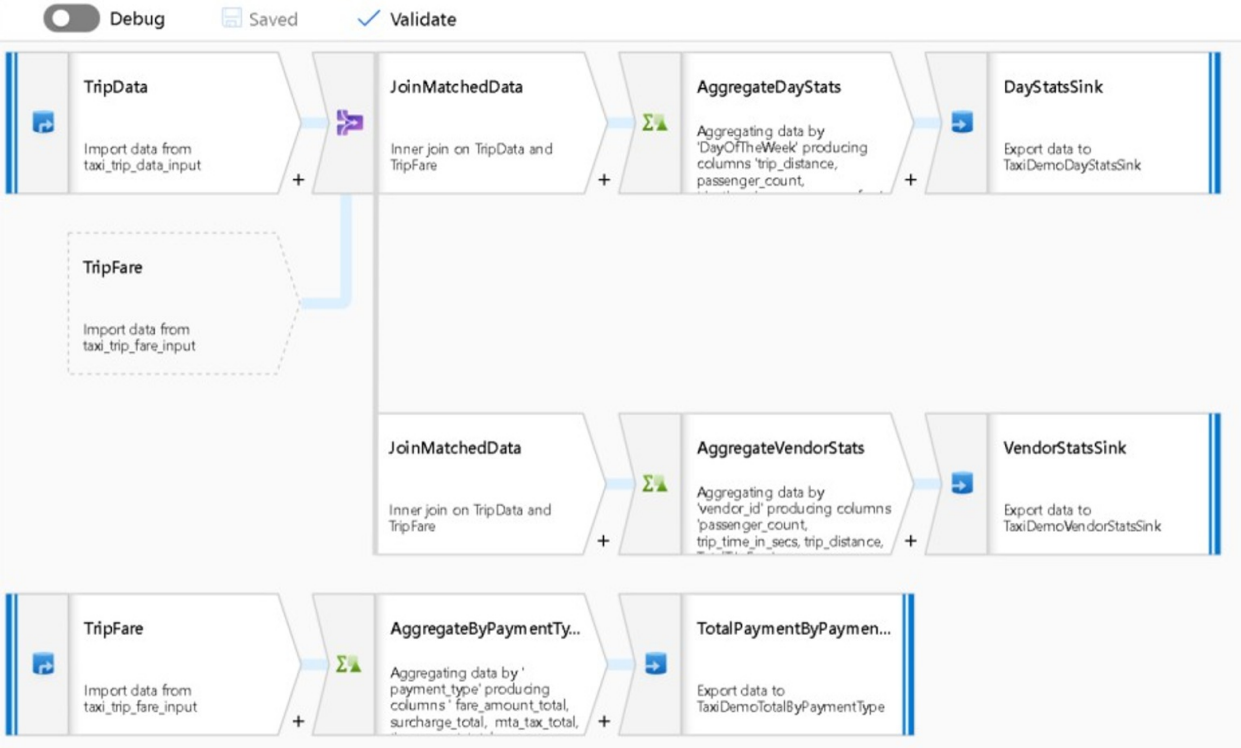
> **TIP**
> Data Factory allows you to build scalable data integration pipelines code-free.

Data Factory connectors support connections to external data sources and databases and include templates for common data integration tasks. A visual front-end, browser-based UI enables non-programmers to create and

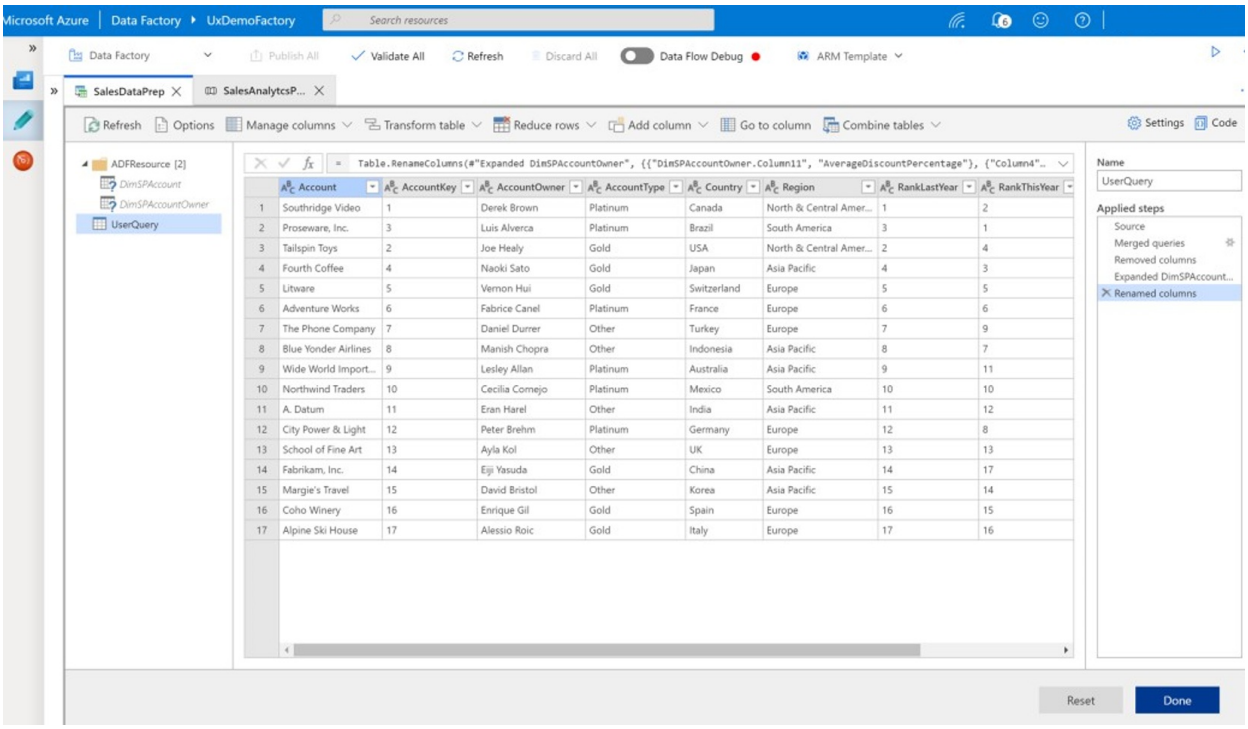
run pipelines to ingest, transform, and load data. More experienced programmers can incorporate custom code, such as Python programs.

> **TIP**
> Data Factory enables collaborative development between business and IT professionals.

Data Factory is also an orchestration tool and is the best Microsoft tool to automate the end-to-end migration process. Automation reduces the risk, effort, and time to migrate, and makes the migration process easily repeatable. The following diagram shows a mapping data flow in Data Factory.



The next screenshot shows a wrangling data flow in Data Factory.



In Data Factory, you can develop simple or comprehensive ETL and ELT processes without coding or

maintenance with just a few clicks. ETL/ELT processes ingest, move, prepare, transform, and process your data. You can design and manage scheduling and triggers in Data Factory to build an automated data integration and loading environment. In Data Factory, you can define, manage, and schedule PolyBase bulk data load processes.

> **TIP**
> Data Factory includes tools to help migrate both your data and your entire data warehouse to Azure.

You can use Data Factory to implement and manage a hybrid environment with on-premises, cloud, streaming, and SaaS data in a secure and consistent way. SaaS data might come from applications such as Salesforce.

Wrangling data flows is a new capability in Data Factory. This capability opens up Data Factory to business users who want to visually discover, explore, and prepare data at scale without writing code. Wrangling data flows offer self-service data preparation, similar to Microsoft Excel, Power Query, and Microsoft Power BI dataflows. Business users can prepare and integrate data through a spreadsheet-style UI with drop-down transform options.

Data Factory is the recommended approach for implementing data integration and ETL/ELT processes in the Azure Synapse environment, especially if you want to refactor existing legacy processes.

Microsoft services for physical data transfer

The following sections discuss a range of products and services that Microsoft offers to assist customers with data transfer.

Azure ExpressRoute

Azure ExpressRoute creates private connections between Azure data centers and infrastructure on your premises or in a collocation environment. ExpressRoute connections don't go over the public internet, and offer more reliability, faster speeds, and lower latencies than typical internet connections. In some cases, you gain significant cost benefits by using ExpressRoute connections to transfer data between on-premises systems and Azure.

AzCopy

AzCopy is a command line utility that copies files to Azure Blob Storage over a standard internet connection. In a warehouse migration project, you can use AzCopy to upload extracted, compressed, delimited text files before loading them into Azure Synapse using PolyBase. AzCopy can upload individual files, file selections, or file folders. If the exported files are in Parquet format, use a native Parquet reader instead.

Azure Data Box

Azure Data Box is a Microsoft service that provides you with a proprietary physical storage device that you can copy migration data onto. You then ship the device to an Azure data center for data upload to cloud storage. This service can be cost-effective for large volumes of data, such as tens or hundreds of terabytes, or where network bandwidth isn't readily available. Azure Data Box is typically used for a large one-off historical data load into Azure Synapse.

Azure Data Box Gateway

Azure Data Box Gateway is a virtualized cloud storage gateway device that resides on your premises and sends your images, media, and other data to Azure. Use Data Box Gateway for one-off migration tasks or ongoing incremental data uploads.

Microsoft services for data ingestion

The following sections discuss the products and services that Microsoft offers to assist customers with data ingestion.

COPY INTO

The COPY INTO statement provides the most flexibility for high-throughput data ingestion into Azure Synapse. For more information about `COPY INTO` capabilities, see COPY (Transact-SQL).

PolyBase

PolyBase is the fastest, most scalable method for bulk data load into Azure Synapse. PolyBase uses the massively parallel processing (MPP) architecture of Azure Synapse for parallel loading of data to achieve the fastest throughput. PolyBase can read data from flat files in Azure Blob Storage, or directly from external data sources and other relational databases via connectors.

> **TIP**
> PolyBase can load data in parallel from Azure Blob Storage into Azure Synapse.

PolyBase can also directly read from files compressed with gzip to reduce the physical volume of data during a load process. PolyBase supports popular data formats such as delimited text, ORC, and Parquet.

> **TIP**
> You can invoke PolyBase from Data Factory as part of a migration pipeline.

PolyBase is tightly integrated with Data Factory to support rapid development of data load ETL/ELT processes. You can schedule data load processes through a visual UI for higher productivity and fewer errors than hand-written code. Microsoft recommends PolyBase for data ingestion into Azure Synapse, especially for high-volume data ingestion.

PolyBase uses `CREATE TABLE AS` or `INSERT...SELECT` statements to load data. `CREATE TABLE AS` minimizes logging to achieve the highest throughput. The most efficient input format for data load is compressed delimited text files. For maximum throughput, split large input files into multiple smaller files and load them in parallel. For fastest loading to a staging table, define the target table as `HEAP` type and use round-robin distribution.

PolyBase has some limitations, it requires data row length to be less than 1 megabyte and doesn't support fixed-width nested formats like JSON and XML.

Microsoft partners for Teradata migrations

Microsoft partners offer tools, services, and expertise to help you migrate your legacy on-premises data warehouse platform to Azure Synapse.

Next steps

To learn more about implementing modern data warehouses, see the next article in this series: Beyond Teradata migration, implement a modern data warehouse in Microsoft Azure.

Beyond Teradata migration, implement a modern data warehouse in Microsoft Azure

8/19/2022 • 23 minutes to read • Edit Online

This article is part seven of a seven-part series that provides guidance on how to migrate from Teradata to Azure Synapse Analytics. The focus of this article is best practices for implementing modern data warehouses.

Beyond data warehouse migration to Azure

A key reason to migrate your existing data warehouse to Azure Synapse Analytics is to utilize a globally secure, scalable, low-cost, cloud-native, pay-as-you-use analytical database. With Azure Synapse, you can integrate your migrated data warehouse with the complete Microsoft Azure analytical ecosystem to take advantage of other Microsoft technologies and modernize your migrated data warehouse. Those technologies include:

- Azure Data Lake Storage for cost effective data ingestion, staging, cleansing, and transformation. Data Lake Storage can free up the data warehouse capacity occupied by fast-growing staging tables.
- Azure Data Factory for collaborative IT and self-service data integration with connectors to cloud and on-premises data sources and streaming data.
- Common Data Model to share consistent trusted data across multiple technologies, including:
  - Azure Synapse
  - Azure Synapse Spark
  - Azure HDInsight
  - Power BI
  - Adobe Customer Experience Platform
  - Azure IoT
  - Microsoft ISV partners
- Microsoft data science technologies, including:
  - Azure Machine Learning studio
  - Azure Machine Learning
  - Azure Synapse Spark (Spark as a service)
  - Jupyter Notebooks
  - RStudio
  - ML.NET
  - .NET for Apache Spark, which lets data scientists use Azure Synapse data to train machine learning models at scale.
- Azure HDInsight to process large amounts of data, and to join big data with Azure Synapse data by creating a logical data warehouse using PolyBase.
- Azure Event Hubs, Azure Stream Analytics, and Apache Kafka to integrate live streaming data from Azure Synapse.

The growth of big data has led to an acute demand for machine learning to enable custom-built, trained machine learning models for use in Azure Synapse. Machine learning models enable in-database analytics to run at scale in batch, on an event-driven basis and on-demand. The ability to take advantage of in-database analytics in Azure Synapse from multiple BI tools and applications also guarantees consistent predictions and

recommendations.

In addition, you can integrate Azure Synapse with Microsoft partner tools on Azure to shorten time to value.

Let's take a closer look at how you can take advantage of technologies in the Microsoft analytical ecosystem to modernize your data warehouse after you've migrated to Azure Synapse.

Offload data staging and ETL processing to Data Lake Storage and Data Factory

Digital transformation has created a key challenge for enterprises by generating a torrent of new data for capture and analysis. A good example is transaction data created by opening online transactional processing (OLTP) systems to service access from mobile devices. Much of this data finds its way into data warehouses, and OLTP systems are the main source. With customers now driving the transaction rate rather than employees, the volume of data in data warehouse staging tables has been growing rapidly.

With the rapid influx of data into the enterprise, along with new sources of data like Internet of Things (IoT), companies must find ways to scale up data integration ETL processing. One method is to offload ingestion, data cleansing, transformation, and integration to a data lake and process data at scale there, as part of a data warehouse modernization program.

Once you've migrated your data warehouse to Azure Synapse, Microsoft can modernize your ETL processing by ingesting and staging data in Data Lake Storage. You can then clean, transform, and integrate your data at scale using Data Factory before loading it into Azure Synapse in parallel using PolyBase.

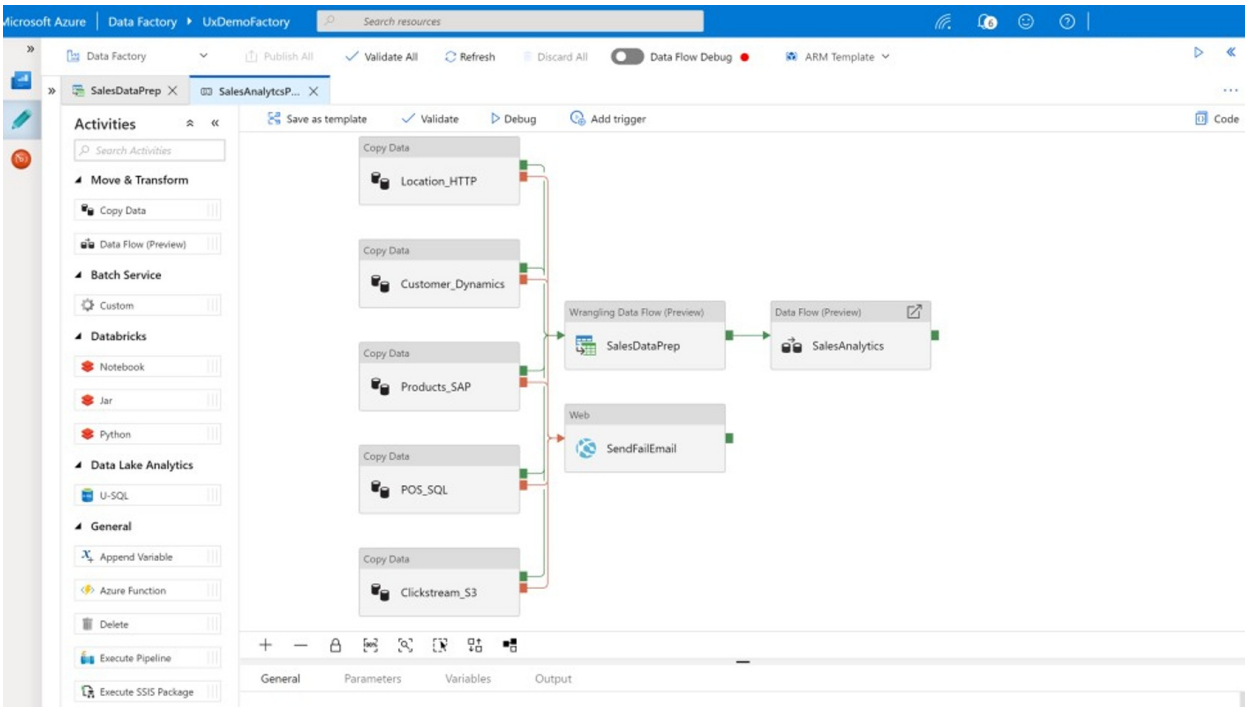
For ELT strategies, consider offloading ELT processing to Data Lake Storage to easily scale as your data volume or frequency grows.

Microsoft Azure Data Factory

Azure Data Factory is a pay-as-you-use, hybrid data integration service for highly scalable ETL and ELT processing. Data Factory provides a web-based UI to build data integration pipelines with no code. With Data Factory, you can:

- Build scalable data integration pipelines code-free.
- Easily acquire data at scale.
- Pay only for what you use.
- Connect to on-premises, cloud, and SaaS-based data sources.
- Ingest, move, clean, transform, integrate, and analyze cloud and on-premises data at scale.
- Seamlessly author, monitor, and manage pipelines that span data stores both on-premises and in the cloud.
- Enable pay-as-you-go scale-out in alignment with customer growth.

You can use these features without writing any code, or you can add custom code to Data Factory pipelines. The following screenshot shows an example Data Factory pipeline.



TIP

Data Factory lets you to build scalable data integration pipelines without code.

Implement Data Factory pipeline development from any of several places, including:

- Microsoft Azure portal.
- Microsoft Azure PowerShell.
- Programmatically from .NET and Python using a multi-language SDK.
- Azure Resource Manager (ARM) templates.
- REST APIs.

TIP

Data Factory can connect to on-premises, cloud, and SaaS data.

Developers and data scientists who prefer to write code can easily author Data Factory pipelines in Java, Python, and .NET using the software development kits (SDKs) available for those programming languages. Data Factory pipelines can be hybrid data pipelines because they can connect, ingest, clean, transform, and analyze data in on-premises data centers, Microsoft Azure, other clouds, and SaaS offerings.

After you develop Data Factory pipelines to integrate and analyze data, you can deploy those pipelines globally and schedule them to run in batch, invoke them on demand as a service, or run them in real-time on an event-driven basis. A Data Factory pipeline can also run on one or more execution engines and monitor execution to ensure performance and to track errors.

TIP

In Azure Data Factory, pipelines control the integration and analysis of data. Data Factory is enterprise-class data integration software aimed at IT professionals and has data wrangling capability for business users.

Use cases

Data Factory supports multiple use cases, such as:

- Prepare, integrate, and enrich data from cloud and on-premises data sources to populate your migrated data warehouse and data marts on Microsoft Azure Synapse.
- Prepare, integrate, and enrich data from cloud and on-premises data sources to produce training data for use in machine learning model development and in retraining analytical models.
- Orchestrate data preparation and analytics to create predictive and prescriptive analytical pipelines for processing and analyzing data in batch, such as sentiment analytics. Either act on the results of the analysis or populate your data warehouse with the results.
- Prepare, integrate, and enrich data for data-driven business applications running on the Azure cloud on top of operational data stores such as Azure Cosmos DB.

> **TIP**
> Build training data sets in data science to develop machine learning models.

Data sources

Data Factory lets you use connectors from both cloud and on-premises data sources. Agent software, known as a *self-hosted integration runtime*, securely accesses on-premises data sources and supports secure, scalable data transfer.

Transform data using Azure Data Factory

Within a Data Factory pipeline, you can ingest, clean, transform, integrate, and analyze any type of data from these sources. Data can be structured, semi-structured like JSON or Avro, or unstructured.

Without writing any code, professional ETL developers can use Data Factory mapping data flows to filter, split, join several types, lookup, pivot, unpivot, sort, union, and aggregate data. In addition, Data Factory supports surrogate keys, multiple write processing options like insert, upsert, update, table recreation, and table truncation, and several types of target data stores—also known as sinks. ETL developers can also create aggregations, including time-series aggregations that require a window to be placed on data columns.

> **TIP**
> Professional ETL developers can use Data Factory mapping data flows to clean, transform, and integrate data without the need to write code.

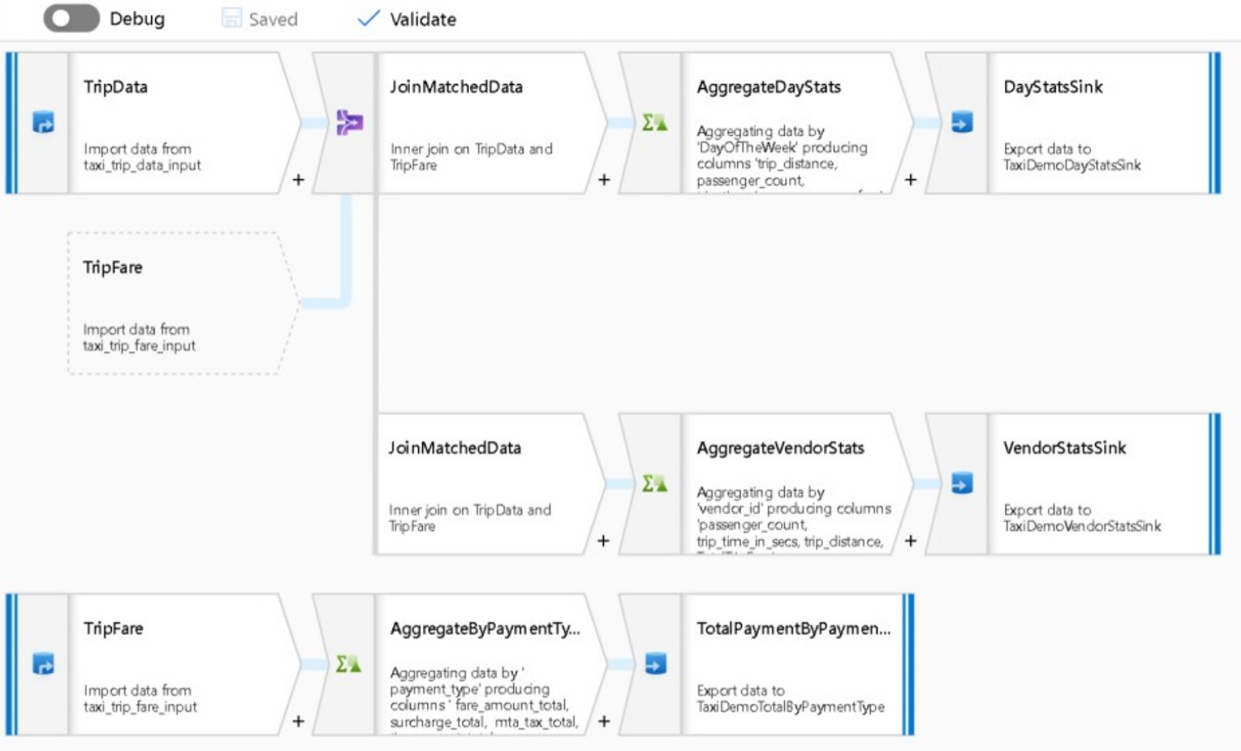
You can run mapping data flows that transform data as activities in a Data Factory pipeline, and if necessary, you can include multiple mapping data flows in a single pipeline. In this way, you can manage the complexity by breaking up challenging data transformation and integration tasks into smaller mapping dataflows that can be combined. And, you can add custom code when needed. In addition to this functionality, Data Factory mapping data flows include the ability to:

- Define expressions to clean and transform data, compute aggregations, and enrich data. For example, these expressions can perform feature engineering on a date field to break it into multiple fields to create training data during machine learning model development. You can construct expressions from a rich set of functions that include mathematical, temporal, split, merge, string concatenation, conditions, pattern match, replace, and many other functions.
- Automatically handle schema drift so that data transformation pipelines can avoid being impacted by schema changes in data sources. This ability is especially important for streaming IoT data, where schema changes can happen without notice if devices are upgraded or when readings are missed by gateway devices collecting IoT data.

- Partition data to enable transformations to run in parallel at scale.
- Inspect streaming data to view the metadata of a stream you're transforming.

> **TIP**
> Data Factory supports the ability to automatically detect and manage schema changes in inbound data, such as in streaming data.

The following screenshot shows an example Data Factory mapping data flow.



Data engineers can profile data quality and view the results of individual data transforms by enabling debug capability during development.

> **TIP**
> Data Factory can also partition data to enable ETL processing to run at scale.

If necessary, you can extend Data Factory transformational and analytical functionality by adding a linked service that contains your code into a pipeline. For example, an Azure Synapse Spark pool notebook might contain Python code that uses a trained model to score the data integrated by a mapping data flow.

You can store integrated data and any results from analytics within a Data Factory pipeline in one or more data stores, such as Data Lake Storage, Azure Synapse, or Hive tables in HDInsight. You can also invoke other activities to act on insights produced by a Data Factory analytical pipeline.

> **TIP**
> Data Factory pipelines are extensible because Data Factory lets you write your own code and run it as part of a pipeline.

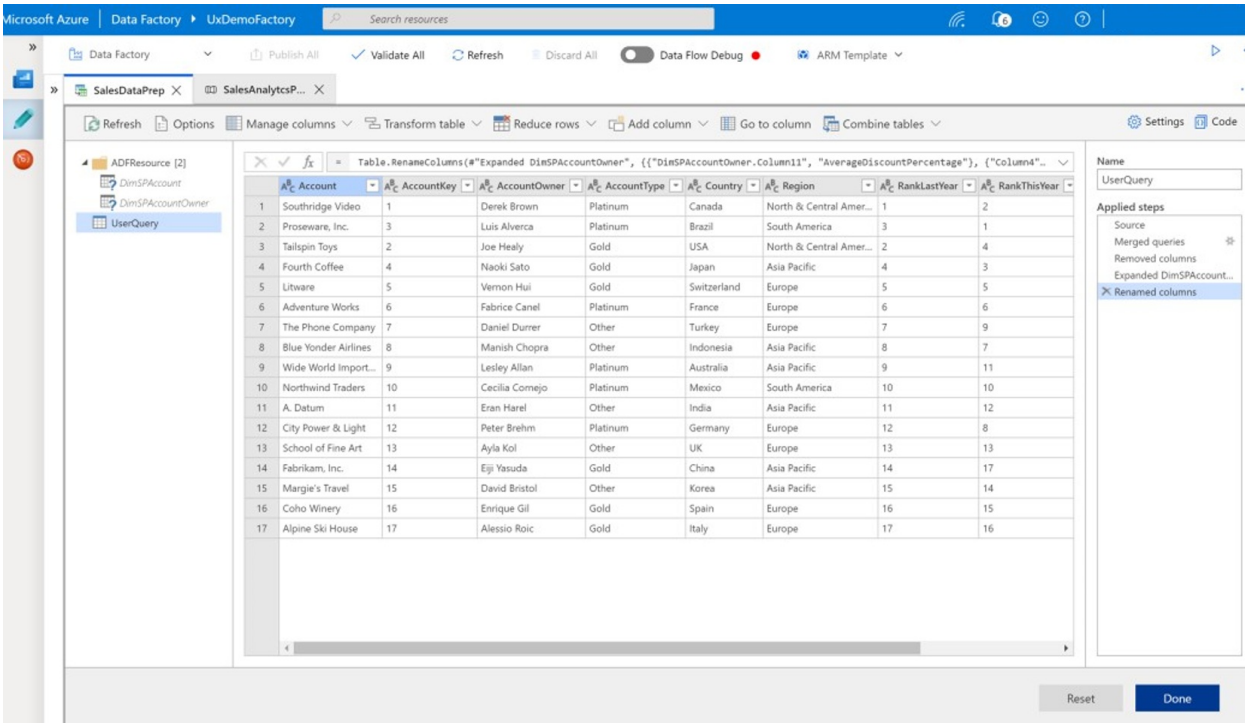
Utilize Spark to scale data integration

At run time, Data Factory internally uses Azure Synapse Spark pools, which are Microsoft's Spark as a service offering, to clean and integrate data in the Azure cloud. You can clean, integrate, and analyze high-volume, high-

velocity data, such as click-stream data, at scale. Microsoft's intention is to also run Data Factory pipelines on other Spark distributions. In addition to running ETL jobs on Spark, Data Factory can invoke Pig scripts and Hive queries to access and transform data stored in HDInsight.

Link self-service data prep and Data Factory ETL processing using wrangling data flows

Data wrangling lets business users, also known as citizen data integrators and data engineers, make use of the platform to visually discover, explore, and prepare data at scale without writing code. This Data Factory capability is easy to use and is similar to Microsoft Excel Power Query or Microsoft Power BI dataflows, where self-service business users use a spreadsheet-style UI with drop-down transforms to prepare and integrate data. The following screenshot shows an example Data Factory wrangling data flow.



Unlike Excel and Power BI, Data Factory wrangling data flows use Power Query to generate M code and then translate it into a massively parallel in-memory Spark job for cloud-scale execution. The combination of mapping data flows and wrangling data flows in Data Factory lets professional ETL developers and business users collaborate to prepare, integrate, and analyze data for a common business purpose. The preceding Data Factory mapping data flows diagram shows how both Data Factory and Azure Synapse Spark pool notebooks can be combined in the same Data Factory pipeline. The combination of mapping and wrangling data flows in Data Factory helps IT and business users stay aware of what data flows each has created, and supports data flow reuse to minimize reinvention and maximize productivity and consistency.

> **TIP**
>
> Data Factory supports both wrangling data flows and mapping data flows, so business users and IT users can integrate data collaboratively on a common platform.

Link data and analytics in analytical pipelines

In addition to cleaning and transforming data, Data Factory can combine data integration and analytics in the same pipeline. You can use Data Factory to create both data integration and analytical pipelines, the latter being an extension of the former. You can drop an analytical model into a pipeline to create an analytical pipeline that generates clean, integrated data for predictions or recommendations. Then, you can act on the predictions or recommendations immediately, or store them in your data warehouse to provide new insights and recommendations that can be viewed in BI tools.

To batch score your data, you can develop an analytical model that you invoke as a service within a Data Factory pipeline. You can develop analytical models code-free with Azure Machine Learning studio, or with the Azure

Machine Learning SDK using Azure Synapse Spark pool notebooks or R in RStudio. When you run Spark machine learning pipelines on Azure Synapse Spark pool notebooks, analysis happens at scale.

You can store integrated data and any Data Factory analytical pipeline results in one or more data stores, such as Data Lake Storage, Azure Synapse, or Hive tables in HDInsight. You can also invoke other activities to act on insights produced by a Data Factory analytical pipeline.

Use a lake database to share consistent trusted data

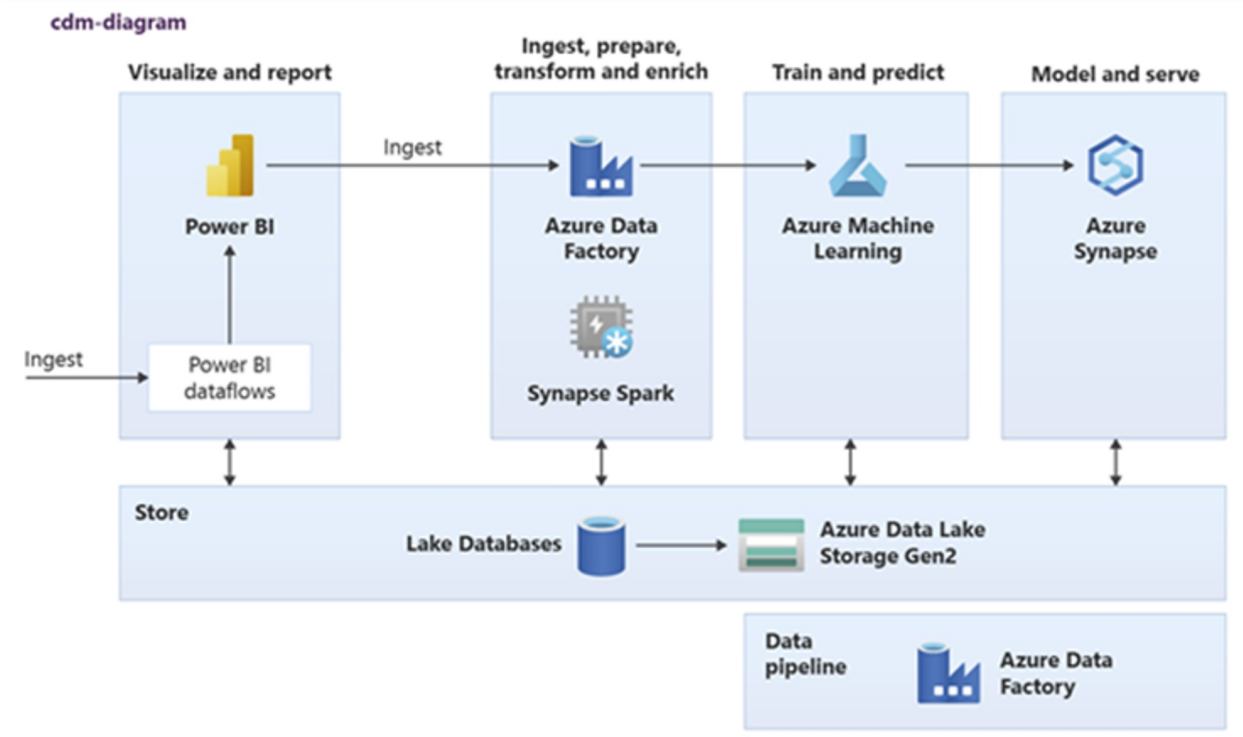
A key objective of any data integration setup is the ability to integrate data once and reuse it everywhere, not just in a data warehouse. For example, you might want to use integrated data in data science. Reuse avoids reinvention and ensures consistent, commonly understood data that everyone can trust.

Common Data Model describes core data entities that can be shared and reused across the enterprise. To achieve reuse, Common Data Model establishes a set of common data names and definitions that describe logical data entities. Examples of common data names include Customer, Account, Product, Supplier, Orders, Payments, and Returns. IT and business professionals can use data integration software to create and store common data assets to maximize their reuse and drive consistency everywhere.

Azure Synapse provides industry-specific database templates to help standardize data in the lake. Lake database templates provide schemas for predefined business areas, enabling data to be loaded into a lake database in a structured way. The power comes when you use data integration software to create lake database common data assets, resulting in self-describing trusted data that can be consumed by applications and analytical systems. You can create common data assets in Data Lake Storage by using Data Factory.

> **TIP**
> Data Lake Storage is shared storage that underpins Microsoft Azure Synapse, Azure Machine Learning, Azure Synapse Spark, and HDInsight.

Power BI, Azure Synapse Spark, Azure Synapse, and Azure Machine Learning can consume common data assets. The following diagram shows how a lake database can be used in Azure Synapse.



> **TIP**
> Integrate data to create lake database logical entities in shared storage to maximize the reuse of common data assets.

Integration with Microsoft data science technologies on Azure

Another key objective when modernizing a data warehouse is to produce insights for competitive advantage. You can produce insights by integrating your migrated data warehouse with Microsoft and third-party data science technologies in Azure. The following sections describe machine learning and data science technologies offered by Microsoft to see how they can be used with Azure Synapse in a modern data warehouse environment.

Microsoft technologies for data science on Azure

Microsoft offers a range of technologies that support advance analysis. With these technologies, you can build predictive analytical models using machine learning or analyze unstructured data using deep learning. The technologies include:

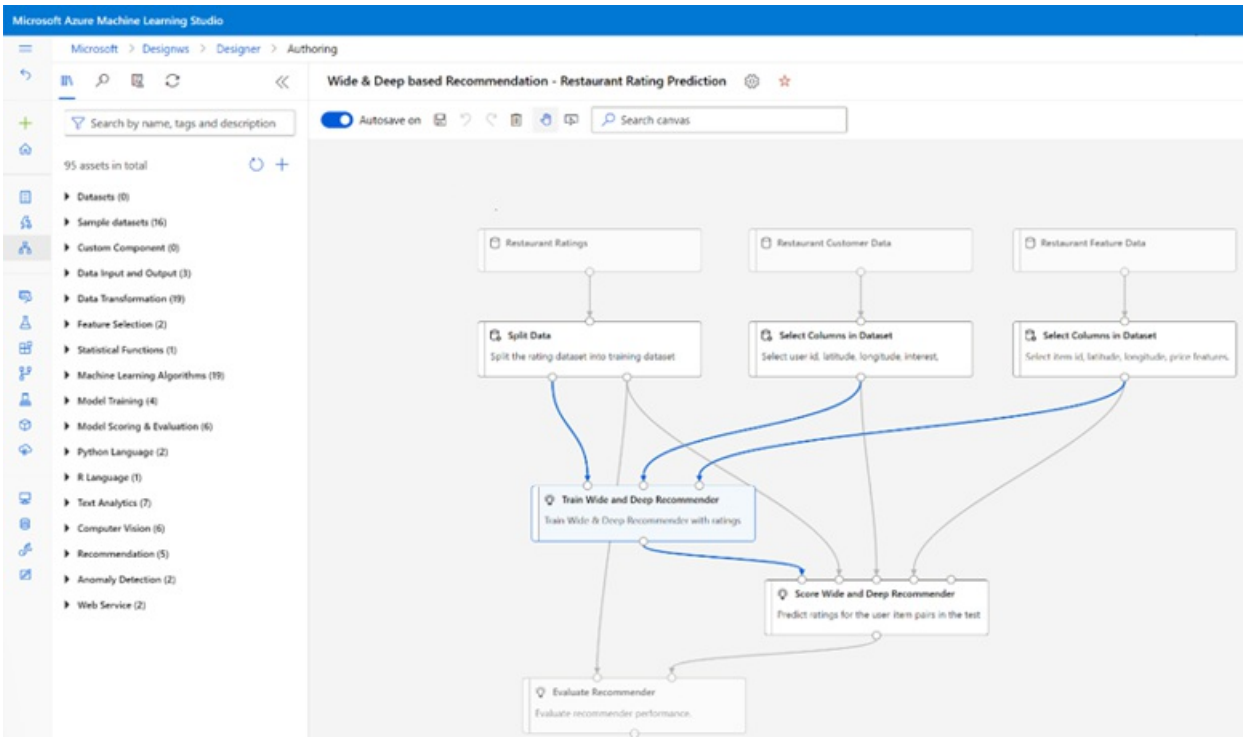
- Azure Machine Learning studio
- Azure Machine Learning
- Azure Synapse Spark pool notebooks
- ML.NET (API, CLI, or ML.NET Model Builder for Visual Studio)
- .NET for Apache Spark

Data scientists can use RStudio (R) and Jupyter Notebooks (Python) to develop analytical models, or they can use frameworks such as Keras or TensorFlow.

> **TIP**
> Develop machine learning models using a no/low-code approach or by using programming languages like Python, R, and .NET.

Azure Machine Learning studio

Azure Machine Learning studio is a fully managed cloud service that lets you build, deploy, and share predictive analytics using a drag-and-drop, web-based UI. The following screenshot shows the Azure Machine Learning studio UI.



Azure Machine Learning

Azure Machine Learning provides an SDK and services for Python that supports can help you quickly prepare data and also train and deploy machine learning models. You can use Azure Machine Learning in Azure notebooks using Jupyter Notebook, with open-source frameworks, such as PyTorch, TensorFlow, scikit-learn, or Spark MLlib—the machine learning library for Spark. Azure Machine Learning provides an AutoML capability that automatically tests multiple algorithms to identify the most accurate algorithms to expedite model development.

> **TIP**
> Azure Machine Learning provides an SDK for developing machine learning models using several open-source frameworks.

You can also use Azure Machine Learning to build machine learning pipelines that manage end-to-end workflow, programmatically scale in the cloud, and deploy models both to the cloud and the edge. Azure Machine Learning contains workspaces, which are logical spaces that you can programmatically or manually create in the Azure portal. These workspaces keep compute targets, experiments, data stores, trained machine learning models, Docker images, and deployed services all in one place to enable teams to work together. You can use Azure Machine Learning in Visual Studio with the Visual Studio for AI extension.

> **TIP**
> Organize and manage related data stores, experiments, trained models, Docker images, and deployed services in workspaces.

Azure Synapse Spark pool notebooks

An Azure Synapse Spark pool notebook is an Azure-optimized Apache Spark service. With Azure Synapse Spark pool notebooks:

- Data engineers can build and run scalable data preparation jobs using Data Factory.
- Data scientists can build and run machine learning models at scale using notebooks written in languages such as Scala, R, Python, Java, and SQL to visualize results.

> **TIP**
> Azure Synapse Spark is a dynamically scalable Spark as a service offering from Microsoft, Spark offers scalable execution of data preparation, model development, and deployed model execution.

Jobs running in Azure Synapse Spark pool notebooks can retrieve, process, and analyze data at scale from Azure Blob Storage, Data Lake Storage, Azure Synapse, HDInsight, and streaming data services such as Apache Kafka.

> **TIP**
> Azure Synapse Spark can access data in a range of Microsoft analytical ecosystem data stores on Azure.

Azure Synapse Spark pool notebooks support autoscaling and auto-termination to reduce total cost of ownership (TCO). Data scientists can use the MLflow open-source framework to manage the machine learning lifecycle.

ML.NET

ML.NET is an open-source, cross-platform machine learning framework for Windows, Linux, macOS. Microsoft created ML.NET so that .NET developers can use existing tools, such as ML.NET Model Builder for Visual Studio, to develop custom machine learning models and integrate them into their .NET applications.

> **TIP**
> Microsoft has extended its machine learning capability to .NET developers.

.NET for Apache Spark

.NET for Apache Spark extends Spark support beyond R, Scala, Python, and Java to .NET and aims to make Spark accessible to .NET developers across all Spark APIs. While .NET for Apache Spark is currently only available on Apache Spark in HDInsight, Microsoft intends to make .NET for Apache Spark available on Azure Synapse Spark pool notebooks.

Use Azure Synapse Analytics with your data warehouse

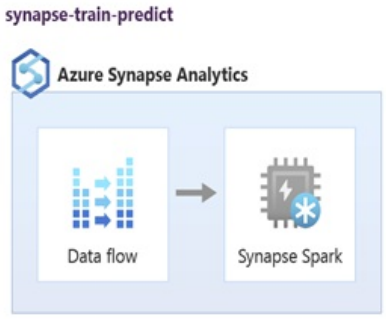
To combine machine learning models with Azure Synapse, you can:

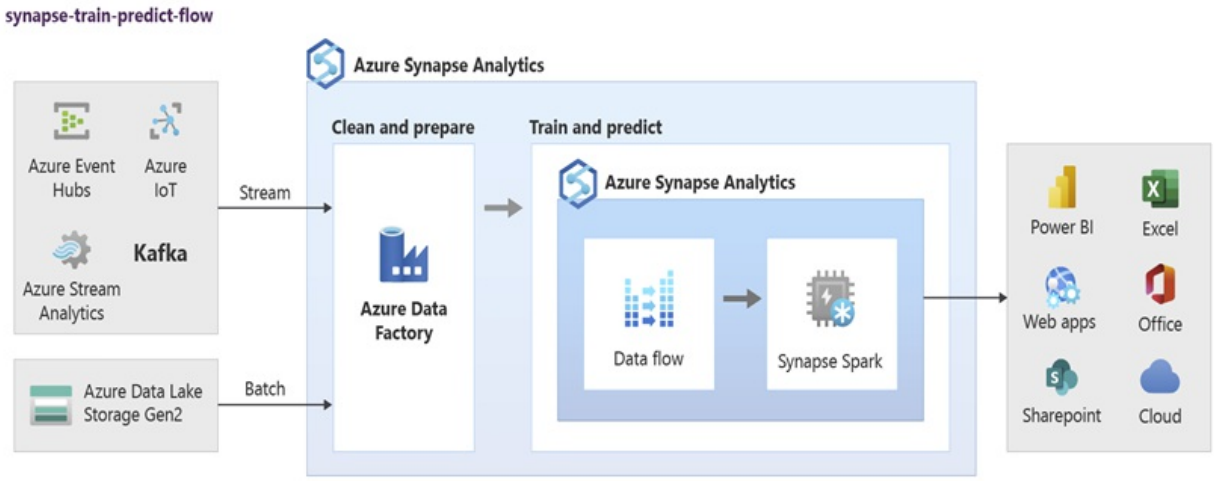
- Use machine learning models in batch or in real-time on streaming data to produce new insights, and add those insights to what you already know in Azure Synapse.
- Use the data in Azure Synapse to develop and train new predictive models for deployment elsewhere, such as in other applications.
- Deploy machine learning models, including models trained elsewhere, in Azure Synapse to analyze data in your data warehouse and drive new business value.

> **TIP**
> Train, test, evaluate, and run machine learning models at scale on Azure Synapse Spark pool notebooks by using data in Azure Synapse.

Data scientists can use RStudio, Jupyter Notebooks, and Azure Synapse Spark pool notebooks together with Azure Machine Learning to develop machine learning models that run at scale on Azure Synapse Spark pool notebooks using data in Azure Synapse. For example, data scientists could create an unsupervised model to segment customers to drive different marketing campaigns. Use supervised machine learning to train a model to predict a specific outcome, such as to predict a customer's propensity to churn, or to recommend the next

best offer for a customer to try to increase their value. The following diagram shows how Azure Synapse can be used for Azure Machine Learning.





In another scenario, you can ingest social network or review website data into Data Lake Storage, then prepare and analyze the data at scale on an Azure Synapse Spark pool notebook using natural language processing to score customer sentiment about your products or brand. You can then add those scores to your data warehouse. By using big data analytics to understand the effect of negative sentiment on product sales, you add to what you already know in your data warehouse.

> **TIP**
> Produce new insights using machine learning on Azure in batch or in real-time and add to what you know in your data warehouse.

Integrate live streaming data into Azure Synapse Analytics

When analyzing data in a modern data warehouse, you must be able to analyze streaming data in real-time and join it with historical data in your data warehouse. An example is combining IoT data with product or asset data.

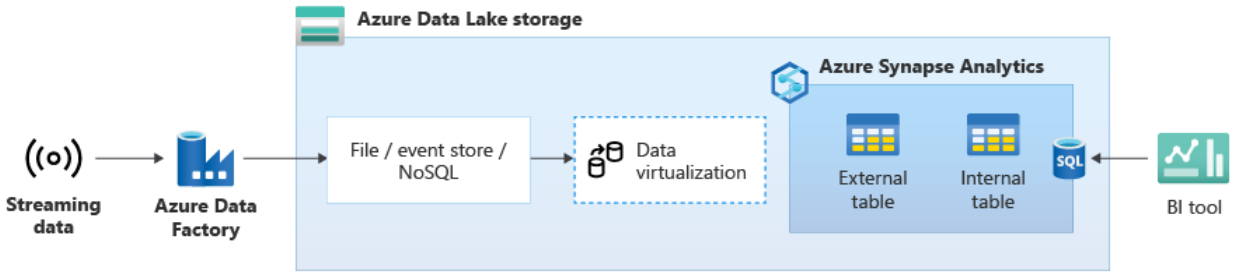
> **TIP**
> Integrate your data warehouse with streaming data from IoT devices or clickstreams.

Once you've successfully migrated your data warehouse to Azure Synapse, you can introduce live streaming data integration as part of a data warehouse modernization exercise by taking advantage of the extra functionality in Azure Synapse. To do so, ingest streaming data via Event Hubs, other technologies like Apache Kafka, or potentially your existing ETL tool if it supports the streaming data sources. Store the data in Data Lake Storage. Then, create an external table in Azure Synapse using PolyBase and point it at the data being streamed into Data Lake Storage so that your data warehouse now contains new tables that provide access to the real-time streaming data. Query the external table as if the data was in the data warehouse by using standard T-SQL

from any BI tool that has access to Azure Synapse. You can also join the streaming data to other tables with historical data to create views that join live streaming data to historical data to make it easier for business users to access the data.

> **TIP**
> Ingest streaming data into Data Lake Storage from Event Hubs or Apache Kafka, and access the data from Azure Synapse using PolyBase external tables.

In the following diagram, a real-time data warehouse on Azure Synapse is integrated with streaming data in Data Lake Storage.



Create a logical data warehouse using PolyBase

With PolyBase, you can create a logical data warehouse to simplify user access to multiple analytical data stores. Many companies have adopted "workload optimized" analytical data stores over the last several years in addition to their data warehouses. The analytical platforms on Azure include:

- Data Lake Storage with Azure Synapse Spark pool notebook (Spark as a service), for big data analytics.
- HDInsight (Hadoop as a service), also for big data analytics.
- NoSQL Graph databases for graph analysis, which could be done in Azure Cosmos DB.
- Event Hubs and Stream Analytics, for real-time analysis of data in motion.

You might have non-Microsoft equivalents of these platforms, or a master data management (MDM) system that needs to be accessed for consistent trusted data on customers, suppliers, products, assets, and more.

> **TIP**
> PolyBase simplifies access to multiple underlying analytical data stores on Azure for ease of access by business users.

Those analytical platforms emerged because of the explosion of new data sources inside and outside the enterprise and the demand by business users to capture and analyze the new data. The new data sources include:

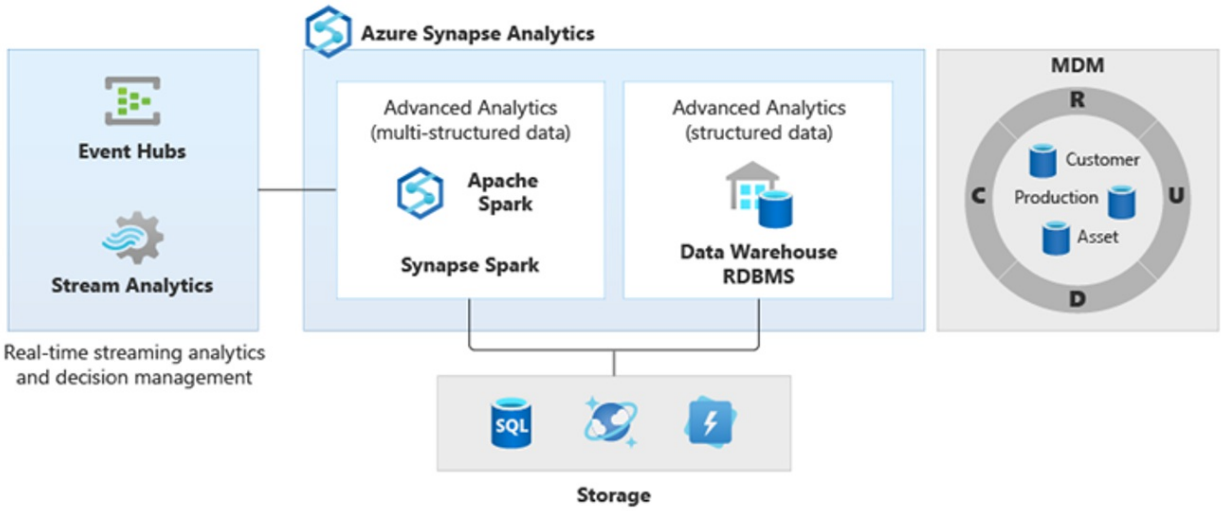
- Machine generated data, such as IoT sensor data and clickstream data.
- Human generated data, such as social network data, review web site data, customer inbound email, images, and video.
- Other external data, such as open government data and weather data.

This new data goes beyond the structured transaction data and main data sources that typically feed data warehouses and often includes:

- Semi-structured data like JSON, XML, or Avro.

- Unstructured data like text, voice, image, or video, which is more complex to process and analyze.
- High volume data, high velocity data, or both.

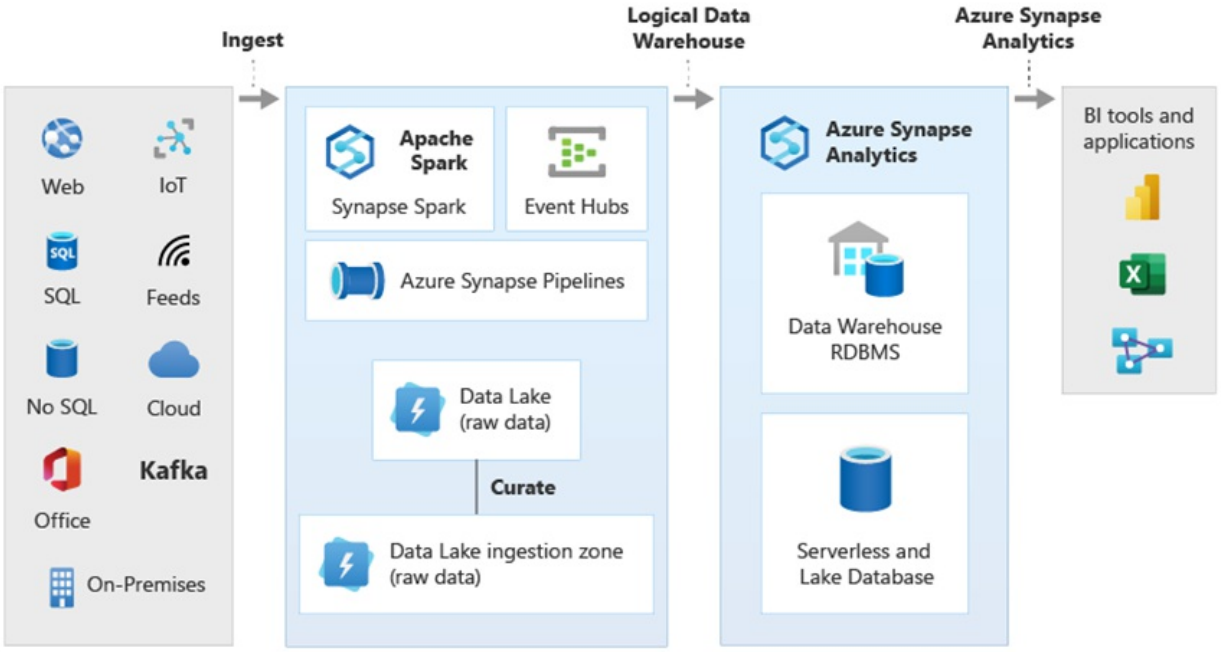
As a result, new more complex kinds of analysis have emerged, such as natural language processing, graph analysis, deep learning, streaming analytics, or complex analysis of large volumes of structured data. These kinds of analysis typically don't happen in a data warehouse, so it's not surprising to see different analytical platforms for different types of analytical workloads, as shown in the following diagram.



> **TIP**
> The ability to make data in multiple analytical data stores look like it's all in one system and join it to Azure Synapse is known as a logical data warehouse architecture.

Because these platforms produce new insights, it's normal to see a requirement to combine the new insights with what you already know in Azure Synapse, which is what PolyBase makes possible.

By using PolyBase data virtualization inside Azure Synapse, you can implement a logical data warehouse where data in Azure Synapse is joined to data in other Azure and on-premises analytical data stores like HDInsight, Azure Cosmos DB, or streaming data flowing into Data Lake Storage from Stream Analytics or Event Hubs. This approach lowers the complexity for users, who access external tables in Azure Synapse and don't need to know that the data they're accessing is stored in multiple underlying analytical systems. The following diagram shows a complex data warehouse structure accessed through comparatively simpler yet still powerful UI methods.



The diagram shows how other technologies in the Microsoft analytical ecosystem can be combined with the capability of the logical data warehouse architecture in Azure Synapse. For example, you can ingest data into Data Lake Storage and curate the data using Data Factory to create trusted data products that represent Microsoft lake database logical data entities. This trusted, commonly understood data can then be consumed and reused in different analytical environments such as Azure Synapse, Azure Synapse Spark pool notebooks, or Azure Cosmos DB. All insights produced in these environments are accessible via a logical data warehouse data virtualization layer made possible by PolyBase.

> **TIP**
> A logical data warehouse architecture simplifies business user access to data and adds new value to what you already know in your data warehouse.

Conclusions

After you migrate your data warehouse to Azure Synapse, you can take advantage of other technologies in the Microsoft analytical ecosystem. By doing so, you not only modernize your data warehouse, but bring insights produced in other Azure analytical data stores into an integrated analytical architecture.

You can broaden your ETL processing to ingest data of any type into Data Lake Storage, and then prepare and integrate the data at scale using Data Factory to produce trusted, commonly understood data assets. Those assets can be consumed by your data warehouse and accessed by data scientists and other applications. You can build real-time and batch oriented analytical pipelines and create machine learning models to run in batch, in real-time on streaming data, and on-demand as a service.

You can use PolyBase or `COPY INTO` to go beyond your data warehouse to simplify access to insights from multiple underlying analytical platforms on Azure. To do so, create holistic integrated views in a logical data warehouse that support access to streaming, big data, and traditional data warehouse insights from BI tools and applications.

By migrating your data warehouse to Azure Synapse, you can take advantage of the rich Microsoft analytical ecosystem running on Azure to drive new value in your business.

Next steps

To learn about migrating to a dedicated SQL pool, see Migrate a data warehouse to a dedicated SQL pool in

Azure Synapse Analytics.

Design and performance for Netezza migrations

8/19/2022 • 19 minutes to read • Edit Online

This article is part one of a seven-part series that provides guidance on how to migrate from Netezza to Azure Synapse Analytics. The focus of this article is best practices for design and performance.

Overview

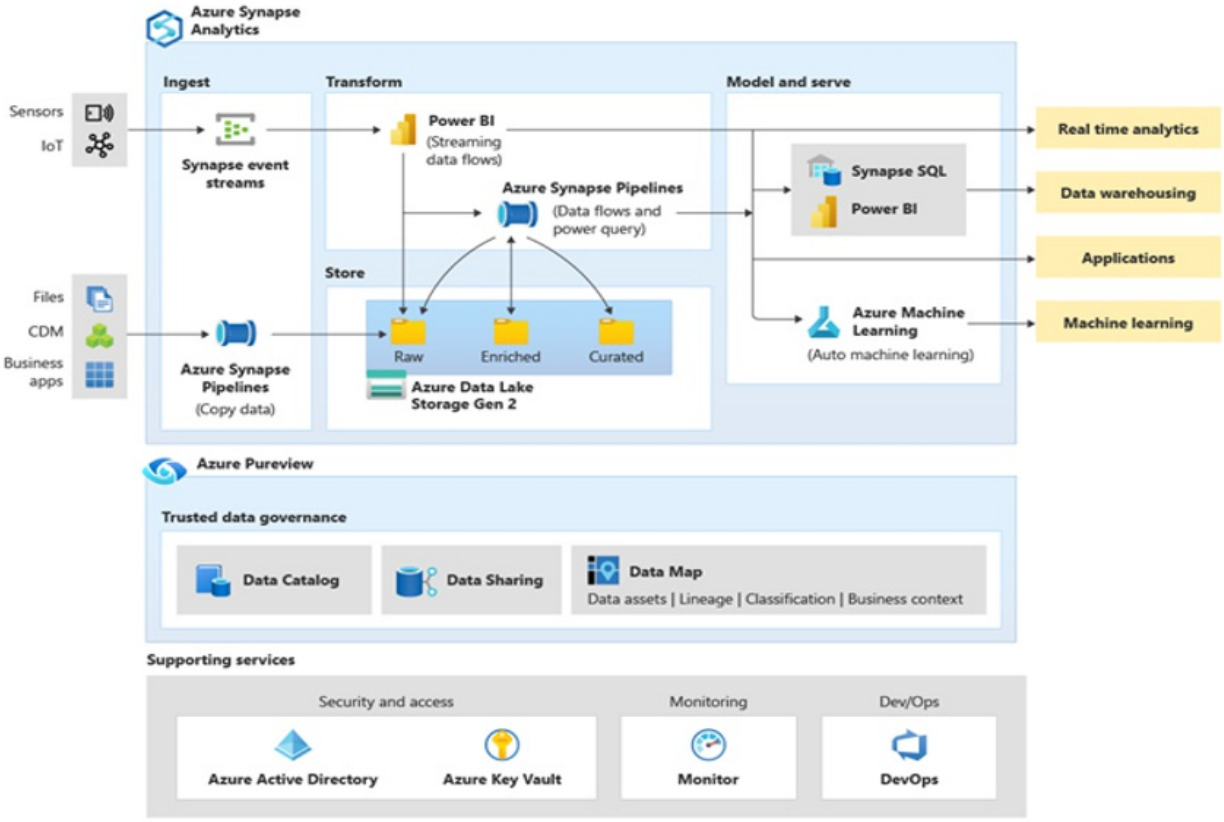
Due to end of support from IBM, many existing users of Netezza data warehouse systems want to take advantage of the innovations provided by modern cloud environments. Infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS) cloud environments let you delegate tasks like infrastructure maintenance and platform development to the cloud provider.

> **TIP**
> More than just a database—the Azure environment includes a comprehensive set of capabilities and tools.

Although Netezza and Azure Synapse Analytics are both SQL databases that use massively parallel processing (MPP) techniques to achieve high query performance on exceptionally large data volumes, there are some basic differences in approach:

- Legacy Netezza systems are often installed on-premises and use proprietary hardware, while Azure Synapse is cloud-based and uses Azure storage and compute resources.
- Upgrading a Netezza configuration is a major task involving extra physical hardware and potentially lengthy database reconfiguration, or dump and reload. Because storage and compute resources are separate in the Azure environment and have elastic scaling capability, those resources can be scaled upwards or downwards independently.
- You can pause or resize Azure Synapse as needed to reduce resource utilization and cost.

Microsoft Azure is a globally available, highly secure, scalable cloud environment that includes Azure Synapse and an ecosystem of supporting tools and capabilities. The next diagram summarizes the Azure Synapse ecosystem.



Azure Synapse provides best-of-breed relational database performance by using techniques such as MPP and multiple levels of automated caching for frequently used data. You can see the results of these techniques in independent benchmarks such as the one run recently by GigaOm, which compares Azure Synapse to other popular cloud data warehouse offerings. Customers who migrate to the Azure Synapse environment see many benefits, including:

- Improved performance and price/performance.
- Increased agility and shorter time to value.
- Faster server deployment and application development.
- Elastic scalability—only pay for actual usage.
- Improved security/compliance.
- Reduced storage and disaster recovery costs.
- Lower overall TCO, better cost control, and streamlined operational expenditure (OPEX).

To maximize these benefits, migrate new or existing data and applications to the Azure Synapse platform. In many organizations, migration includes moving an existing data warehouse from a legacy on-premises platform, such as Netezza, to Azure Synapse. At a high level, the migration process includes these steps:

Preparation

- Define scope—what is to be migrated.
- Build inventory of data and processes for migration.
- Define data model changes (if any).
- Define source data extract mechanism.
- Identify the appropriate Azure and third-party tools and features to be used.
- Train staff early on the new platform.

- Set up the Azure target platform.

Migration □

- Start small and simple.
- Automate wherever possible.
- Leverage Azure built-in tools and features to reduce migration effort.
- Migrate metadata for tables and views.
- Migrate historical data to be maintained.
- Migrate or refactor stored procedures and business processes.
- Migrate or refactor ETL/ELT incremental load processes.

Post migration

- Monitor and document all stages of the process.
- Use the experience gained to build a template for future migrations.
- Re-engineer the data model if required (using new platform performance and scalability).
- Test applications and query tools.
- Benchmark and optimize query performance.

This article provides general information and guidelines for performance optimization when migrating a data warehouse from an existing Netezza environment to Azure Synapse. The goal of performance optimization is to achieve the same or better data warehouse performance in Azure Synapse after schema migration.

Design considerations

Migration scope

When you're preparing to migrate from a Netezza environment, consider the following migration choices.

Choose the workload for the initial migration

Typically, legacy Netezza environments have evolved over time to encompass multiple subject areas and mixed workloads. When you're deciding where to start on a migration project, choose an area where you'll be able to:

- Prove the viability of migrating to Azure Synapse by quickly delivering the benefits of the new environment.
- Allow your in-house technical staff to gain relevant experience with the processes and tools that they'll use when they migrate other areas.
- Create a template for further migrations that's specific to the source Netezza environment and the current tools and processes that are already in place.

A good candidate for an initial migration from a Netezza environment supports the preceding items, and:

- Implements a BI/Analytics workload rather than an online transaction processing (OLTP) workload.
- Has a data model, such as a star or snowflake schema, that can be migrated with minimal modification.

> **TIP**
> Create an inventory of objects that need to be migrated, and document the migration process.

The volume of migrated data in an initial migration should be large enough to demonstrate the capabilities and benefits of the Azure Synapse environment but not too large to quickly demonstrate value. A size in the 1-10 terabyte range is typical.

For your initial migration project, minimize the risk, effort, and migration time so you can quickly see the benefits of the Azure cloud environment. Both the lift-and-shift and phased migration approaches limit the scope of the initial migration to just the data marts and don't address broader migration aspects such as ETL migration and historical data migration. However, you can address those aspects in later phases of the project once the migrated data mart layer is backfilled with data and the required build processes.

Lift and shift migration vs. Phased approach

In general, there are two types of migration regardless of the purpose and scope of the planned migration: lift and shift as-is and a phased approach that incorporates changes.

Lift and shift

In a lift and shift migration, an existing data model, like a star schema, is migrated unchanged to the new Azure Synapse platform. This approach minimizes risk and migration time by reducing the work needed to realize the benefits of moving to the Azure cloud environment. Lift and shift migration is a good fit for these scenarios:

- You have an existing Netezza environment with a single data mart to migrate, or
- You have an existing Netezza environment with data that's already in a well-designed star or snowflake schema, or
- You're under time and cost pressures to move to a modern cloud environment.

> **TIP**
> Lift and shift is a good starting point, even if subsequent phases implement changes to the data model.

Phased approach that incorporates changes

If a legacy data warehouse has evolved over a long period of time, you might need to re-engineer it to maintain the required performance levels. You might also have to re-engineer to support new data like Internet of Things (IoT) streams. As part of the re-engineering process, migrate to Azure Synapse to get the benefits of a scalable cloud environment. Migration can also include a change in the underlying data model, such as a move from an Inmon model to a data vault.

Microsoft recommends moving your existing data model as-is to Azure and using the performance and flexibility of the Azure environment to apply the re-engineering changes. That way, you can use Azure's capabilities to make the changes without impacting the existing source system.

Use Azure Data Factory to implement a metadata-driven migration

You can automate and orchestrate the migration process by using the capabilities of the Azure environment. This approach minimizes the performance hit on the existing Netezza environment, which may already be running close to capacity.

Azure Data Factory is a cloud-based data integration service that supports creating data-driven workflows in the cloud that orchestrate and automate data movement and data transformation. You can use Data Factory to create and schedule data-driven workflows (pipelines) that ingest data from disparate data stores. Data Factory can process and transform data by using compute services such as Azure HDInsight Hadoop, Spark, Azure Data Lake Analytics, and Azure Machine Learning.

When you're planning to use Data Factory facilities to manage the migration process, create metadata that lists all the data tables to be migrated and their location.

Design differences between Netezza and Azure Synapse

As mentioned earlier, there are some basic differences in approach between Netezza and Azure Synapse

Analytics databases and these differences are discussed next.

Multiple databases vs. a single database and schemas

The Netezza environment often contains multiple separate databases. For instance, there could be separate databases for: data ingestion and staging tables, core warehouse tables, and data marts (sometimes referred to as the semantic layer). ETL or ELT pipeline processes might implement cross-database joins and move data between the separate databases.

In contrast, the Azure Synapse environment contains a single database and uses schemas to separate tables into logically separate groups. We recommend that you use a series of schemas within the target Azure Synapse database to mimic the separate databases migrated from the Netezza environment. If the Netezza environment already uses schemas, you may need to use a new naming convention when you move the existing Netezza tables and views to the new environment. For example, you could concatenate the existing Netezza schema and table names into the new Azure Synapse table name, and use schema names in the new environment to maintain the original separate database names. If schema consolidation naming has dots, Azure Synapse Spark might have issues. Although you can use SQL views on top of the underlying tables to maintain the logical structures, there are potential downsides to that approach:

- Views in Azure Synapse are read-only, so any updates to the data must take place on the underlying base tables.
- There may already be one or more layers of views in existence and adding an extra layer of views could affect performance and supportability because nested views are difficult to troubleshoot.

> **TIP**
> Combine multiple databases into a single database within Azure Synapse and use schema names to logically separate the tables.

Table considerations

When you migrate tables between different environments, typically only the raw data and the metadata that describes it physically migrate. Other database elements from the source system, such as indexes, usually aren't migrated because they might be unnecessary or implemented differently in the new environment.

Performance optimizations in the source environment, such as indexes, indicate where you might add performance optimization in the new environment. For example, if queries in the source Netezza environment frequently use zone maps, that suggests that a non-clustered index should be created within Azure Synapse. Other native performance optimization techniques like table replication may be more applicable than straight like-for-like index creation.

> **TIP**
> Existing indexes indicate candidates for indexing in the migrated warehouse.

Unsupported Netezza database object types

Netezza-specific features can often be replaced by Azure Synapse features. However, some Netezza database objects aren't directly supported in Azure Synapse. The following list of unsupported Netezza database objects describes how you can achieve an equivalent functionality in Azure Synapse.

- **Zone maps**: in Netezza, zone maps are automatically created and maintained for the following column types and are used at query time to restrict the amount of data to be scanned:
  - `INTEGER` columns of length 8 bytes or less.
  - Temporal columns, such as `DATE`, `TIME`, and `TIMESTAMP`.

- `CHAR` columns if they're part of a materialized view and mentioned in the `ORDER BY` clause.

  You can find out which columns have zone maps by using the `nz_zonemap` utility, which is part of the NZ Toolkit. Azure Synapse doesn't include zone maps, but you can achieve similar results by using other user-defined index types and/or partitioning.

- **Clustered base tables (CBT)**: in Netezza, CBTs are commonly used for fact tables, which can have billions of records. Scanning such a huge table requires considerable processing time because a full table scan might be needed to get the relevant records. Organizing records on restrictive CBTs allows Netezza to group records in same or nearby extents. This process also creates zone maps that improve the performance by reducing the amount of data that needs to be scanned.

  In Azure Synapse, you can achieve a similar effect by partitioning and/or using other indexes.

- **Materialized views**: Netezza supports materialized views and recommends using one or more materialized views for large tables with many columns if only a few columns are regularly used in queries. Materialized views are automatically refreshed by the system when data in the base table is updated.

  Azure Synapse supports materialized views, with the same functionality as Netezza.

Netezza data type mapping

Most Netezza data types have a direct equivalent in Azure Synapse. The following table shows the recommended approach for mapping Netezza data types to Azure Synapse.

NETEZZA DATA TYPE	AZURE SYNAPSE DATA TYPE
BIGINT	BIGINT
BINARY VARYING(n)	VARBINARY(n)
BOOLEAN	BIT
BYTEINT	TINYINT
CHARACTER VARYING(n)	VARCHAR(n)
CHARACTER(n)	CHAR(n)
DATE	DATE(date)
DECIMAL(p,s)	DECIMAL(p,s)
DOUBLE PRECISION	FLOAT
FLOAT(n)	FLOAT(n)
INTEGER	INT
INTERVAL	INTERVAL data types aren't currently directly supported in Azure Synapse, but can be calculated using temporal functions such as DATEDIFF.
MONEY	MONEY

NETEZZA DATA TYPE	AZURE SYNAPSE DATA TYPE
NATIONAL CHARACTER VARYING(n)	NVARCHAR(n)
NATIONAL CHARACTER(n)	NCHAR(n)
NUMERIC(p,s)	NUMERIC(p,s)
REAL	REAL
SMALLINT	SMALLINT
ST_GEOMETRY(n)	Spatial data types such as ST_GEOMETRY aren't currently supported in Azure Synapse, but the data could be stored as VARCHAR or VARBINARY.
TIME	TIME
TIME WITH TIME ZONE	DATETIMEOFFSET
TIMESTAMP	DATETIME

> **TIP**
> Assess the number and type of unsupported data types during the migration preparation phase.

Third-party vendors offer tools and services to automate migration, including the mapping of data types. If a third-party ETL tool is already in use in the Netezza environment, use that tool to implement any required data transformations.

SQL DML syntax differences

SQL DML syntax differences exist between Netezza SQL and Azure Synapse T-SQL. Those differences are discussed in detail in Minimize SQL issues for Netezza migrations.

- `STRPOS`: in Netezza, the `STRPOS` function returns the position of a substring within a string. The equivalent function in Azure Synapse is `CHARINDEX` with the order of the arguments reversed. For example, `SELECT STRPOS('abcdef','def')...` in Netezza is equivalent to `SELECT CHARINDEX('def','abcdef')...` in Azure Synapse.
- `AGE`: Netezza supports the `AGE` operator to give the interval between two temporal values, such as timestamps or dates, for example: `SELECT AGE('23-03-1956','01-01-2019') FROM...`. In Azure Synapse, use `DATEDIFF` to get the interval, for example: `SELECT DATEDIFF(day, '1956-03-26','2019-01-01') FROM...`. Note the date representation sequence.
- `NOW()`: Netezza uses `NOW()` to represent `CURRENT_TIMESTAMP` in Azure Synapse.

Functions, stored procedures, and sequences

When migrating a data warehouse from a mature environment like Netezza, you probably need to migrate elements other than simple tables and views. Check whether tools within the Azure environment can replace the functionality of functions, stored procedures, and sequences because it's usually more efficient to use built-in Azure tools than to recode those elements for Azure Synapse.

As part of your preparation phase, create an inventory of objects that need to be migrated, define a method for handling them, and allocate appropriate resources in your migration plan.

Data integration partners offer tools and services that can automate the migration of functions, stored procedures, and sequences.

The following sections further discuss the migration of functions, stored procedures, and sequences.

Functions

As with most database products, Netezza supports system and user-defined functions within a SQL implementation. When you migrate a legacy database platform to Azure Synapse, common system functions can usually be migrated without change. Some system functions might have a slightly different syntax, but any required changes can be automated.

For Netezza system functions or arbitrary user-defined functions that have no equivalent in Azure Synapse, recode those functions using a target environment language. Netezza user-defined functions are coded in nzlua or C++ languages. Azure Synapse uses the Transact-SQL language to implement user-defined functions.

Stored procedures

Most modern database products support storing procedures within the database. Netezza provides the NZPLSQL language, which is based on Postgres PL/pgSQL, for this purpose. A stored procedure typically contains both SQL statements and procedural logic, and returns data or a status.

Azure Synapse supports stored procedures using T-SQL, so you need to recode any migrated stored procedures in that language.

Sequences

In Netezza, a sequence is a named database object created using `CREATE SEQUENCE`. A sequence provides unique numeric values via the `NEXT VALUE FOR` method. You can use the generated unique numbers as surrogate key values for primary keys.

Azure Synapse doesn't implement `CREATE SEQUENCE`, but you can implement sequences using IDENTITY columns or SQL code that generates the next sequence number in a series.

Extract metadata and data from a Netezza environment

Data Definition Language (DDL) generation

The ANSI SQL standard defines the basic syntax for Data Definition Language (DDL) commands. Some DDL commands, such as `CREATE TABLE` and `CREATE VIEW`, are common to both Netezza and Azure Synapse but have been extended to provide implementation-specific features.

You can edit existing Netezza `CREATE TABLE` and `CREATE VIEW` scripts to achieve equivalent definitions in Azure Synapse. To do so, you might need to use modified data types and remove or modify Netezza-specific clauses such as `ORGANIZE ON`.

Within the Netezza environment, system catalog tables specify the current table and view definition. Unlike user-maintained documentation, system catalog information is always complete and in sync with current table definitions. By using utilities such as `nz_ddl_table`, you can access system catalog information to generate `CREATE TABLE` DDL statements that create equivalent tables in Azure Synapse.

You can also use third-party migration and ETL tools that process system catalog information to achieve similar results.

Data extraction from Netezza

You can extract raw table data from Netezza tables to flat delimited files, such as CSV files, using standard Netezza utilities like nzsql and nzunload, or via external tables. Then, you can compress the flat delimited files using gzip, and upload the compressed files to Azure Blob Storage using AzCopy or Azure data transport tools like Azure Data Box.

Extract table data as efficiently as possible. Use the external tables approach because it's the fastest extract method. Perform multiple extracts in parallel to maximize data extraction throughput. The following SQL statement performs an external table extract:

```
CREATE EXTERNAL TABLE '/tmp/export_tab1.csv' USING (DELIM ',') AS SELECT * from <TABLENAME>;
```

If sufficient network bandwidth is available, you can extract data from an on-premises Netezza system directly into Azure Synapse tables or Azure Blob Data Storage. To do so, use Data Factory processes or third-party data migration or ETL products.

> **TIP**
> Use Netezza external tables for the most efficient data extraction.

Extracted data files should contain delimited text in CSV, Optimized Row Columnar (ORC), or Parquet format.

For more information about migrating data and ETL from a Netezza environment, see Data migration, ETL, and load for Netezza migrations.

Performance recommendations for Netezza migrations

The goal of performance optimization is same or better data warehouse performance after migration to Azure Synapse.

Similarities in performance tuning approach concepts

Many performance tuning concepts for Netezza databases hold true for Azure Synapse databases. For example:

- Use data distribution to collocate data-to-be-joined onto the same processing node.
- Use the smallest data type for a given column to save storage space and accelerate query processing.
- Ensure that columns to be joined have the same data type in order to optimize join processing and reduce the need for data transforms.
- To help the optimizer produce the best execution plan, ensure statistics are up to date.
- Monitor performance using built-in database capabilities to ensure that resources are being used efficiently.

> **TIP**
> Prioritize familiarity with the tuning options in Azure Synapse at the start of a migration.

Differences in performance tuning approach

This section highlights low-level performance tuning implementation differences between Netezza and Azure Synapse.

Data distribution options

For performance, Azure Synapse was designed with multi-node architecture and uses parallel processing. To optimize table performance, you can define a data distribution option in `CREATE TABLE` statements using `DISTRIBUTION` in Azure Synapse and `DISTRIBUTE ON` in Netezza.

Unlike Netezza, Azure Synapse supports local joins between a small table and a large table through small table replication. For instance, consider a small dimension table and a large fact table within a star schema model. Azure Synapse can replicate the smaller dimension table across all nodes to ensure that the value of any join key for the large table has a matching, locally available dimension row. The overhead of dimension table replication is relatively low for a small dimension table. For large dimension tables, a hash distribution approach is more appropriate. For more information about data distribution options, see Design guidance for using replicated tables and Guidance for designing distributed tables.

Data indexing

Azure Synapse supports several user-definable indexing options that have a different operation and usage compared to system-managed zone maps in Netezza. For more information about the different indexing options in Azure Synapse, see Indexes on dedicated SQL pool tables.

The existing system-managed zone maps within a source Netezza environment provide a useful indication of data usage and the candidate columns for indexing in the Azure Synapse environment.

Data partitioning

In an enterprise data warehouse, fact tables can contain billions of rows. Partitioning optimizes the maintenance and query performance of these tables by splitting them into separate parts to reduce the amount of data processed. In Azure Synapse, the `CREATE TABLE` statement defines the partitioning specification for a table.

You can only use one field per table for partitioning. That field is often a date field because many queries are filtered by date or date range. It's possible to change the partitioning of a table after initial load by using the `CREATE TABLE AS` (CTAS) statement to recreate the table with a new distribution. For a detailed discussion of partitioning in Azure Synapse, see Partitioning tables in dedicated SQL pool.

Data table statistics

You should ensure that statistics on data tables are up to date by building in a statistics step to ETL/ELT jobs.

PolyBase or COPY INTO for data loading

PolyBase supports efficient loading of large amounts of data to a data warehouse by using parallel loading streams. For more information, see PolyBase data loading strategy.

COPY INTO also supports high-throughput data ingestion, and:

- Data retrieval from all files within a folder and subfolders.
- Data retrieval from multiple locations in the same storage account. You can specify multiple locations by using comma separated paths.
- Azure Data Lake Storage (ADLS) and Azure Blob Storage.
- CSV, PARQUET, and ORC file formats.

Workload management

Running mixed workloads can pose resource challenges on busy systems. A successful workload management scheme effectively manages resources, ensures highly efficient resource utilization, and maximizes return on investment (ROI). Workload classification, workload importance, and workload isolation give more control over how workload utilizes system resources.

The workload management guide describes the techniques to analyze the workload, manage and monitor workload importance, and the steps to convert a resource class to a workload group. Use the Azure portal and T-SQL queries on DMVs to monitor the workload to ensure that the applicable resources are efficiently utilized.

Next steps

To learn about ETL and load for Netezza migration, see the next article in this series: Data migration, ETL, and load for Netezza migrations.

Data migration, ETL, and load for Netezza migrations

8/19/2022 • 17 minutes to read • Edit Online

This article is part two of a seven-part series that provides guidance on how to migrate from Netezza to Azure Synapse Analytics. The focus of this article is best practices for ETL and load migration.

Data migration considerations

Initial decisions for data migration from Netezza

When migrating a Netezza data warehouse, you need to ask some basic data-related questions. For example:

- Should unused table structures be migrated?
- What's the best migration approach to minimize risk and user impact?
- When migrating data marts: stay physical or go virtual?

The next sections discuss these points within the context of migration from Netezza.

Migrate unused tables?

> **TIP**
> In legacy systems, it's not unusual for tables to become redundant over time—these don't need to be migrated in most cases.

It makes sense to only migrate tables that are in use in the existing system. Tables that aren't active can be archived rather than migrated, so that the data is available if necessary in future. It's best to use system metadata and log files rather than documentation to determine which tables are in use, because documentation can be out of date.

If enabled, Netezza query history tables contain information that can determine when a given table was last accessed—which can in turn be used to decide whether a table is a candidate for migration.

Here's an example query that looks for the usage of a specific table within a given time window:

```
SELECT FORMAT_TABLE_ACCESS (usage),
  hq.submittime
FROM "$v_hist_queries" hq
  INNER JOIN "$hist_table_access_3" hta USING
(NPSID, NPSINSTANCEID, OPID, SESSIONID)
WHERE hq.dbname = 'PROD'
AND hta.schemaname = 'ADMIN'
AND hta.tablename = 'TEST_1'
AND hq.SUBMITTIME > '01-01-2015'
AND hq.SUBMITTIME <= '08-06-2015'
AND
(
  instr(FORMAT_TABLE_ACCESS(usage),'ins') > 0
  OR instr(FORMAT_TABLE_ACCESS(usage),'upd') > 0
  OR instr(FORMAT_TABLE_ACCESS(usage),'del') > 0
)
AND status=0;
```

```
| FORMAT_TABLE_ACCESS | SUBMITTIME
---------------------+--------------------------
ins                   | 2015-06-16 18:32:25.728042
ins                   | 2015-06-16 17:46:14.337105
ins                   | 2015-06-16 17:47:14.430995
(3 rows)
```

This query uses the helper function `FORMAT_TABLE_ACCESS` and the digit at the end of the `$v_hist_table_access_3` view to match the installed query history version.

What is the best migration approach to minimize risk and impact on users?

This question comes up frequently because companies may want to lower the impact of changes on the data warehouse data model to improve agility. Companies often see an opportunity to further modernize or transform their data during an ETL migration. This approach carries a higher risk because it changes multiple factors simultaneously, making it difficult to compare the outcomes of the old system versus the new. Making data model changes here could also affect upstream or downstream ETL jobs to other systems. Because of that risk, it's better to redesign on this scale after the data warehouse migration.

Even if a data model is intentionally changed as part of the overall migration, it's good practice to migrate the existing model as-is to Azure Synapse, rather than do any re-engineering on the new platform. This approach minimizes the effect on existing production systems, while benefiting from the performance and elastic scalability of the Azure platform for one-off re-engineering tasks.

When migrating from Netezza, often the existing data model is already suitable for as-is migration to Azure Synapse.

> **TIP**
> Migrate the existing model as-is initially, even if a change to the data model is planned in the future.

Migrate data marts: stay physical or go virtual?

> **TIP**
> Virtualizing data marts can save on storage and processing resources.

In legacy Netezza data warehouse environments, it's common practice to create several data marts that are structured to provide good performance for ad hoc self-service queries and reports for a given department or business function within an organization. As such, a data mart typically consists of a subset of the data warehouse and contains aggregated versions of the data in a form that enables users to easily query that data with fast response times via user-friendly query tools such as Microsoft Power BI, Tableau, or MicroStrategy. This form is typically a dimensional data model. One use of data marts is to expose the data in a usable form, even if the underlying warehouse data model is something different, such as a data vault.

You can use separate data marts for individual business units within an organization to implement robust data security regimes, by only allowing users to access specific data marts that are relevant to them, and eliminating, obfuscating, or anonymizing sensitive data.

If these data marts are implemented as physical tables, they'll require additional storage resources to store them, and additional processing to build and refresh them regularly. Also, the data in the mart will only be as up to date as the last refresh operation, and so may be unsuitable for highly volatile data dashboards.

> **TIP**
> The performance and scalability of Azure Synapse enables virtualization without sacrificing performance.

With the advent of relatively low-cost scalable MPP architectures, such as Azure Synapse, and the inherent performance characteristics of such architectures, it may be that you can provide data mart functionality without having to instantiate the mart as a set of physical tables. This is achieved by effectively virtualizing the data marts via SQL views onto the main data warehouse, or via a virtualization layer using features such as views in Azure or the visualization products of Microsoft partners. This approach simplifies or eliminates the need for additional storage and aggregation processing and reduces the overall number of database objects to be migrated.

There's another potential benefit to this approach. By implementing the aggregation and join logic within a virtualization layer, and presenting external reporting tools via a virtualized view, the processing required to create these views is "pushed down" into the data warehouse, which is generally the best place to run joins, aggregations, and other related operations on large data volumes.

The primary drivers for choosing a virtual data mart implementation over a physical data mart are:

- More agility: a virtual data mart is easier to change than physical tables and the associated ETL processes.
- Lower total cost of ownership: a virtualized implementation requires fewer data stores and copies of data.
- Elimination of ETL jobs to migrate and simplify data warehouse architecture in a virtualized environment.
- Performance: although physical data marts have historically been more performant, virtualization products now implement intelligent caching techniques to mitigate.

Data migration from Netezza

Understand your data

Part of migration planning is understanding in detail the volume of data that needs to be migrated since that can impact decisions about the migration approach. Use system metadata to determine the physical space taken up by the "raw data" within the tables to be migrated. In this context, "raw data" means the amount of space used by the data rows within a table, excluding overheads such as indexes and compression. This is especially true for the largest fact tables since these will typically comprise more than 95% of the data.

You can get an accurate number for the volume of data to be migrated for a given table by extracting a representative sample of the data—for example, one million rows—to an uncompressed delimited flat ASCII data file. Then, use the size of that file to get an average raw data size per row of that table. Finally, multiply that average size by the total number of rows in the full table to give a raw data size for the table. Use that raw data size in your planning.

Netezza data type mapping

> **TIP**
> Assess the impact of unsupported data types as part of the preparation phase.

Most Netezza data types have a direct equivalent in Azure Synapse. The following table shows these data types, together with the recommended approach for mapping them.

NETEZZA DATA TYPE	AZURE SYNAPSE DATA TYPE
BIGINT	BIGINT

NETEZZA DATA TYPE	AZURE SYNAPSE DATA TYPE
BINARY VARYING(n)	VARBINARY(n)
BOOLEAN	BIT
BYTEINT	TINYINT
CHARACTER VARYING(n)	VARCHAR(n)
CHARACTER(n)	CHAR(n)
DATE	DATE(date)
DECIMAL(p,s)	DECIMAL(p,s)
DOUBLE PRECISION	FLOAT
FLOAT(n)	FLOAT(n)
INTEGER	INT
INTERVAL	INTERVAL data types aren't currently directly supported in Azure Synapse Analytics, but can be calculated using temporal functions, such as DATEDIFF.
MONEY	MONEY
NATIONAL CHARACTER VARYING(n)	NVARCHAR(n)
NATIONAL CHARACTER(n)	NCHAR(n)
NUMERIC(p,s)	NUMERIC(p,s)
REAL	REAL
SMALLINT	SMALLINT
ST_GEOMETRY(n)	Spatial data types such as ST_GEOMETRY aren't currently supported in Azure Synapse Analytics, but the data could be stored as VARCHAR or VARBINARY.
TIME	TIME
TIME WITH TIME ZONE	DATETIMEOFFSET
TIMESTAMP	DATETIME

Use the metadata from the Netezza catalog tables to determine whether any of these data types need to be migrated, and allow for this in your migration plan. The important metadata views in Netezza for this type of query are:

- `_V_USER`: the user view gives information about the users in the Netezza system.

- `_V_TABLE`: the table view contains the list of tables created in the Netezza performance system.
- `_V_RELATION_COLUMN`: the relation column system catalog view contains the columns available in a table.
- `_V_OBJECTS`: the objects view lists the different objects like tables, view, functions, and so on, that are available in Netezza.

For example, this Netezza SQL query shows columns and column types:

```
SELECT
tablename,
  attname AS COL_NAME,
  b.FORMAT_TYPE AS COL_TYPE,
  attnum AS COL_NUM
FROM _v_table a
  JOIN _v_relation_column b
  ON a.objid = b.objid
WHERE a.tablename = 'ATT_TEST'
AND a.schema = 'ADMIN'
ORDER BY attnum;
```

```
TABLENAME | COL_NAME    | COL_TYPE             | COL_NUM
----------+-------------+----------------------+--------
ATT_TEST  | COL_INT     | INTEGER              | 1
ATT_TEST  | COL_NUMERIC | NUMERIC(10,2)        | 2
ATT_TEST  | COL_VARCHAR | CHARACTER VARYING(5) | 3
ATT_TEST  | COL_DATE    | DATE                 | 4
(4 rows)
```

The query can be modified to search all tables for any occurrences of unsupported data types.

Azure Data Factory can be used to move data from a legacy Netezza environment. For more information, see IBM Netezza connector.

Third-party vendors offer tools and services to automate migration, including the mapping of data types as previously described. Also, third-party ETL tools, like Informatica or Talend, already in use in the Netezza environment can implement all required data transformations. The next section explores the migration of existing third-party ETL processes.

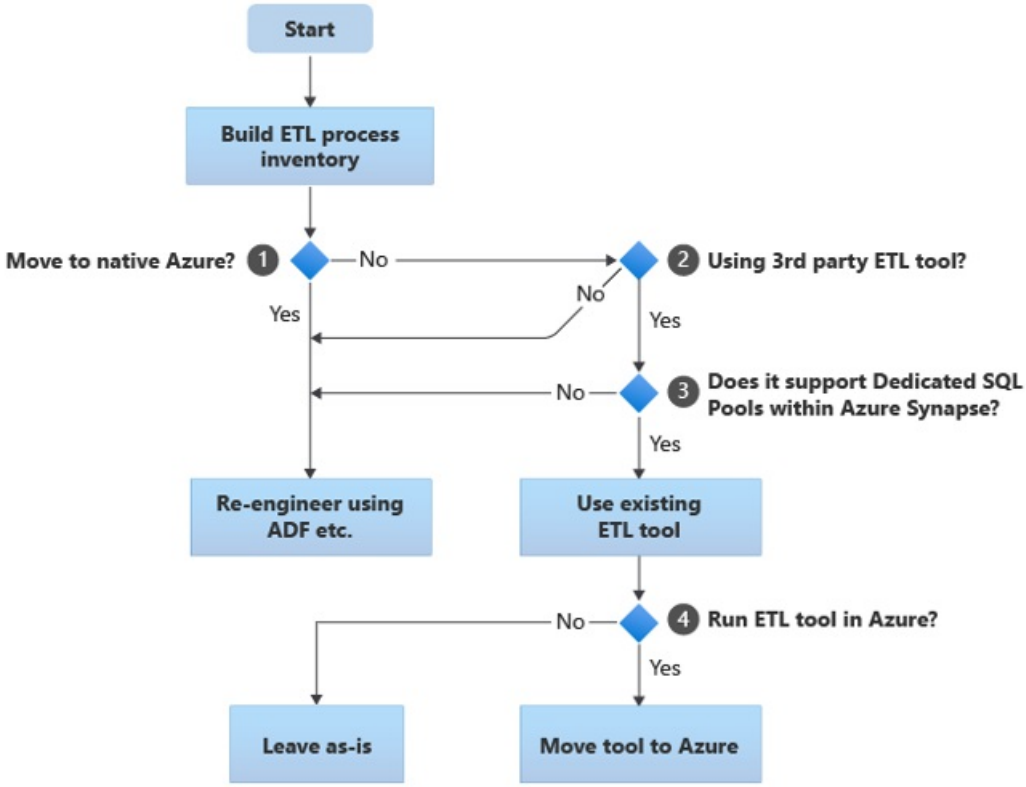
ETL migration considerations

Initial decisions regarding Netezza ETL migration

> **TIP**
> Plan the approach to ETL migration ahead of time and leverage Azure facilities where appropriate.

For ETL/ELT processing, legacy Netezza data warehouses may use custom-built scripts using Netezza utilities such as nzsql and nzload, or third-party ETL tools such as Informatica or Ab Initio. Sometimes, Netezza data warehouses use a combination of ETL and ELT approaches that's evolved over time. When planning a migration to Azure Synapse, you need to determine the best way to implement the required ETL/ELT processing in the new environment, while minimizing the cost and risk involved. To learn more about ETL and ELT processing, see ELT vs ETL design approach.

The following sections discuss migration options and make recommendations for various use cases. This flowchart summarizes one approach:



The first step is always to build an inventory of ETL/ELT processes that need to be migrated. As with other steps, it's possible that the standard "built-in" Azure features make it unnecessary to migrate some existing processes. For planning purposes, it's important to understand the scale of the migration to be performed.

In the preceding flowchart, decision 1 relates to a high-level decision about whether to migrate to a totally Azure-native environment. If you're moving to a totally Azure-native environment, we recommend that you re-engineer the ETL processing using Pipelines and activities in Azure Data Factory or Azure Synapse Pipelines. If you're not moving to a totally Azure-native environment, then decision 2 is whether an existing third-party ETL tool is already in use.

> **TIP**
> Leverage investment in existing third-party tools to reduce cost and risk.

If a third-party ETL tool is already in use, and especially if there's a large investment in skills or several existing workflows and schedules use that tool, then decision 3 is whether the tool can efficiently support Azure Synapse as a target environment. Ideally, the tool will include "native" connectors that can leverage Azure facilities like PolyBase or COPY INTO, for the most efficient data loading. There's a way to call an external process, such as PolyBase or `COPY INTO`, and pass in the appropriate parameters. In this case, leverage existing skills and workflows, with Azure Synapse as the new target environment.

If you decide to retain an existing third-party ETL tool, there may be benefits to running that tool within the Azure environment (rather than on an existing on-premises ETL server) and having Azure Data Factory handle the overall orchestration of the existing workflows. One particular benefit is that less data needs to be downloaded from Azure, processed, and then uploaded back into Azure. So, decision 4 is whether to leave the existing tool running as-is or move it into the Azure environment to achieve cost, performance, and scalability benefits.

Re-engineer existing Netezza-specific scripts

If some or all the existing Netezza warehouse ETL/ELT processing is handled by custom scripts that utilize Netezza-specific utilities, such as nzsql or nzload, then these scripts need to be recoded for the new Azure Synapse environment. Similarly, if ETL processes were implemented using stored procedures in Netezza, then

these will also have to be recoded.

> **TIP**
> The inventory of ETL tasks to be migrated should include scripts and stored procedures.

Some elements of the ETL process are easy to migrate, for example by simple bulk data load into a staging table from an external file. It may even be possible to automate those parts of the process, for example, by using PolyBase instead of nzload. Other parts of the process that contain arbitrary complex SQL and/or stored procedures will take more time to re-engineer.

One way of testing Netezza SQL for compatibility with Azure Synapse is to capture some representative SQL statements from Netezza query history, then prefix those queries with `EXPLAIN`, and then—assuming a like-for-like migrated data model in Azure Synapse—run those `EXPLAIN` statements in Azure Synapse. Any incompatible SQL will generate an error, and the error information can determine the scale of the recoding task.

Microsoft partners offer tools and services to migrate Netezza SQL and stored procedures to Azure Synapse.

Use third-party ETL tools

As described in the previous section, in many cases the existing legacy data warehouse system will already be populated and maintained by third-party ETL products. For a list of Microsoft data integration partners for Azure Synapse, see Data integration partners.

Data loading from Netezza

Choices available when loading data from Netezza

> **TIP**
> Third-party tools can simplify and automate the migration process and therefore reduce risk.

When it comes to migrating data from a Netezza data warehouse, there are some basic questions associated with data loading that need to be resolved. You'll need to decide how the data will be physically moved from the existing on-premises Netezza environment into Azure Synapse in the cloud, and which tools will be used to perform the transfer and load. Consider the following questions, which are discussed in the next sections.

- Will you extract the data to files, or move it directly via a network connection?
- Will you orchestrate the process from the source system, or from the Azure target environment?
- Which tools will you use to automate and manage the process?

Transfer data via files or network connection?

> **TIP**
> Understand the data volumes to be migrated and the available network bandwidth since these factors influence the migration approach decision.

Once the database tables to be migrated have been created in Azure Synapse, you can move the data to populate those tables out of the legacy Netezza system and into the new environment. There are two basic approaches:

- **File extract**: extract the data from the Netezza tables to flat files, normally in CSV format, via nzsql with the -o option or via the `CREATE EXTERNAL TABLE` statement. Use an external table whenever possible since it's the most efficient in terms of data throughput. The following SQL example creates a CSV file via an

external table:

```
CREATE EXTERNAL TABLE '/data/export.csv' USING (delimiter ',')
AS SELECT col1, col2, expr1, expr2, col3, col1 || col2 FROM your table;
```

Use an external table if you're exporting data to a mounted file system on a local Netezza host. If you're exporting data to a remote machine that has JDBC, ODBC, or OLEDB installed, then your "remotesource odbc" option is the `USING` clause.

This approach requires space to land the extracted data files. The space could be local to the Netezza source database (if sufficient storage is available), or remote in Azure Blob Storage. The best performance is achieved when a file is written locally, since that avoids network overhead.

To minimize the storage and network transfer requirements, it's good practice to compress the extracted data files using a utility like gzip.

Once extracted, the flat files can either be moved into Azure Blob Storage (collocated with the target Azure Synapse instance), or loaded directly into Azure Synapse using PolyBase or COPY INTO. The method for physically moving data from local on-premises storage to the Azure cloud environment depends on the amount of data and the available network bandwidth.

Microsoft provides various options to move large volumes of data, including AzCopy for moving files across the network into Azure Storage, Azure ExpressRoute for moving bulk data over a private network connection, and Azure Data Box for files moving to a physical storage device that's then shipped to an Azure data center for loading. For more information, see data transfer.

- **Direct extract and load across network**: the target Azure environment sends a data extract request, normally via a SQL command, to the legacy Netezza system to extract the data. The results are sent across the network and loaded directly into Azure Synapse, with no need to land the data into intermediate files. The limiting factor in this scenario is normally the bandwidth of the network connection between the Netezza database and the Azure environment. For very large data volumes, this approach may not be practical.

There's also a hybrid approach that uses both methods. For example, you can use the direct network extract approach for smaller dimension tables and samples of the larger fact tables to quickly provide a test environment in Azure Synapse. For large volume historical fact tables, you can use the file extract and transfer approach using Azure Data Box.

Orchestrate from Netezza or Azure?

The recommended approach when moving to Azure Synapse is to orchestrate the data extract and loading from the Azure environment using Azure Synapse Pipelines or Azure Data Factory, as well as associated utilities, such as PolyBase or COPY INTO, for the most efficient data loading. This approach leverages Azure capabilities and provides an easy method to build reusable data loading pipelines.

Other benefits of this approach include reduced impact on the Netezza system during the data load process since the management and loading process is running in Azure, and the ability to automate the process by using metadata-driven data load pipelines.

Which tools can be used?

The task of data transformation and movement is the basic function of all ETL products. If one of these products is already in use in the existing Netezza environment, then using the existing ETL tool may simplify data migration from Netezza to Azure Synapse. This approach assumes that the ETL tool supports Azure Synapse as a target environment. For more information on tools that support Azure Synapse, see Data integration partners.

If you're using an ETL tool, consider running that tool within the Azure environment to benefit from Azure cloud performance, scalability, and cost, and free up resources in the Netezza data center. Another benefit is reduced

data movement between the cloud and on-premises environments.

Summary

To summarize, our recommendations for migrating data and associated ETL processes from Netezza to Azure Synapse are:

- Plan ahead to ensure a successful migration exercise.
- Build a detailed inventory of data and processes to be migrated as soon as possible.
- Use system metadata and log files to get an accurate understanding of data and process usage. Don't rely on documentation since it may be out of date.
- Understand the data volumes to be migrated, and the network bandwidth between the on-premises data center and Azure cloud environments.
- Leverage standard "built-in" Azure features to minimize the migration workload.
- Identify and understand the most efficient tools for data extraction and loading in both Netezza and Azure environments. Use the appropriate tools in each phase of the process.
- Use Azure facilities, such as Azure Synapse Pipelines or Azure Data Factory, to orchestrate and automate the migration process while minimizing impact on the Netezza system.

Next steps

To learn more about security access operations, see the next article in this series: Security, access, and operations for Netezza migrations.

Security, access, and operations for Netezza migrations

8/19/2022 • 21 minutes to read • Edit Online

This article is part three of a seven-part series that provides guidance on how to migrate from Netezza to Azure Synapse Analytics. The focus of this article is best practices for security access operations.

Security considerations

This article discusses connection methods for existing legacy Netezza environments and how they can be migrated to Azure Synapse Analytics with minimal risk and user impact.

This article assumes that there's a requirement to migrate the existing methods of connection and user/role/permission structure as-is. If not, use the Azure portal to create and manage a new security regime.

For more information on the Azure Synapse security options, see Security whitepaper.

Connection and authentication

> **TIP**
> Authentication in both Netezza and Azure Synapse can be "in database" or through external methods.

Netezza authorization options

The IBM Netezza system offers several authentication methods for Netezza database users:

- **Local authentication**: Netezza administrators define database users and their passwords by using the `CREATE USER` command or through Netezza administrative interfaces. In local authentication, use the Netezza system to manage database accounts and passwords, and to add and remove database users from the system. This method is the default authentication method.
- **LDAP authentication**: use an LDAP name server to authenticate database users, and manage passwords, database account activations, and deactivations. The Netezza system uses a Pluggable Authentication Module (PAM) to authenticate users on the LDAP name server. Microsoft Active Directory conforms to the LDAP protocol, so it can be treated like an LDAP server for the purposes of LDAP authentication.
- **Kerberos authentication**: use a Kerberos distribution server to authenticate database users, and manage passwords, database account activations, and deactivations.

Authentication is a system-wide setting. Users must be either locally authenticated or authenticated by using the LDAP or Kerberos method. If you choose LDAP or Kerberos authentication, create users with local authentication on a per-user basis. LDAP and Kerberos can't be used at the same time to authenticate users. Netezza host supports LDAP or Kerberos authentication for database user logins only, not for operating system logins on the host.

Azure Synapse authorization options

Azure Synapse supports two basic options for connection and authorization:

- **SQL authentication**: SQL authentication is via a database connection that includes a database identifier, user ID, and password plus other optional parameters. This is functionally equivalent to Netezza local connections.

- **Azure Active Directory (Azure AD) authentication**: with Azure AD authentication, you can centrally manage the identities of database users and other Microsoft services in one central location. Central ID management provides a single place to manage Azure Synapse users and simplifies permission management. Azure AD can also support connections to LDAP and Kerberos services—for example, Azure AD can be used to connect to existing LDAP directories if these are to remain in place after migration of the database.

Users, roles, and permissions

Overview

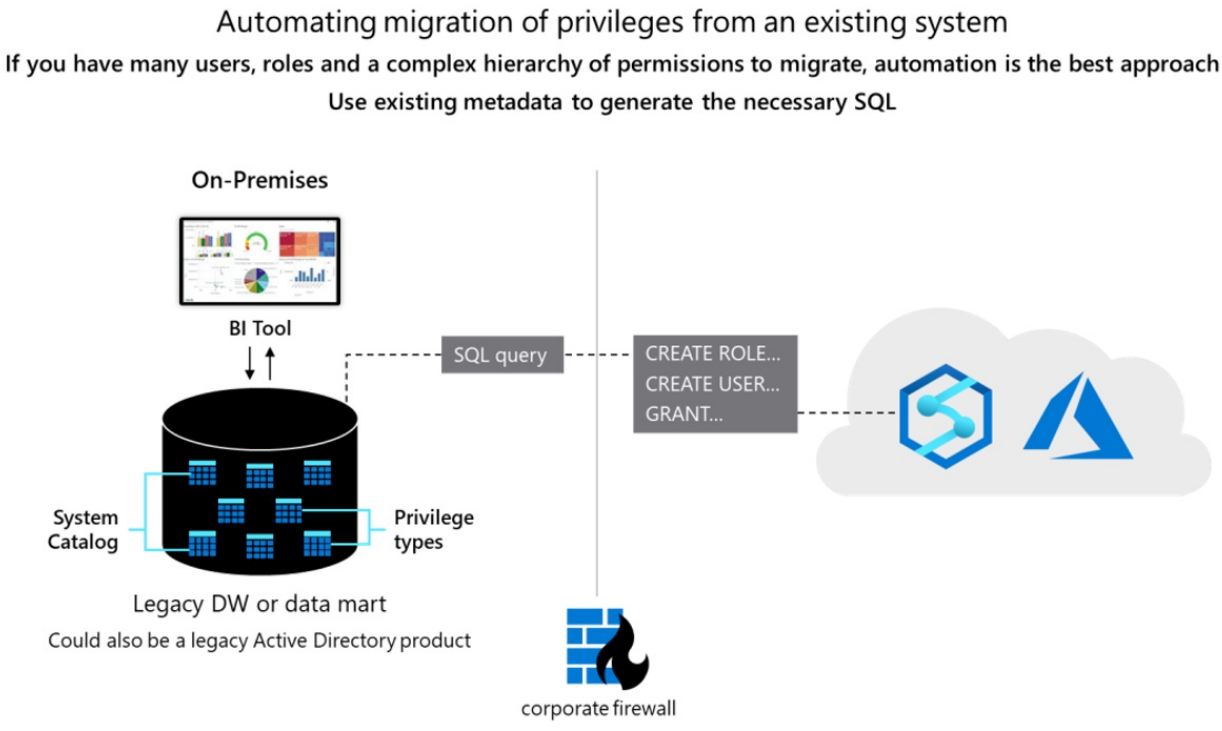
> **TIP**
> High-level planning is essential for a successful migration project.

Both Netezza and Azure Synapse implement database access control via a combination of users, roles (groups in Netezza), and permissions. Both use standard SQL `CREATE USER` and `CREATE ROLE/GROUP` statements to define users and roles, and `GRANT` and `REVOKE` statements to assign or remove permissions to those users and/or roles.

> **TIP**
> Automation of migration processes is recommended to reduce elapsed time and scope for errors.

Conceptually the two databases are similar, and it might be possible to automate the migration of existing user IDs, groups, and permissions to some degree. Migrate such data by extracting the existing legacy user and group information from the Netezza system catalog tables and generating matching equivalent `CREATE USER` and `CREATE ROLE` statements to be run in Azure Synapse to recreate the same user/role hierarchy.

After data extraction, use Netezza system catalog tables to generate equivalent `GRANT` statements to assign permissions (where an equivalent one exists). The following diagram shows how to use existing metadata to generate the necessary SQL.



See the following sections for more details.

Users and roles

> **TIP**
> Migration of a data warehouse requires more than just tables, views, and SQL statements.

The information about current users and groups in a Netezza system is held in system catalog views `_v_users` and `_v_groupusers`. Use the nzsql utility or tools such as the Netezza Performance, NzAdmin, or Netezza Utility scripts to list user privileges. For example, use the `dpu` and `dpgu` commands in nzsql to display users or groups with their permissions.

Use or edit the utility scripts `nz_get_users` and `nz_get_user_groups` to retrieve the same information in the required format.

Query system catalog views directly (if the user has `SELECT` access to those views) to obtain current lists of users and roles defined within the system. See examples to list users, groups, or users and their associated groups:

```
-- List of users
SELECT USERNAME FROM _V_USER;

--List of groups
SELECT DISTINCT(GROUPNAME) FROM _V_USERGROUPS;

--List of users and their associated groups
SELECT USERNAME, GROUPNAME FROM _V_GROUPUSERS;
```

Modify the example `SELECT` statement to produce a result set that is a series of `CREATE USER` and `CREATE GROUP` statements by including the appropriate text as a literal within the `SELECT` statement.

There's no way to retrieve existing passwords, so you need to implement a scheme for allocating new initial passwords on Azure Synapse.

Permissions

> **TIP**
> There are equivalent Azure Synapse permissions for basic database operations such as DML and DDL.

In a Netezza system, the system table `_t_usrobj_priv` holds the access rights for users and roles. Query these tables (if the user has `SELECT` access to those tables) to obtain current lists of access rights defined within the system.

In Netezza, the individual permissions are represented as individual bits within field privileges or g_privileges. See example SQL statement at user group permissions

The simplest way to obtain a DDL script that contains the `GRANT` commands to replicate the current privileges for users and groups is to use the appropriate Netezza utility scripts:

```
--List of group privileges
nz_ddl_grant_group -usrobj dbname > output_file_dbname;

--List of user privileges
nz_ddl_grant_user -usrobj dbname > output_file_dbname;
```

The output file can be modified to produce a script that is a series of `GRANT` statements for Azure Synapse.

Netezza supports two classes of access rights, Admin and Object. See the following tables for a list of Netezza

access rights and their equivalent in Azure Synapse.

ADMIN PRIVILEGE	DESCRIPTION	AZURE SYNAPSE EQUIVALENT
Backup	Allows user to create backups. The user can run backups. The user can run the command `nzbackup`.	[1]
[Create] Aggregate	Allows the user to create user-defined aggregates (UDAs). Permission to operate on existing UDAs is controlled by object privileges.	CREATE FUNCTION [3]
[Create] Database	Allows the user to create databases. Permission to operate on existing databases is controlled by object privileges.	CREATE DATABASE
[Create] External Table	Allows the user to create external tables. Permission to operate on existing tables is controlled by object privileges.	CREATE TABLE
[Create] Function	Allows the user to create user-defined functions (UDFs). Permission to operate on existing UDFs is controlled by object privileges.	CREATE FUNCTION
[Create] Group	Allows the user to create groups. Permission to operate on existing groups is controlled by object privileges.	CREATE ROLE
[Create] Index	For system use only. Users can't create indexes.	CREATE INDEX
[Create] Library	Allows the user to create shared libraries. Permission to operate on existing shared libraries is controlled by object privileges.	[1]
[Create] Materialized View	Allows the user to create materialized views.	CREATE VIEW
[Create] Procedure	Allows the user to create stored procedures. Permission to operate on existing stored procedures is controlled by object privileges.	CREATE PROCEDURE
[Create] Schema	Allows the user to create schemas. Permission to operate on existing schemas is controlled by object privileges.	CREATE SCHEMA
[Create] Sequence	Allows the user to create database sequences.	[1]
[Create] Synonym	Allows the user to create synonyms.	CREATE SYNONYM

ADMIN PRIVILEGE	DESCRIPTION	AZURE SYNAPSE EQUIVALENT
[Create] Table	Allows the user to create tables. Permission to operate on existing tables is controlled by object privileges.	CREATE TABLE
[Create] Temp Table	Allows the user to create temporary tables. Permission to operate on existing tables is controlled by object privileges.	CREATE TABLE
[Create] User	Allows the user to create users. Permission to operate on existing users is controlled by object privileges.	CREATE USER
[Create] View	Allows the user to create views. Permission to operate on existing views is controlled by object privileges.	CREATE VIEW
[Manage Hardware	Allows the user to do the following hardware-related operations: view hardware status, manage SPUs, manage topology and mirroring, and run diagnostic tests. The user can run these commands: nzhw and nzds.	[4]
[Manage Security	Allows the user to run commands and operations that relate to the following advanced security options such as: managing and configuring history databases, managing multi-level security objects, and specifying security for users and groups, managing database key stores and keys and key stores for the digital signing of audit data.	[4]
[Manage System	Allows the user to do the following management operations: start/stop/pause/resume the system, abort sessions, view the distribution map, system statistics, and logs. The user can use these commands: nzsystem, nzstate, nzstats, and nzsession.	[4]
Restore	Allows the user to restore the system. The user can run the nzrestore command.	[2]
Unfence	Allows the user to create or alter a user-defined function or aggregate to run in unfenced mode.	[1]

OBJECT PRIVILEGE ABORT	DESCRIPTION	AZURE SYNAPSE EQUIVALENT
Abort	Allows the user to abort sessions. Applies to groups and users.	KILL DATABASE CONNECTION

OBJECT PRIVILEGE ABORT	DESCRIPTION	AZURE SYNAPSE EQUIVALENT
Alter	Allows the user to modify object attributes. Applies to all objects.	ALTER
Delete	Allows the user to delete table rows. Applies only to tables.	DELETE
Drop	Allows the user to drop objects. Applies to all object types.	DROP
Execute	Allows the user to run user-defined functions, user-defined aggregates, or stored procedures.	EXECUTE
GenStats	Allows the user to generate statistics on tables or databases. The user can run GENERATE STATISTICS command.	[2]
Groom	Allows the user to reclaim disk space for deleted or outdated rows, and reorganize a table by the organizing keys, or to migrate data for tables that have multiple stored versions.	[2]
Insert	Allows the user to insert rows into a table. Applies only to tables.	INSERT
List	Allows the user to display an object name, either in a list or in another manner. Applies to all objects.	LIST
Select	Allows the user to select (or query) rows within a table. Applies to tables and views.	SELECT
Truncate	Allows the user to delete all rows from a table. Applies only to tables.	TRUNCATE
Update	Allows the user to modify table rows. Applies to tables only.	UPDATE

Table notes:

1. There's no direct equivalent to this function in Azure Synapse.
2. These Netezza functions are handled automatically in Azure Synapse.
3. The Azure Synapse `CREATE FUNCTION` feature incorporates Netezza aggregate functionality.
4. These features are managed automatically by the system or via the Azure portal in Azure Synapse. See the next section on Operational considerations.

Refer to Azure Synapse Analytics security permissions.

Operational considerations

> **TIP**
> Operational tasks are necessary to keep any data warehouse operating efficiently.

This section discusses how to implement typical Netezza operational tasks in Azure Synapse with minimal risk and impact to users.

As with all data warehouse products, once in production there are ongoing management tasks that are necessary to keep the system running efficiently and to provide data for monitoring and auditing. Resource utilization and capacity planning for future growth also falls into this category, as does backup/restore of data.

Netezza administration tasks typically fall into two categories:

- System administration, which is managing the hardware, configuration settings, system status, access, disk space, usage, upgrades, and other tasks.
- Database administration, which is managing user databases and their content, loading data, backing up data, restoring data, and controlling access to data and permissions.

IBM Netezza offers several ways or interfaces that you can use to perform the various system and database management tasks:

- Netezza commands (`nz*` commands) are installed in the `/nz/kit/bin` directory on the Netezza host. For many of the `nz*` commands, you must be able to sign into the Netezza system to access and run those commands. In most cases, users sign in as the default `nz` user account, but you can create other Linux user accounts on your system. Some commands require you to specify a database user account, password, and database to ensure that you have permission to do the task.
- The Netezza CLI client kits package a subset of the `nz*` commands that can be run from Windows and UNIX client systems. The client commands might also require you to specify a database user account, password, and database to ensure that you have database administrative and object permissions to perform the task.
- The SQL commands support administration tasks and queries within a SQL database session. You can run the SQL commands from the Netezza nzsql command interpreter or through SQL APIs such as ODBC, JDBC, and the OLE DB Provider. You must have a database user account to run the SQL commands with appropriate permissions for the queries and tasks that you perform.
- The NzAdmin tool is a Netezza interface that runs on Windows client workstations to manage Netezza systems.

While conceptually the management and operations tasks for different data warehouses are similar, the individual implementations may differ. In general, modern cloud-based products such as Azure Synapse tend to incorporate a more automated and "system managed" approach (as opposed to a more "manual" approach in legacy data warehouses such as Netezza).

The following sections compare Netezza and Azure Synapse options for various operational tasks.

Housekeeping tasks

> **TIP**
> Housekeeping tasks keep a production warehouse operating efficiently and optimize use of resources such as storage.

In most legacy data warehouse environments, regular "housekeeping" tasks are time-consuming. Reclaim disk storage space by removing old versions of updated or deleted rows or reorganizing data, log files, or index

blocks for efficiency (`GROOM` and `VACUUM` in Netezza). Collecting statistics is also a potentially time-consuming task, required after a bulk data ingest to provide the query optimizer with up-to-date data on which to base query execution plans.

Netezza recommends collecting statistics as follows:

- Collect statistics on unpopulated tables to set up the interval histogram used in internal processing. This initial collection makes subsequent statistics collections faster. Make sure to recollect statistics after data is added.
- Collect prototype phase statistics for newly populated tables.
- Collect production phase statistics after a significant percentage of change to the table or partition (~10% of rows). For high volumes of nonunique values, such as dates or timestamps, it may be advantageous to recollect at 7%.
- Collect production phase statistics after you've created users and applied real world query loads to the database (up to about three months of querying).
- Collect statistics in the first few weeks after an upgrade or migration during periods of low CPU utilization.

Netezza database contains many log tables in the data dictionary that accumulate data, either automatically or after certain features are enabled. Because log data grows over time, purge older information to avoid using up permanent space. There are options to automate the maintenance of these logs available.

> **TIP**
> Automate and monitor housekeeping tasks in Azure.

Azure Synapse has an option to automatically create statistics so that they can be used as needed. Perform defragmentation of indexes and data blocks manually, on a scheduled basis, or automatically. Leveraging native built-in Azure capabilities can reduce the effort required in a migration exercise.

Monitoring and auditing

> **TIP**
> Netezza Performance Portal is the recommended method of monitoring and logging for Netezza systems.

Netezza provides the Netezza Performance Portal to monitor various aspects of one or more Netezza systems including activity, performance, queuing, and resource utilization. Netezza Performance Portal is an interactive GUI that allows users to drill down into low-level details for any chart.

> **TIP**
> The Azure portal provides a UI to manage monitoring and auditing tasks for all Azure data and processes.

Similarly, Azure Synapse provides a rich monitoring experience within the Azure portal to provide insights into your data warehouse workload. The Azure portal is the recommended tool when monitoring your data warehouse as it provides configurable retention periods, alerts, recommendations, and customizable charts and dashboards for metrics and logs.

The portal also enables integration with other Azure monitoring services such as Operations Management Suite (OMS) and Azure Monitor (logs) to provide a holistic monitoring experience for not only the data warehouse but also the entire Azure analytics platform for an integrated monitoring experience.

> **TIP**
> Low-level and system-wide metrics are automatically logged in Azure Synapse.

Resource utilization statistics for Azure Synapse are automatically logged within the system. The metrics for each query include usage statistics for CPU, memory, cache, I/O, and temporary workspace, as well as connectivity information like failed connection attempts.

Azure Synapse provides a set of Dynamic Management Views (DMVs). These views are useful when actively troubleshooting and identifying performance bottlenecks with your workload.

For more information, see Azure Synapse operations and management options.

High Availability (HA) and Disaster Recovery (DR)

Netezza appliances are redundant, fault-tolerant systems, and there are diverse options in a Netezza system to enable high availability and disaster recovery.

Adding IBM Netezza Replication Services for disaster recovery improves fault tolerance by extending redundancy across local and wide area networks.

IBM Netezza Replication Services protects against data loss by synchronizing data on a primary system (the primary node) with data on one or more target nodes (subordinates). These nodes make up a replication set.

High-Availability Linux (also called *Linux-HA*) provides the failover capabilities from a primary or active Netezza host to a secondary or standby Netezza host. The main cluster management daemon in the Linux-HA solution is called *Heartbeat*. Heartbeat watches the hosts and manages the communication and status checks of services.

Each service is a resource.

Netezza groups the Netezza-specific services into the nps resource group. When Heartbeat detects problems that imply a host failure condition or loss of service to the Netezza users, Heartbeat can initiate a failover to the standby host.

Distributed Replicated Block Device (DRBD) is a block device driver that mirrors the content of block devices (hard disks, partitions, and logical volumes) between the hosts. Netezza uses the DRBD replication only on the **/nz** and **/export/home** partitions. As new data is written to the **/nz** partition and the **/export/home** partition on the primary host, the DRBD software automatically makes the same changes to the **/nz** and **/export/home** partition of the standby host.

> **TIP**
> Azure Synapse creates snapshots automatically to ensure fast recovery times.

Azure Synapse uses database snapshots to provide high availability of the warehouse. A data warehouse snapshot creates a restore point that can be used to recover or copy a data warehouse to a previous state. Since Azure Synapse is a distributed system, a data warehouse snapshot consists of many files that are in Azure Storage. Snapshots capture incremental changes from the data stored in your data warehouse.

> **TIP**
> Use user-defined snapshots to define a recovery point before key updates.

> **TIP**
> Microsoft Azure provides automatic backups to a separate geographical location to enable DR.

Azure Synapse automatically takes snapshots throughout the day, creating restore points that are available for seven days. You can't change this retention period. Azure Synapse supports an eight-hour recovery point objective (RPO). A data warehouse can be restored in the primary region from any one of the snapshots taken in the past seven days.

User-defined restore points are also supported, allowing manual triggering of snapshots to create restore points of a data warehouse before and after large modifications. This capability ensures that restore points are logically consistent, which provides additional data protection in case of any workload interruptions or user errors for a desired RPO less than 8 hours.

As well as the snapshots described previously, Azure Synapse also performs as standard a geo-backup once per day to a paired data center. The RPO for a geo-restore is 24 hours. You can restore the geo-backup to a server in any other region where Azure Synapse is supported. A geo-backup ensures that a data warehouse can be restored in case the restore points in the primary region aren't available.

Workload management

> **TIP**
> In a production data warehouse, there are typically mixed workloads with different resource usage characteristics running concurrently.

Netezza incorporates various features for managing workloads:

TECHNIQUE	DESCRIPTION
Scheduler rules	Scheduler rules influence the scheduling of plans. Each scheduler rule specifies a condition or set of conditions. Each time the scheduler receives a plan, it evaluates all modifying scheduler rules and carries out the appropriate actions. Each time the scheduler selects a plan for execution, it evaluates all limiting scheduler rules. The plan is executed only if doing so wouldn't exceed a limit imposed by a limiting scheduler rule. Otherwise, the plan waits. This provides you with a way to classify and manipulate plans in a way that influences the other WLM techniques (SQB, GRA, and PQE).
Guaranteed resource allocation (GRA)	You can assign a minimum share and a maximum percentage of total system resources to entities called *resource groups*. The scheduler ensures that each resource group receives system resources in proportion to its minimum share. A resource group receives a larger share of resources when other resource groups are idle, but never receives more than its configured maximum percentage. Each plan is associated with a resource group, and the settings of that resource group settings determine what fraction of available system resources are to be made available to process the plan.

TECHNIQUE	DESCRIPTION
Short query bias (SQB)	Resources (that is, scheduling slots, memory, and preferential queuing) are reserved for short queries. A short query is a query for which the cost estimate is less than a specified maximum value (the default is two seconds). With SQB, short queries can run even when the system is busy processing other, longer queries.
Prioritized query execution (PQE)	Based on settings that you configure, the system assigns a priority—critical, high, normal, or low—to each query. The priority depends on factors such as the user, group, or session associated with the query. The system can then use the priority as a basis for allocating resources.

Azure Synapse automatically logs resource utilization statistics. Metrics include usage statistics for CPU, memory, cache, I/O, and temporary workspace for each query. Azure Synapse also logs connectivity information, such as failed connection attempts.

> **TIP**
> Low-level and system-wide metrics are automatically logged within Azure.

In Azure Synapse, resource classes are pre-determined resource limits that govern compute resources and concurrency for query execution. Resource classes can help you manage your workload by setting limits on the number of queries that run concurrently and on the compute resources assigned to each query. There's a trade-off between memory and concurrency.

Azure Synapse supports these basic workload management concepts:

- **Workload classification**: you can assign a request to a workload group to set importance levels.
- **Workload importance**: you can influence the order in which a request gets access to resources. By default, queries are released from the queue on a first-in, first-out basis as resources become available. Workload importance allows higher priority queries to receive resources immediately regardless of the queue.
- **Workload isolation**: you can reserve resources for a workload group, assign maximum and minimum usage for varying resources, limit the resources a group of requests can consume can, and set a timeout value to automatically kill runaway queries.

Running mixed workloads can pose resource challenges on busy systems. A successful workload management scheme effectively manages resources, ensures highly efficient resource utilization, and maximizes return on investment (ROI). The workload classification, workload importance, and workload isolation gives more control over how workload utilizes system resources.

The workload management guide describes the techniques to analyze the workload, manage and monitor workload importance](../../sql-data-warehouse/sql-data-warehouse-how-to-manage-and-monitor-workload-importance.md), and the steps to convert a resource class to a workload group. Use the Azure portal and T-SQL queries on DMVs to monitor the workload to ensure that the applicable resources are efficiently utilized. Azure Synapse provides a set of Dynamic Management Views (DMVs) for monitoring all aspects of workload management. These views are useful when actively troubleshooting and identifying performance bottlenecks in your workload.

This information can also be used for capacity planning, determining the resources required for additional users or application workload. This also applies to planning scale up/scale downs of compute resources for cost-

effective support of "spiky" workloads, such as workloads with temporary, intense bursts of activity surrounded by periods of infrequent activity.

For more information on workload management in Azure Synapse, see Workload management with resource classes.

Scale compute resources

> **TIP**
> A major benefit of Azure is the ability to independently scale up and down compute resources on demand to handle peaky workloads cost-effectively.

The architecture of Azure Synapse separates storage and compute, allowing each to scale independently. As a result, compute resources can be scaled to meet performance demands independent of data storage. You can also pause and resume compute resources. A natural benefit of this architecture is that billing for compute and storage is separate. If a data warehouse isn't in use, you can save on compute costs by pausing compute.

Compute resources can be scaled up or scaled back by adjusting the data warehouse units setting for the data warehouse. Loading and query performance will increase linearly as you add more data warehouse units.

Adding more compute nodes adds more compute power and ability to leverage more parallel processing. As the number of compute nodes increases, the number of distributions per compute node decreases, providing more compute power and parallel processing for queries. Similarly, decreasing data warehouse units reduces the number of compute nodes, which reduces the compute resources for queries.

Next steps

To learn more about visualization and reporting, see the next article in this series: Visualization and reporting for Netezza migrations.

Visualization and reporting for Netezza migrations

8/19/2022 • 20 minutes to read • Edit Online

This article is part four of a seven-part series that provides guidance on how to migrate from Netezza to Azure Synapse Analytics. The focus of this article is best practices for visualization and reporting.

Access Azure Synapse Analytics using Microsoft and third-party BI tools

Organizations access data warehouses and data marts using a range of business intelligence (BI) tools and applications. Some examples of BI products are:

- Microsoft BI tools, such as Power BI.
- Office applications, such as Microsoft Excel spreadsheets.
- Third-party BI tools from different vendors.
- Custom analytics applications with embedded BI tool functionality.
- Operational applications that support on-demand BI by running queries and reports on a BI platform that in turn queries data in a data warehouse or data mart.
- Interactive data science development tools, such as Azure Synapse Spark Notebooks, Azure Machine Learning, RStudio, and Jupyter Notebooks.

If you migrate visualization and reporting as part of your data warehouse migration, all existing queries, reports, and dashboards generated by BI products need to run in the new environment. Your BI products must yield the same results on Azure Synapse as they did in your legacy data warehouse environment.

For consistent results after migration, all BI tools and application dependencies must work after you've migrated your data warehouse schema and data to Azure Synapse. The dependencies include less visible aspects, such as access and security. When you address access and security, ensure that you migrate:

- Authentication so users can sign into the data warehouse and data mart databases on Azure Synapse.
- All users to Azure Synapse.
- All user groups to Azure Synapse.
- All roles to Azure Synapse.
- All authorization privileges governing access control to Azure Synapse.
- User, role, and privilege assignments to mirror what you had in your existing data warehouse before migration. For example:
    - Database object privileges assigned to roles
    - Roles assigned to user groups
    - Users assigned to user groups and/or roles

Access and security are important considerations for data access in the migrated system and are discussed in more detail in Security, access, and operations for Netezza migrations.

TIP

Existing users, user groups, roles, and assignments of access security privileges need to be migrated first for migration of reports and visualizations to succeed.

Migrate all required data to ensure that the reports and dashboards that query data in the legacy environment produce the same results in Azure Synapse.

Business users will expect a seamless migration, with no surprises that destroy their confidence in the migrated system on Azure Synapse. Take care to allay any fears that your users might have through good communication. Your users will expect that:

- Table structure remains the same when directly referred to in queries.
- Table and column names remain the same when directly referred to in queries. For instance, calculated fields defined on columns in BI tools shouldn't fail when aggregate reports are produced.
- Historical analysis remains the same.
- Data types remain the same, if possible.
- Query behavior remains the same.
- ODBC/JDBC drivers are tested to ensure that query behavior remains the same.

TIP

Communication and business user involvement are critical to success.

If BI tools query views in the underlying data warehouse or data mart database, will those views still work after the migration? Some views might not work if there are proprietary SQL extensions specific to your legacy data warehouse DBMS that have no equivalent in Azure Synapse. If so, you need to know about those incompatibilities and find a way to resolve them.

TIP

Views and SQL queries using proprietary SQL query extensions are likely to result in incompatibilities that impact BI reports and dashboards.

Other issues, like the behavior of `NULL` values or data type variations across DBMS platforms, need to be tested to ensure that even slight differences don't exist in calculation results. Minimize those issues and take all necessary steps to shield business users from being affected by them. Depending on your legacy data warehouse environment, third-party tools that can help hide the differences between the legacy and new environments so that BI tools and applications run unchanged.

Testing is critical to visualization and report migration. You need a test suite and agreed-on test data to run and rerun tests in both environments. A test harness is also useful, and a few are mentioned in this guide. Also, it's important to involve business users in the testing aspect of the migration to keep confidence high and to keep them engaged and part of the project.

TIP

Use repeatable tests to ensure reports, dashboards, and other visualizations migrate successfully.

You might be thinking about switching BI tools, for example to migrate to Power BI. The temptation is to make such changes at the same time you're migrating your schema, data, ETL processing, and more. However, to minimize risk, it's better to migrate to Azure Synapse first and get everything working before undertaking further modernization.

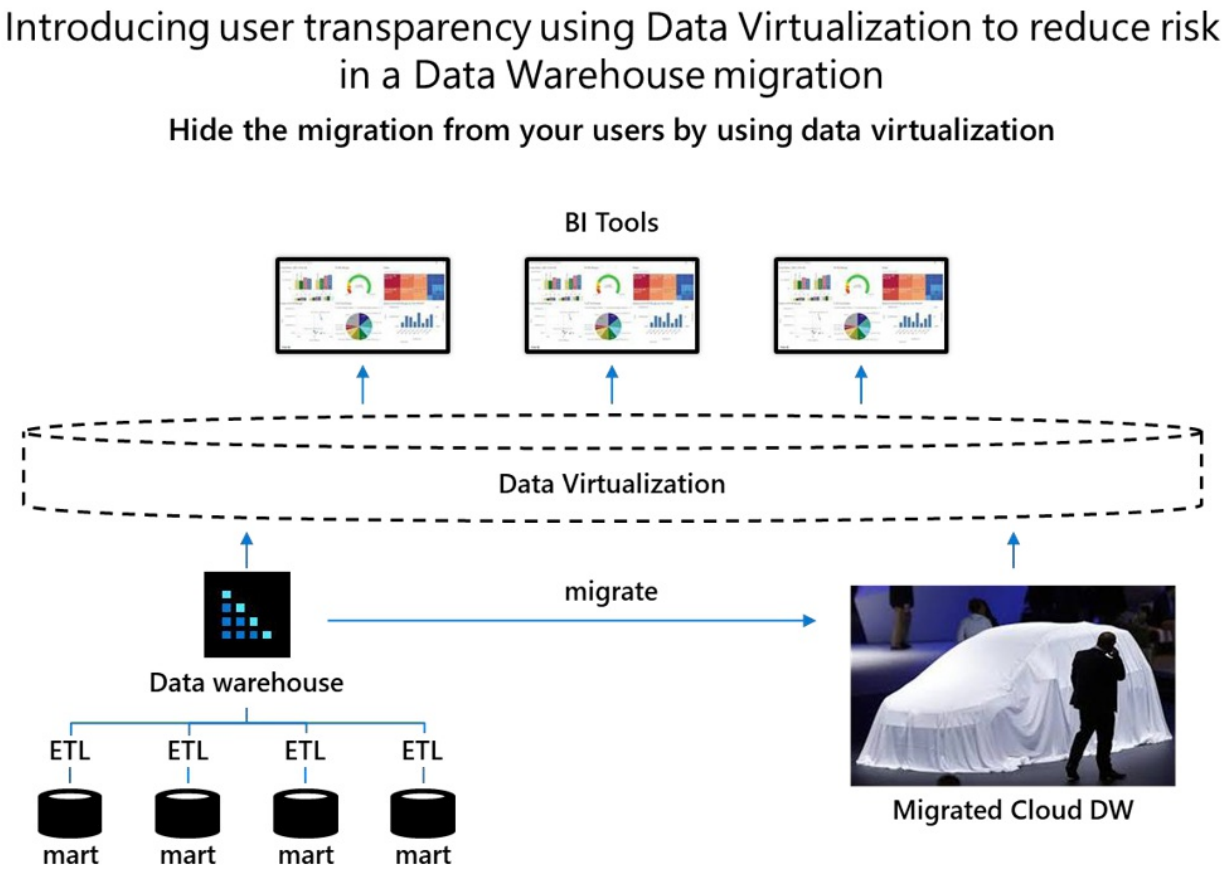
If your existing BI tools run on-premises, ensure they can connect to Azure Synapse through your firewall so you're able to run comparisons against both environments. Alternatively, if the vendor of your existing BI tools offers their product on Azure, you can try it there. The same applies for applications running on-premises that embed BI or call your BI server on demand, for example by requesting a "headless report" with XML or JSON data.

There's a lot to think about here, so let's take a closer look.

Use data virtualization to minimize the impact of migration on BI tools and reports

During migration, you might be tempted to fulfill long-term requirements like opening business requests, adding missing data, or implementing new features. However, such changes can affect BI tool access to your data warehouse, especially if the change involves structural changes to your data model. If you want to adopt an agile data modeling technique or implement structural changes, do so *after* migration.

One way to minimize the effect of schema changes or other structural changes on your BI tools is to introduce data virtualization between the BI tools and your data warehouse and data marts. The following diagram shows how data virtualization can hide a migration from users.



Data virtualization breaks the dependency between business users utilizing self-service BI tools and the physical schema of the underlying data warehouse and data marts that are being migrated.

> **TIP**
> Data virtualization allows you to shield business users from structural changes during migration so they remain unaware of those changes. Structural changes include schema alterations that tune your data model for Azure Synapse.

With data virtualization, any schema alterations made during a migration to Azure Synapse, for example to optimize performance, can be hidden from business users because they only have access to virtual tables in the data virtualization layer. And, if you make structural changes, you only need to update the mappings between the data warehouse or data marts and any virtual tables. With data virtualization, users remain unaware of structural changes. Microsoft partners provide data virtualization software.

Identify high-priority reports to migrate first

A key question when migrating your existing reports and dashboards to Azure Synapse is which ones to migrate first. Several factors might drive that decision, such as:

- Usage
- Business value
- Ease of migration
- Data migration strategy

The following sections discuss these factors.

Whatever your decision, it must involve your business users because they produce the reports, dashboards, and other visualizations, and make business decisions based on insights from those items. Everyone benefits when you can:

- Migrate reports and dashboards seamlessly,
- Migrate reports and dashboards with minimal effort, and
- Point your BI tool(s) at Azure Synapse instead of your legacy data warehouse system, and get like-for-like reports, dashboards, and other visualizations.

Migrate reports based on usage

Usage is often an indicator of business value. Unused reports and dashboards clearly don't contribute to business decisions or offer current value. If you don't have a way to find out which reports and dashboards are unused, you can use one of the several BI tools that provide usage statistics.

If your legacy data warehouse has been up and running for years, there's a good chance you have hundreds, if not thousands, of reports in existence. It's worth compiling an inventory of reports and dashboards and identifying their business purpose and usage statistics.

For unused reports, determine whether to decommission them to reduce your migration effort. A key question when deciding whether to decommission an unused report is whether the report is unused because people don't know it exists, because it offers no business value, or because it's been superseded by another report.

Migrate reports based on business value

Usage alone isn't always a good indicator of business value. You might want to consider the extent to which a report's insights contribute to business value. One way to do that is to evaluate the profitability of every business decision that relies on the report and the extent of the reliance. However, that information is unlikely to be readily available in most organizations.

Another way to evaluate business value is to look at the alignment of a report with business strategy. The business strategy set by your executive typically lays out strategic business objectives (SBOs), key performance

indicators (KPIs), KPI targets that need to be achieved, and who is accountable for achieving them. You can classify a report by which SBOs the report contributes to, such as fraud reduction, improved customer engagement, and optimized business operations. Then, you can prioritize for migration the reports and dashboards that are associated with high-priority objectives. In this way, the initial migration can deliver business value in a strategic area.

Another way to evaluate business value is to classify reports and dashboards as operational, tactical, or strategic to identify at which business level they're used. SBOs require contributions at all these levels. By knowing which reports and dashboards are used, at what level, and what objectives they're associated with, you're able to focus the initial migration on high-priority business value. You can use the following *business strategy objective* table to evaluate reports and dashboards.

LEVEL	REPORT / DASHBOARD NAME	BUSINESS PURPOSE	DEPARTMENT USED	USAGE FREQUENCY	BUSINESS PRIORITY
Strategic					
Tactical					
Operational					

Metadata discovery tools like Azure Data Catalog let business users tag and rate data sources to enrich the metadata for those data sources to assist with their discovery and classification. You can use the metadata for a report or dashboard to help you understand its business value. Without such tools, understanding the contribution of reports and dashboards to business value is likely to be a time-consuming task, whether you're migrating or not.

Migrate reports based on data migration strategy

If your migration strategy is based on migrating data marts first, then the order of data mart migration will affect which reports and dashboards are migrated first. If your strategy is based on business value, the order in which you migrate data marts to Azure Synapse will reflect business priorities. Metadata discovery tools can help you implement your strategy by showing you which data mart tables supply data for which reports.

> **TIP**
> Your data migration strategy affects which reports and visualizations get migrated first.

Migration incompatibility issues that can affect reports and visualizations

BI tools produce reports, dashboards, and other visualizations by issuing SQL queries that access physical tables and/or views in your data warehouse or data mart. When you migrate your legacy data warehouse to Azure Synapse, several factors can affect the ease of migration of reports, dashboards, and other visualizations. Those factors include:

- Schema incompatibilities between the environments.
- SQL incompatibilities between the environments.

Schema incompatibilities

During a migration, schema incompatibilities in the data warehouse or data mart tables that supply data for reports, dashboards, and other visualizations can be:

- Non-standard table types in your legacy data warehouse DBMS that don't have an equivalent in Azure

Synapse.

- Data types in your legacy data warehouse DBMS that don't have an equivalent in Azure Synapse.

In most cases, there's a workaround to the incompatibilities. For example, you can migrate the data in an unsupported table type into a standard table with appropriate data types and indexed or partitioned on a date/time column. Similarly, it might be possible to represent unsupported data types in another type of column and perform calculations in Azure Synapse to achieve the same results.

> **TIP**
> Schema incompatibilities include legacy warehouse DBMS table types and data types that are unsupported on Azure Synapse.

To identify the reports affected by schema incompatibilities, run queries against the system catalog of your legacy data warehouse to identify the tables with unsupported data types. Then, you can use metadata from your BI tool to identify the reports that access data in those tables. For more information about how to identify object type incompatibilities, see Unsupported Netezza database object types.

> **TIP**
> Query the system catalog of your legacy warehouse DBMS to identify schema incompatibilities with Azure Synapse.

The effect of schema incompatibilities on reports, dashboards, and other visualizations might be less than you think because many BI tools don't support the less generic data types. As a result, your legacy data warehouse might already have views that `CAST` unsupported data types to more generic types.

SQL incompatibilities

During a migration, SQL incompatibilities are likely to affect any report, dashboard, or other visualization in an application or tool that:

- Accesses legacy data warehouse DBMS views that include proprietary SQL functions that have no equivalent in Azure Synapse.
- Issues SQL queries that include proprietary SQL functions, specific to the SQL dialect of your legacy environment, that have no equivalent in Azure Synapse.

Gauge the impact of SQL incompatibilities on your reporting portfolio

Your reporting portfolio might include embedded query services, reports, dashboards, and other visualizations. Don't rely on the documentation associated with those items to gauge the effect of SQL incompatibilities on the migration of your reporting portfolio to Azure Synapse. You need to use a more precise way to assess the effect of SQL incompatibilities.

Use EXPLAIN statements to find SQL incompatibilities

You can find SQL incompatibilities by querying the `_v_qryhist` system table to view recent SQL activity in your legacy Netezza data warehouse. For more information, see Query history table. Use a script to extract a representative set of SQL statements to a file. Then, prefix each SQL statement with an `EXPLAIN` statement, and run those `EXPLAIN` statements in Azure Synapse. Any SQL statements containing proprietary unsupported SQL extensions will be rejected by Azure Synapse when the `EXPLAIN` statements are executed. This approach lets you assess the extent of SQL incompatibilities.

Metadata from your legacy data warehouse DBMS can also help you identify incompatible views. As before, capture a representative set of SQL statements, prefix each SQL statement with an `EXPLAIN` statement, and run those `EXPLAIN` statements in Azure Synapse to identify views with incompatible SQL.

> **TIP**
> Gauge the impact of SQL incompatibilities by harvesting your DBMS log files and running `EXPLAIN` statements.

Test report and dashboard migration to Azure Synapse Analytics

A key element of data warehouse migration is testing of reports and dashboards in Azure Synapse to verify the migration has worked. Define a series of tests and a set of required outcomes for each test that you will run to verify success. Test and compare the reports and dashboards across your existing and migrated data warehouse systems to:

- Identify whether any schema changes that were made during migration affected the ability of reports to run, report results, or the corresponding report visualizations. An example of a schema change is if you mapped an incompatible data type to an equivalent data type that's supported in Azure Synapse.
- Verify that all users are migrated.
- Verify that all roles are migrated, and users are assigned to those roles.
- Verify that all data access security privileges are migrated to ensure access control list (ACL) migration.
- Ensure consistent results for all known queries, reports, and dashboards.
- Ensure that data and ETL migration is complete and error-free.
- Ensure that data privacy is upheld.
- Test performance and scalability.
- Test analytical functionality.

> **TIP**
> Test and tune performance to minimize compute costs.

For information about how to migrate users, user groups, roles, and privileges, see Security, access, and operations for Netezza migrations.

Automate testing as much as possible to make each test repeatable and to support a consistent approach to evaluating test results. Automation works well for known regular reports, and can be managed via Azure Synapse pipelines or Azure Data Factory orchestration. If you already have a suite of test queries in place for regression testing, you can use the existing testing tools to automate post migration testing.

> **TIP**
> Best practice is to build an automated test suite to make tests repeatable.

Ad-hoc analysis and reporting are more challenging and require compilation of a set of tests to verify that the same reports and dashboards from before and after migration are consistent. If you find inconsistencies, then your ability to compare metadata lineage across the original and migrated systems during migration testing becomes crucial. That comparison can highlight differences and pinpoint where inconsistencies originated, when detection by other means is difficult.

> **TIP**
> Leverage tools that compare metadata lineage to verify results.

Analyze lineage to understand dependencies between reports, dashboards, and data

Your understanding of lineage is a critical factor in the successful migration of reports and dashboards. Lineage is metadata that shows the journey of migrated data so you can track its path from a report or dashboard all the way back to the data source. Lineage shows how data has traveled from point to point, its location in the data warehouse and/or data mart, and which reports and dashboards use it. Lineage can help you understand what happens to data as it travels through different data stores, such as files and databases, different ETL pipelines, and into reports. When business users have access to data lineage, it improves trust, instills confidence, and supports informed business decisions.

> **TIP**
> Your ability to access metadata and data lineage from reports all the way back to a data source is critical for verifying that migrated reports work correctly.

In multi-vendor data warehouse environments, business analysts in BI teams might map out data lineage. For example, if you use different vendors for ETL, data warehouse, and reporting, and each vendor has its own metadata repository, then figuring out where a specific data element in a report came from can be challenging and time-consuming.

> **TIP**
> Tools that automate the collection of metadata and show end-to-end lineage in a multi-vendor environment are valuable during a migration.

To migrate seamlessly from a legacy data warehouse to Azure Synapse, use end-to-end data lineage to prove like-for-like migration when you're comparing the reports and dashboards generated by each environment. To show the end-to-end data journey, you'll need to capture and integrate metadata from several tools. Having access to tools that support automated metadata discovery and data lineage, helps you identify duplicate reports or ETL processes, and find reports that rely on obsolete, questionable, or non-existent data sources. You can use that information to reduce the number of reports and ETL processes that you migrate.

You can also compare the end-to-end lineage of a report in Azure Synapse to the end-to-end lineage of the same report in your legacy environment to check for differences that might have inadvertently occurred during migration. This type of comparison is exceptionally useful when you need to test and verify migration success.

Data lineage visualization not only reduces time, effort, and error in the migration process, but also enables faster migration.

By using automated metadata discovery and data lineage tools that compare lineage, you can verify that a report in Azure Synapse that's produced from migrated data is produced in the same way in your legacy environment. This capability also helps you determine:

- What data needs to be migrated to ensure successful report and dashboard execution in Azure Synapse.
- What transformations have been and should be performed to ensure successful execution in Azure Synapse.

- How to reduce report duplication.

Automated metadata discovery and data lineage tools substantially simplify the migration process because they help businesses become more aware of their data assets and to know what needs to be migrated to Azure Synapse to achieve a solid reporting environment.

Several ETL tools provide end-to-end lineage capability, so check whether your existing ETL tool has that capability if you plan to use it with Azure Synapse. Azure Synapse pipelines or Data Factory both support the ability to view lineage in mapping flows. Microsoft partners also provide automated metadata discovery, data lineage, and lineage comparison tools.

Migrate BI tool semantic layers to Azure Synapse Analytics

Some BI tools have what is known as a semantic metadata layer. That layer simplifies business user access to the underlying physical data structures in a data warehouse or data mart database. The semantic metadata layer simplifies access by providing high-level objects like dimensions, measures, hierarchies, calculated metrics, and joins. The high-level objects use business terms that are familiar to business analysts, and map to physical data structures in your data warehouse or data mart.

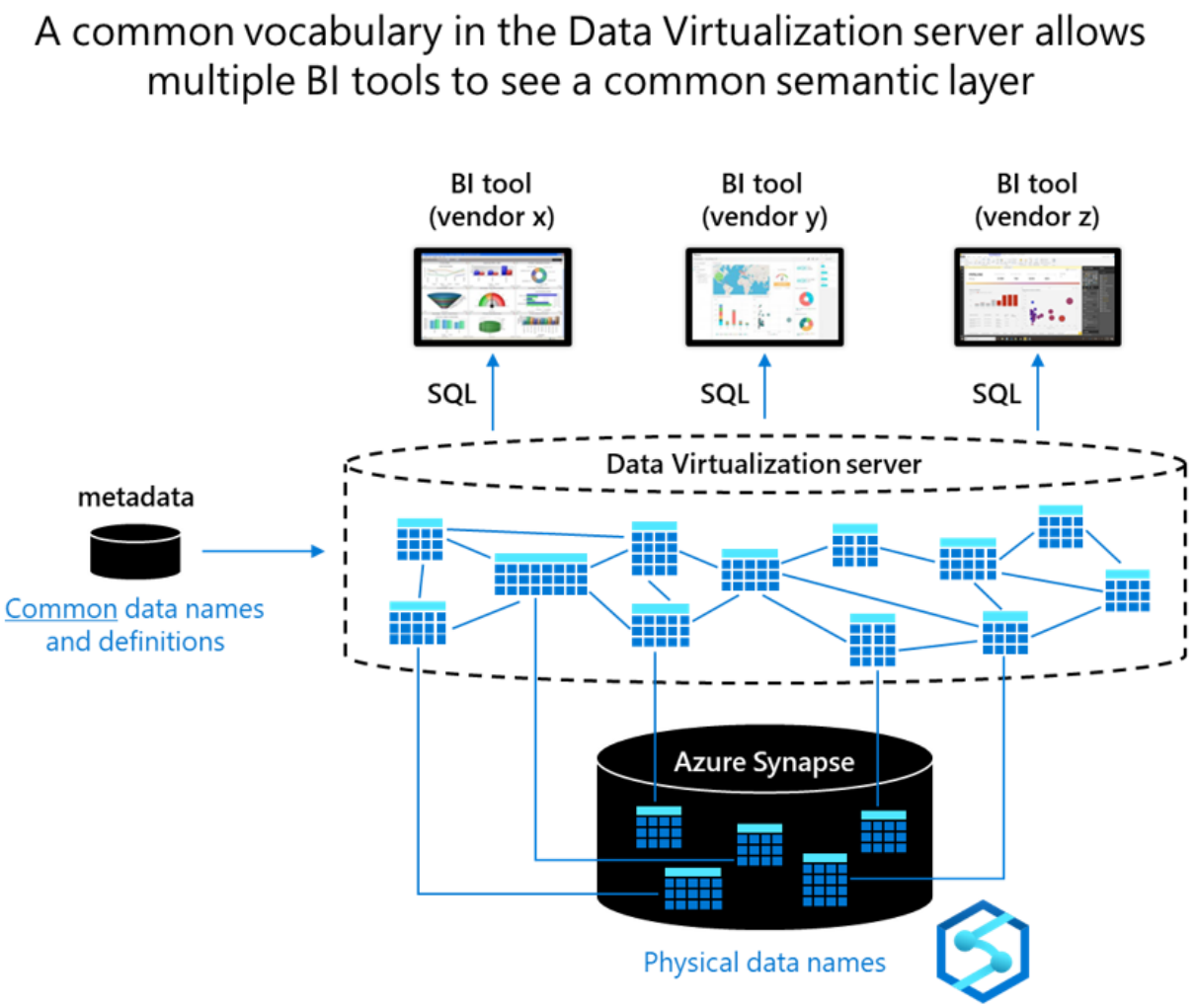
> **TIP**
> Some BI tools have semantic layers that simplify business user access to physical data structures in your data warehouse or data mart.

In a data warehouse migration, changes to column names or table names may be forced upon you. For example, in IBM Netezza, table names can have a "#". In Azure Synapse, the "#" is only allowed as a prefix to a table name to indicate a temporary table. In IBM Netezza, TEMPORARY TABLES do not necessarily have a "#" in the name, but in Synapse they must. You may need to do some rework to change table mappings in such cases.

To achieve consistency across multiple BI tools, create a universal semantic layer by using a data virtualization server that sits between BI tools and applications and Azure Synapse. In the data virtualization server, use common data names for high-level objects like dimensions, measures, hierarchies, and joins. That way you configure everything, including calculated fields, joins, and mappings, only once instead of in every tool. Then, point all BI tools at the data virtualization server.

> **TIP**
> Use data virtualization to create a common semantic layer to guarantee consistency across all BI tools in an Azure Synapse environment.

With data virtualization, you get consistency across all BI tools and break the dependency between BI tools and applications and the underlying physical data structures in Azure Synapse. Microsoft partners can help you achieve consistency in Azure. The following diagram shows how a common vocabulary in the data virtualization server lets multiple BI tools see a common semantic layer.



Conclusions

In a lift and shift data warehouse migration, most reports, dashboards, and other visualizations should migrate easily.

During a migration from a legacy environment, you might find that data in the legacy data warehouse or data mart tables is stored in unsupported data types. Or, you may find legacy data warehouse views that include proprietary SQL with no equivalent in Azure Synapse. If so, you'll need to resolve those issues to ensure a successful migration to Azure Synapse.

Don't rely on user-maintained documentation to identify where issues are located. Instead, use `EXPLAIN` statements because they're a quick, pragmatic way to identify SQL incompatibilities. Rework the incompatible SQL statements to achieve equivalent functionality in Azure Synapse. Also, use automated metadata discovery and lineage tools to understand dependencies, find duplicate reports, and identify invalid reports that rely on obsolete, questionable, or non-existent data sources. Use lineage tools to compare lineage to verify that reports running in your legacy data warehouse environment are produced identically in Azure Synapse.

Don't migrate reports that you no longer use. BI tool usage data can help you determine which reports aren't in use. For the reports, dashboards, and other visualizations that you do want to migrate, migrate all users, user groups, roles, and privileges. If you're using business value to drive your report migration strategy, associate reports with strategic business objectives and priorities to help identify the contribution of report insights to specific objectives. If you're migrating data mart by data mart, use metadata to identify which reports are dependent on which tables and views, so you can make an informed decision about which data marts to migrate first.

> **TIP**
> Identify incompatibilities early to gauge the extent of the migration effort. Migrate your users, group roles, and privilege assignments. Only migrate the reports and visualizations that are used and are contributing to business value.

Structural changes to the data model of your data warehouse or data mart can occur during a migration. Consider using data virtualization to shield BI tools and applications from structural changes. With data virtualization, you can use a common vocabulary to define a common semantic layer. The common semantic layer guarantees consistent common data names, definitions, metrics, hierarchies, and joins across all BI tools and applications in the new Azure Synapse environment.

Next steps

To learn more about minimizing SQL issues, see the next article in this series: Minimizing SQL issues for Netezza migrations.

Minimize SQL issues for Netezza migrations

8/19/2022 • 12 minutes to read • Edit Online

This article is part five of a seven-part series that provides guidance on how to migrate from Netezza to Azure Synapse Analytics. The focus of this article is best practices for minimizing SQL issues.

Overview

Characteristics of Netezza environments

> **TIP**
> Netezza pioneered the "data warehouse appliance" concept in the early 2000s.

In 2003, Netezza initially released their data warehouse appliance product. It reduced the cost of entry and improved the ease-of-use of massively parallel processing (MPP) techniques to enable data processing at scale more efficiently than the existing mainframe or other MPP technologies available at the time. Since then, the product has evolved and has many installations among large financial institutions, telecommunications, and retail companies. The original implementation used proprietary hardware including field programmable gate arrays—or FPGAs—and was accessible via ODBC or JDBC network connection over TCP/IP.

Most existing Netezza installations are on-premises, so many users are considering migrating some or all their Netezza data to Azure Synapse Analytics to gain the benefits of a move to a modern cloud environment.

> **TIP**
> Many existing Netezza installations are data warehouses using a dimensional data model.

Netezza technology is often used to implement a data warehouse, supporting complex analytic queries on large data volumes using SQL. Dimensional data models—star or snowflake schemas—are common, as is the implementation of data marts for individual departments.

This combination of SQL and dimensional data models simplifies migration to Azure Synapse, since the basic concepts and SQL skills are transferable. The recommended approach is to migrate the existing data model as-is to reduce risk and time taken. Even if the eventual intention is to make changes to the data model (for example, moving to a data vault model), perform an initial as-is migration and then make changes within the Azure cloud environment, leveraging the performance, elastic scalability, and cost advantages there.

While the SQL language has been standardized, individual vendors have in some cases implemented proprietary extensions. This document highlights potential SQL differences you may encounter while migrating from a legacy Netezza environment, and provides workarounds.

Use Azure Data Factory to implement a metadata-driven migration

> **TIP**
> Automate the migration process by using Azure Data Factory capabilities.

Automate and orchestrate the migration process by making use of the capabilities in the Azure environment. This approach also minimizes the migration's impact on the existing Netezza environment, which may already be running close to full capacity.

Azure Data Factory is a cloud-based data integration service that allows creation of data-driven workflows in the cloud for orchestrating and automating data movement and data transformation. Using Data Factory, you can create and schedule data-driven workflows—called pipelines—that can ingest data from disparate data stores. It can process and transform data by using compute services such as Azure HDInsight Hadoop, Spark, Azure Data Lake Analytics, and Azure Machine Learning.

By creating metadata to list the data tables to be migrated and their location, you can use the Data Factory facilities to manage and automate parts of the migration process. You can also use Azure Synapse Pipelines.

SQL DDL differences between Netezza and Azure Synapse

SQL Data Definition Language (DDL)

> **TIP**
> SQL DDL commands `CREATE TABLE` and `CREATE VIEW` have standard core elements but are also used to define implementation-specific options.

The ANSI SQL standard defines the basic syntax for DDL commands such as `CREATE TABLE` and `CREATE VIEW`. These commands are used within both Netezza and Azure Synapse, but they've also been extended to allow definition of implementation-specific features such as indexing, table distribution, and partitioning options.

The following sections discuss Netezza-specific options to consider during a migration to Azure Synapse.

Table considerations

> **TIP**
> Use existing indexes to give an indication of candidates for indexing in the migrated warehouse.

When migrating tables between different technologies, only the raw data and its descriptive metadata get physically moved between the two environments. Other database elements from the source system, such as indexes and log files, aren't directly migrated as these may not be needed or may be implemented differently within the new target environment. For example, the `TEMPORARY` option within Netezza's `CREATE TABLE` syntax is equivalent to prefixing the table name with a "#" character in Azure Synapse.

It's important to understand where performance optimizations—such as indexes—were used in the source environment. This indicates where performance optimization can be added in the new target environment. For example, if zone maps were created in the source Netezza environment, this might indicate that a non-clustered index should be created in the migrated Azure Synapse database. Other native performance optimization techniques, such as table replication, may be more applicable than a straight "like-for-like" index creation.

Unsupported Netezza database object types

> **TIP**
> Netezza-specific features can be replaced by Azure Synapse features.

Netezza implements some database objects that aren't directly supported in Azure Synapse, but there are methods to achieve the same functionality within the new environment:

- Zone maps: in Netezza, zone maps are automatically created and maintained for some column types and are used at query time to restrict the amount of data to be scanned. Zone maps are created on the following column types:

- `INTEGER` columns of length 8 bytes or less.
- Temporal columns. For instance, `DATE`, `TIME`, and `TIMESTAMP`.
- `CHAR` columns, if these are part of a materialized view and mentioned in the `ORDER BY` clause.

You can find out which columns have zone maps by using the `nz_zonemap` utility, which is part of the NZ Toolkit. Azure Synapse doesn't include zone maps, but you can achieve similar results by using other user-defined index types and/or partitioning.

- Clustered base tables (CBT): in Netezza, CBTs are commonly used for fact tables, which can have billions of records. Scanning such a huge table requires a lot of processing time, since a full table scan might be needed to get relevant records. Organizing records on restrictive CBT allows Netezza to group records in same or nearby extents. This process also creates zone maps that improve the performance by reducing the amount of data to be scanned.

  In Azure Synapse, you can achieve a similar effect by use of partitioning and/or use of other indexes.

- Materialized views: Netezza supports materialized views and recommends creating one or more of these over large tables having many columns where only a few of those columns are regularly used in queries. The system automatically maintains materialized views when data in the base table is updated.

  Azure Synapse supports materialized views, with the same functionality as Netezza.

Netezza data type mapping

> **TIP**
> Assess the impact of unsupported data types as part of the preparation phase.

Most Netezza data types have a direct equivalent in Azure Synapse. The following table shows these data types along with the recommended approach for mapping them.

NETEZZA DATA TYPE	AZURE SYNAPSE DATA TYPE
BIGINT	BIGINT
BINARY VARYING(n)	VARBINARY(n)
BOOLEAN	BIT
BYTEINT	TINYINT
CHARACTER VARYING(n)	VARCHAR(n)
CHARACTER(n)	CHAR(n)
DATE	DATE(date)
DECIMAL(p,s)	DECIMAL(p,s)
DOUBLE PRECISION	FLOAT
FLOAT(n)	FLOAT(n)
INTEGER	INT

NETEZZA DATA TYPE	AZURE SYNAPSE DATA TYPE
INTERVAL	INTERVAL data types aren't currently directly supported in Azure Synapse but can be calculated using temporal functions such as DATEDIFF.
MONEY	MONEY
NATIONAL CHARACTER VARYING(n)	NVARCHAR(n)
NATIONAL CHARACTER(n)	NCHAR(n)
NUMERIC(p,s)	NUMERIC(p,s)
REAL	REAL
SMALLINT	SMALLINT
ST_GEOMETRY(n)	Spatial data types such as ST_GEOMETRY aren't currently supported in Azure Synapse, but the data could be stored as VARCHAR or VARBINARY.
TIME	TIME
TIME WITH TIME ZONE	DATETIMEOFFSET
TIMESTAMP	DATETIME

Data Definition Language (DDL) generation

> **TIP**
> Use existing Netezza metadata to automate the generation of `CREATE TABLE` and `CREATE VIEW` DDL for Azure Synapse.

Edit existing Netezza `CREATE TABLE` and `CREATE VIEW` scripts to create the equivalent definitions with modified data types as described previously if necessary. Typically, this involves removing or modifying any extra Netezza-specific clauses such as `ORGANIZE ON`.

However, all the information that specifies the current definitions of tables and views within the existing Netezza environment is maintained within system catalog tables. This is the best source of this information as it's guaranteed to be up to date and complete. Be aware that user-maintained documentation may not be in sync with the current table definitions.

Access this information by using utilities such as `nz_ddl_table` and generate the `CREATE TABLE` DDL statements. Edit these statements for the equivalent tables in Azure Synapse.

> **TIP**
> Third-party tools and services can automate data mapping tasks.

There are Microsoft partners who offer tools and services to automate migration, including data-type mapping. Also, if a third-party ETL tool such as Informatica or Talend is already in use in the Netezza environment, that tool can implement any required data transformations.

SQL DML differences between Netezza and Azure Synapse

SQL Data Manipulation Language (DML)

> **TIP**
> SQL DML commands `SELECT`, `INSERT`, and `UPDATE` have standard core elements but may also implement different syntax options.

The ANSI SQL standard defines the basic syntax for DML commands such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. Both Netezza and Azure Synapse use these commands, but in some cases there are implementation differences.

The following sections discuss the Netezza-specific DML commands that you should consider during a migration to Azure Synapse.

SQL DML syntax differences

Be aware of these differences in SQL Data Manipulation Language (DML) syntax between Netezza SQL and Azure Synapse when migrating:

- `STRPOS`: in Netezza, the `STRPOS` function returns the position of a substring within a string. The equivalent function in Azure Synapse is `CHARINDEX`, with the order of the arguments reversed. For example, `SELECT STRPOS('abcdef','def')...` in Netezza is equivalent to `SELECT CHARINDEX('def','abcdef')...` in Azure Synapse.
- `AGE`: Netezza supports the `AGE` operator to give the interval between two temporal values, such as timestamps or dates. For example, `SELECT AGE('23-03-1956','01-01-2019') FROM...`. In Azure Synapse, `DATEDIFF` gives the interval. For example, `SELECT DATEDIFF(day, '1956-03-26','2019-01-01') FROM...`. Note the date representation sequence.
- `NOW()`: Netezza uses `NOW()` to represent `CURRENT_TIMESTAMP` in Azure Synapse.

Functions, stored procedures, and sequences

> **TIP**
> As part of the preparation phase, assess the number and type of non-data objects being migrated.

When migrating from a mature legacy data warehouse environment such as Netezza, there are often elements other than simple tables and views that need to be migrated to the new target environment. Examples of this include functions, stored procedures, and sequences.

As part of the preparation phase, create an inventory of the objects that need to be migrated and define the methods for handling them. Then assign an appropriate allocation of resources in the project plan.

There may be facilities in the Azure environment that replace the functionality implemented as either functions or stored procedures in the Netezza environment. In this case, it's often more efficient to use the built-in Azure facilities rather than recoding the Netezza functions.

> **TIP**
> Third-party products and services can automate migration of non-data elements.

Microsoft partners offer tools and services that can automate the migration, including the mapping of data types. Also, third-party ETL tools, such as Informatica or Talend, that are already in use in the IBM Netezza

environment can implement any required data transformations.

See the following sections for more information on each of these elements.

Functions

As with most database products, Netezza supports system functions and user-defined functions within the SQL implementation. When migrating to another database platform such as Azure Synapse, common system functions are available and can be migrated without change. Some system functions may have slightly different syntax, but the required changes can be automated. System functions where there's no equivalent, such as arbitrary user-defined functions, may need to be recoded using the languages available in the target environment. Azure Synapse uses the popular Transact-SQL language to implement user-defined functions. Netezza user-defined functions are coded in nzlua or C++ languages.

Stored procedures

Most modern database products allow for procedures to be stored within the database. Netezza provides the NZPLSQL language, which is based on Postgres PL/pgSQL. A stored procedure typically contains SQL statements and some procedural logic, and may return data or a status.

Azure Synapse Analytics also supports stored procedures using T-SQL, so if you must migrate stored procedures, recode them accordingly.

Sequences

In Netezza, a sequence is a named database object created via `CREATE SEQUENCE` that can provide the unique value via the `NEXT VALUE FOR` method. Use these to generate unique numbers for use as surrogate key values for primary key values.

In Azure Synapse, there's no `CREATE SEQUENCE`. Sequences are handled using IDENTITY to create surrogate keys or managed identity using SQL code to create the next sequence number in a series.

Use EXPLAIN to validate legacy SQL

> **TIP**
> Find potential migration issues by using real queries from the existing system query logs.

Capture some representative SQL statements from the legacy query history logs to evaluate legacy Netezza SQL for compatibility with Azure Synapse. Then prefix those queries with `EXPLAIN` and—assuming a "like-for-like" migrated data model in Azure Synapse with the same table and column names—run those `EXPLAIN` statements in Azure Synapse. Any incompatible SQL will return an error. Use this information to determine the scale of the recoding task. This approach doesn't require data to be loaded into the Azure environment, only that the relevant tables and views have been created.

IBM Netezza to T-SQL mapping

The IBM Netezza to T-SQL compliant with Azure Synapse SQL data type mapping is in this table:

IBM NETEZZA DATA TYPE	AZURE SYNAPSE SQL DATA TYPE
array	*Not supported*
bigint	bigint
binary large object [(n[K|M|G])]	nvarchar [(n|max)]
blob [(n[K|M|G])]	nvarchar [(n|max)]
byte [(n)]	binary [(n)]|varbinary(max)

IBM NETEZZA DATA TYPE	AZURE SYNAPSE SQL DATA TYPE
byteint	smallint
char varying [(n)]	varchar [(n|max)]
character varying [(n)]	varchar [(n|max)]
char [(n)]	char [(n)]|varchar(max)
character [(n)]	char [(n)]|varchar(max)
character large object [(n[K|M|G])]	varchar [(n|max)
clob [(n[K|M|G])]	varchar [(n|max)
dataset	*Not supported*
date	date
dec [(p[,s])]	decimal [(p[,s])]
decimal [(p[,s])]	decimal [(p[,s])]
double precision	float(53)
float [(n)]	float [(n)]
graphic [(n)]	nchar [(n)]| varchar(max)
interval	*Not supported*
json [(n)]	nvarchar [(n|max)]
long varchar	nvarchar(max)
long vargraphic	nvarchar(max)
mbb	*Not supported*
mbr	*Not supported*
number [((p|*)[,s])]	numeric [(p[,s])]
numeric [(p [,s])]	numeric [(p[,s])]
period	*Not supported*
real	real
smallint	smallint

IBM NETEZZA DATA TYPE	AZURE SYNAPSE SQL DATA TYPE
st_geometry	*Not supported*
time	time
time with time zone	datetimeoffset
timestamp	datetime2
timestamp with time zone	datetimeoffset
varbyte	varbinary [(n|max)]
varchar [(n)]	varchar [(n)]
vargraphic [(n)]	nvarchar [(n|max)]
varray	*Not supported*
xml	*Not supported*
xmltype	*Not supported*

Summary

Typical existing legacy Netezza installations are implemented in a way that makes migration to Azure Synapse easy. They use SQL for analytical queries on large data volumes, and are in some form of dimensional data model. These factors make them good candidates for migration to Azure Synapse.

To minimize the task of migrating the actual SQL code, follow these recommendations:

- Initial migration of the data warehouse should be as-is to minimize risk and time taken, even if the eventual final environment will incorporate a different data model such as data vault.
- Understand the differences between Netezza SQL implementation and Azure Synapse.
- Use metadata and query logs from the existing Netezza implementation to assess the impact of the differences and plan an approach to mitigate.
- Automate the process wherever possible to minimize errors, risk, and time for the migration.
- Consider using specialist Microsoft partners and services to streamline the migration.

Next steps

To learn more about Microsoft and third-party tools, see the next article in this series: Tools for Netezza data warehouse migration to Azure Synapse Analytics.

Tools for Netezza data warehouse migration to Azure Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

This article is part six of a seven-part series that provides guidance on how to migrate from Netezza to Azure Synapse Analytics. The focus of this article is best practices for Microsoft and third-party tools.

Data warehouse migration tools

By migrating your existing data warehouse to Azure Synapse, you benefit from:

- A globally secure, scalable, low-cost, cloud-native, pay-as-you-use analytical database.
- The rich Microsoft analytical ecosystem that exists on Azure. This ecosystem consists of technologies to help modernize your data warehouse once it's migrated and extend your analytical capabilities to drive new value.

Several tools from both Microsoft and third-party partners can help you migrate your existing data warehouse to Azure Synapse. This article discusses the following types of tools:

- Microsoft data and database migration tools.
- Third-party data warehouse automation tools to automate and document the migration to Azure Synapse.
- Third-party data warehouse migration tools to migrate schema and data to Azure Synapse.
- Third-party tools to bridge the SQL differences between your existing data warehouse DBMS and Azure Synapse.

Microsoft data migration tools

Microsoft offers several tools to help you migrate your existing data warehouse to Azure Synapse, such as:

- Azure Data Factory.
- Microsoft services for physical data transfer.
- Microsoft services for data ingestion.

The next sections discuss these tools in more detail.

Microsoft Azure Data Factory

Data Factory is a fully managed, pay-as-you-use, hybrid data integration service for highly scalable ETL and ELT processing. It uses Apache Spark to process and analyze data in parallel and in-memory to maximize throughput.

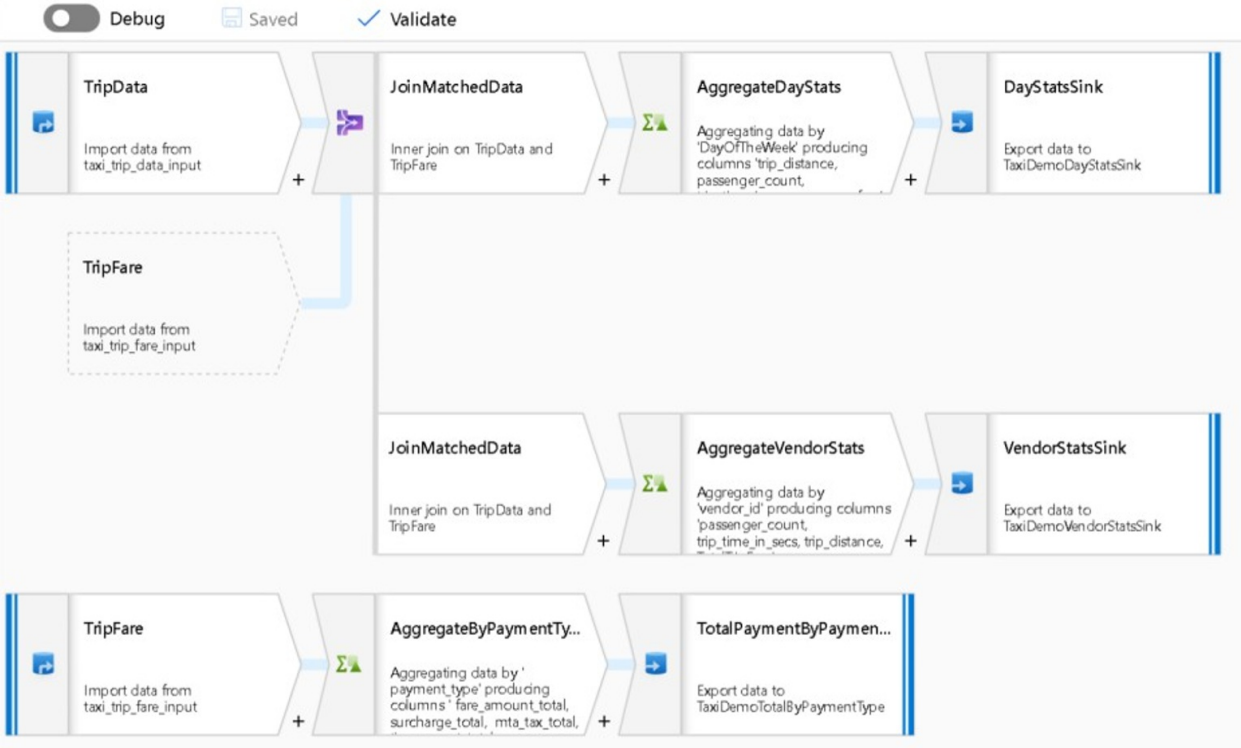
> **TIP**
> Data Factory allows you to build scalable data integration pipelines code-free.

Data Factory connectors support connections to external data sources and databases and include templates for common data integration tasks. A visual front-end, browser-based UI enables non-programmers to create and

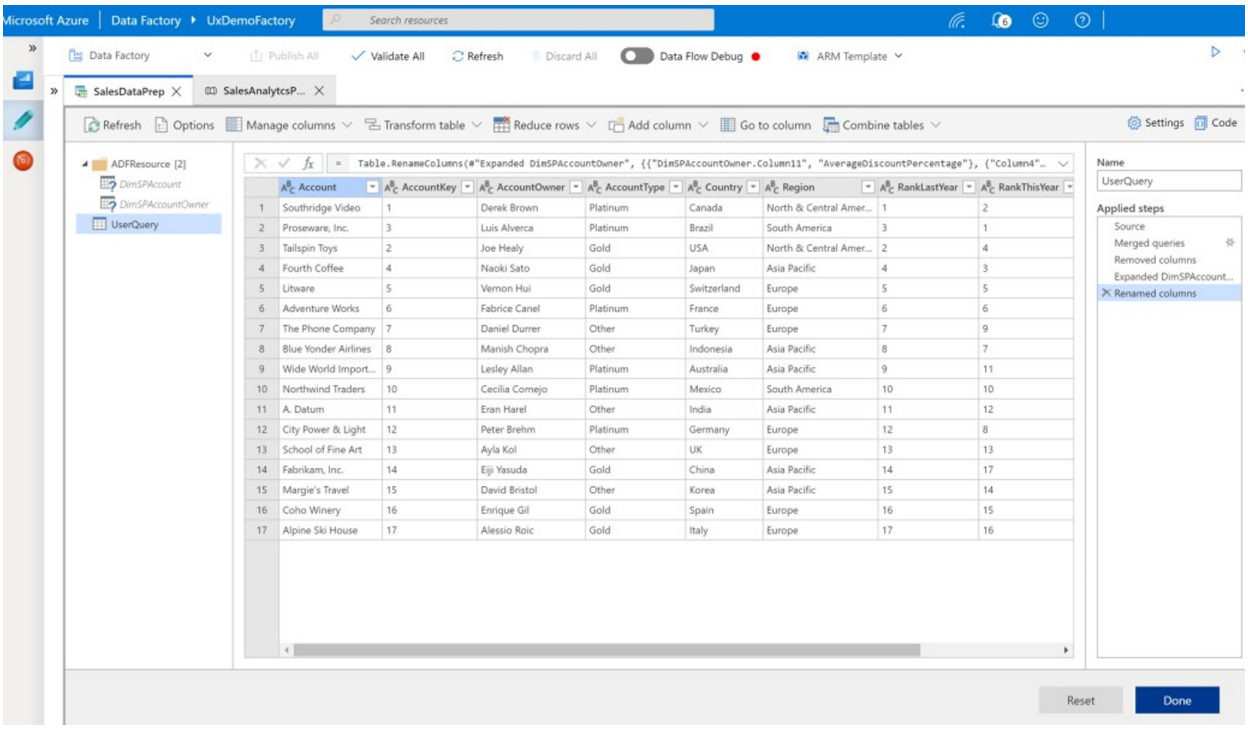
run pipelines to ingest, transform, and load data. More experienced programmers can incorporate custom code, such as Python programs.

> **TIP**
> Data Factory enables collaborative development between business and IT professionals.

Data Factory is also an orchestration tool and is the best Microsoft tool to automate the end-to-end migration process. Automation reduces the risk, effort, and time to migrate, and makes the migration process easily repeatable. The following diagram shows a mapping data flow in Data Factory.



The next screenshot shows a wrangling data flow in Data Factory.



In Data Factory, you can develop simple or comprehensive ETL and ELT processes without coding or

maintenance with just a few clicks. ETL/ELT processes ingest, move, prepare, transform, and process your data. You can design and manage scheduling and triggers in Data Factory to build an automated data integration and loading environment. In Data Factory, you can define, manage, and schedule PolyBase bulk data load processes.

> **TIP**
> Data Factory includes tools to help migrate both your data and your entire data warehouse to Azure.

You can use Data Factory to implement and manage a hybrid environment with on-premises, cloud, streaming, and SaaS data in a secure and consistent way. SaaS data might come from applications such as Salesforce.

Wrangling data flows is a new capability in Data Factory. This capability opens up Data Factory to business users who want to visually discover, explore, and prepare data at scale without writing code. Wrangling data flows offer self-service data preparation, similar to Microsoft Excel, Power Query, and Microsoft Power BI dataflows. Business users can prepare and integrate data through a spreadsheet-style UI with drop-down transform options.

Data Factory is the recommended approach for implementing data integration and ETL/ELT processes in the Azure Synapse environment, especially if you want to refactor existing legacy processes.

Microsoft services for physical data transfer

The following sections discuss a range of products and services that Microsoft offers to assist customers with data transfer.

Azure ExpressRoute

Azure ExpressRoute creates private connections between Azure data centers and infrastructure on your premises or in a collocation environment. ExpressRoute connections don't go over the public internet, and offer more reliability, faster speeds, and lower latencies than typical internet connections. In some cases, you gain significant cost benefits by using ExpressRoute connections to transfer data between on-premises systems and Azure.

AzCopy

AzCopy is a command line utility that copies files to Azure Blob Storage over a standard internet connection. In a warehouse migration project, you can use AzCopy to upload extracted, compressed, delimited text files before loading them into Azure Synapse using PolyBase. AzCopy can upload individual files, file selections, or file folders. If the exported files are in Parquet format, use a native Parquet reader instead.

Azure Data Box

Azure Data Box is a Microsoft service that provides you with a proprietary physical storage device that you can copy migration data onto. You then ship the device to an Azure data center for data upload to cloud storage. This service can be cost-effective for large volumes of data, such as tens or hundreds of terabytes, or where network bandwidth isn't readily available. Azure Data Box is typically used for a large one-off historical data load into Azure Synapse.

Azure Data Box Gateway

Azure Data Box Gateway is a virtualized cloud storage gateway device that resides on your premises and sends your images, media, and other data to Azure. Use Data Box Gateway for one-off migration tasks or ongoing incremental data uploads.

Microsoft services for data ingestion

The following sections discuss the products and services that Microsoft offers to assist customers with data ingestion.

COPY INTO

The COPY INTO statement provides the most flexibility for high-throughput data ingestion into Azure Synapse. For more information about `COPY INTO` capabilities, see COPY (Transact-SQL).

PolyBase

PolyBase is the fastest, most scalable method for bulk data load into Azure Synapse. PolyBase uses the massively parallel processing (MPP) architecture of Azure Synapse for parallel loading of data to achieve the fastest throughput. PolyBase can read data from flat files in Azure Blob Storage, or directly from external data sources and other relational databases via connectors.

> **TIP**
> PolyBase can load data in parallel from Azure Blob Storage into Azure Synapse.

PolyBase can also directly read from files compressed with gzip to reduce the physical volume of data during a load process. PolyBase supports popular data formats such as delimited text, ORC, and Parquet.

> **TIP**
> You can invoke PolyBase from Data Factory as part of a migration pipeline.

PolyBase is tightly integrated with Data Factory to support rapid development of data load ETL/ELT processes. You can schedule data load processes through a visual UI for higher productivity and fewer errors than hand-written code. Microsoft recommends PolyBase for data ingestion into Azure Synapse, especially for high-volume data ingestion.

PolyBase uses `CREATE TABLE AS` or `INSERT...SELECT` statements to load data. `CREATE TABLE AS` minimizes logging to achieve the highest throughput. The most efficient input format for data load is compressed delimited text files. For maximum throughput, split large input files into multiple smaller files and load them in parallel. For fastest loading to a staging table, define the target table as `HEAP` type and use round-robin distribution.

PolyBase has some limitations, it requires data row length to be less than 1 megabyte and doesn't support fixed-width nested formats like JSON and XML.

Microsoft partners for Netezza migrations

Microsoft partners offer tools, services, and expertise to help you migrate your legacy on-premises data warehouse platform to Azure Synapse.

Next steps

To learn more about implementing modern data warehouses, see the next article in this series: Beyond Netezza migration, implement a modern data warehouse in Microsoft Azure.

Beyond Netezza migration, implement a modern data warehouse in Microsoft Azure

8/19/2022 • 23 minutes to read • Edit Online

This article is part seven of a seven-part series that provides guidance on how to migrate from Netezza to Azure Synapse Analytics. The focus of this article is best practices for implementing modern data warehouses.

Beyond data warehouse migration to Azure

A key reason to migrate your existing data warehouse to Azure Synapse Analytics is to utilize a globally secure, scalable, low-cost, cloud-native, pay-as-you-use analytical database. With Azure Synapse, you can integrate your migrated data warehouse with the complete Microsoft Azure analytical ecosystem to take advantage of other Microsoft technologies and modernize your migrated data warehouse. Those technologies include:

- Azure Data Lake Storage for cost effective data ingestion, staging, cleansing, and transformation. Data Lake Storage can free up the data warehouse capacity occupied by fast-growing staging tables.
- Azure Data Factory for collaborative IT and self-service data integration with connectors to cloud and on-premises data sources and streaming data.
- Common Data Model to share consistent trusted data across multiple technologies, including:
  - Azure Synapse
  - Azure Synapse Spark
  - Azure HDInsight
  - Power BI
  - Adobe Customer Experience Platform
  - Azure IoT
  - Microsoft ISV partners
- Microsoft data science technologies, including:
  - Azure Machine Learning studio
  - Azure Machine Learning
  - Azure Synapse Spark (Spark as a service)
  - Jupyter Notebooks
  - RStudio
  - ML.NET
  - .NET for Apache Spark, which lets data scientists use Azure Synapse data to train machine learning models at scale.
- Azure HDInsight to process large amounts of data, and to join big data with Azure Synapse data by creating a logical data warehouse using PolyBase.
- Azure Event Hubs, Azure Stream Analytics, and Apache Kafka to integrate live streaming data from Azure Synapse.

The growth of big data has led to an acute demand for machine learning to enable custom-built, trained machine learning models for use in Azure Synapse. Machine learning models enable in-database analytics to run at scale in batch, on an event-driven basis and on-demand. The ability to take advantage of in-database analytics in Azure Synapse from multiple BI tools and applications also guarantees consistent predictions and

recommendations.

In addition, you can integrate Azure Synapse with Microsoft partner tools on Azure to shorten time to value.

Let's take a closer look at how you can take advantage of technologies in the Microsoft analytical ecosystem to modernize your data warehouse after you've migrated to Azure Synapse.

Offload data staging and ETL processing to Data Lake Storage and Data Factory

Digital transformation has created a key challenge for enterprises by generating a torrent of new data for capture and analysis. A good example is transaction data created by opening online transactional processing (OLTP) systems to service access from mobile devices. Much of this data finds its way into data warehouses, and OLTP systems are the main source. With customers now driving the transaction rate rather than employees, the volume of data in data warehouse staging tables has been growing rapidly.

With the rapid influx of data into the enterprise, along with new sources of data like Internet of Things (IoT), companies must find ways to scale up data integration ETL processing. One method is to offload ingestion, data cleansing, transformation, and integration to a data lake and process data at scale there, as part of a data warehouse modernization program.

Once you've migrated your data warehouse to Azure Synapse, Microsoft can modernize your ETL processing by ingesting and staging data in Data Lake Storage. You can then clean, transform, and integrate your data at scale using Data Factory before loading it into Azure Synapse in parallel using PolyBase.

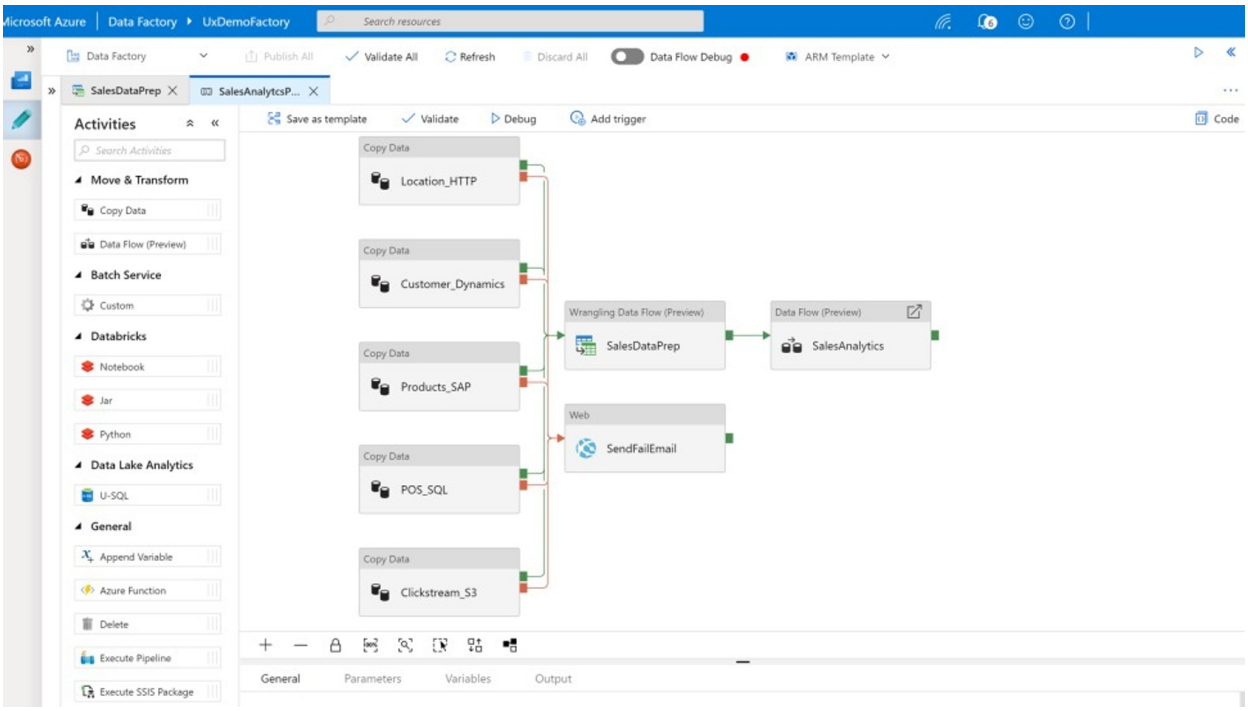
For ELT strategies, consider offloading ELT processing to Data Lake Storage to easily scale as your data volume or frequency grows.

Microsoft Azure Data Factory

Azure Data Factory is a pay-as-you-use, hybrid data integration service for highly scalable ETL and ELT processing. Data Factory provides a web-based UI to build data integration pipelines with no code. With Data Factory, you can:

- Build scalable data integration pipelines code-free.
- Easily acquire data at scale.
- Pay only for what you use.
- Connect to on-premises, cloud, and SaaS-based data sources.
- Ingest, move, clean, transform, integrate, and analyze cloud and on-premises data at scale.
- Seamlessly author, monitor, and manage pipelines that span data stores both on-premises and in the cloud.
- Enable pay-as-you-go scale-out in alignment with customer growth.

You can use these features without writing any code, or you can add custom code to Data Factory pipelines. The following screenshot shows an example Data Factory pipeline.



TIP

Data Factory lets you to build scalable data integration pipelines without code.

Implement Data Factory pipeline development from any of several places, including:

- Microsoft Azure portal.
- Microsoft Azure PowerShell.
- Programmatically from .NET and Python using a multi-language SDK.
- Azure Resource Manager (ARM) templates.
- REST APIs.

TIP

Data Factory can connect to on-premises, cloud, and SaaS data.

Developers and data scientists who prefer to write code can easily author Data Factory pipelines in Java, Python, and .NET using the software development kits (SDKs) available for those programming languages. Data Factory pipelines can be hybrid data pipelines because they can connect, ingest, clean, transform, and analyze data in on-premises data centers, Microsoft Azure, other clouds, and SaaS offerings.

After you develop Data Factory pipelines to integrate and analyze data, you can deploy those pipelines globally and schedule them to run in batch, invoke them on demand as a service, or run them in real-time on an event-driven basis. A Data Factory pipeline can also run on one or more execution engines and monitor execution to ensure performance and to track errors.

TIP

In Azure Data Factory, pipelines control the integration and analysis of data. Data Factory is enterprise-class data integration software aimed at IT professionals and has data wrangling capability for business users.

Use cases

Data Factory supports multiple use cases, such as:

- Prepare, integrate, and enrich data from cloud and on-premises data sources to populate your migrated data warehouse and data marts on Microsoft Azure Synapse.
- Prepare, integrate, and enrich data from cloud and on-premises data sources to produce training data for use in machine learning model development and in retraining analytical models.
- Orchestrate data preparation and analytics to create predictive and prescriptive analytical pipelines for processing and analyzing data in batch, such as sentiment analytics. Either act on the results of the analysis or populate your data warehouse with the results.
- Prepare, integrate, and enrich data for data-driven business applications running on the Azure cloud on top of operational data stores such as Azure Cosmos DB.

> **TIP**
> Build training data sets in data science to develop machine learning models.

Data sources

Data Factory lets you use connectors from both cloud and on-premises data sources. Agent software, known as a *self-hosted integration runtime*, securely accesses on-premises data sources and supports secure, scalable data transfer.

Transform data using Azure Data Factory

Within a Data Factory pipeline, you can ingest, clean, transform, integrate, and analyze any type of data from these sources. Data can be structured, semi-structured like JSON or Avro, or unstructured.

Without writing any code, professional ETL developers can use Data Factory mapping data flows to filter, split, join several types, lookup, pivot, unpivot, sort, union, and aggregate data. In addition, Data Factory supports surrogate keys, multiple write processing options like insert, upsert, update, table recreation, and table truncation, and several types of target data stores—also known as sinks. ETL developers can also create aggregations, including time-series aggregations that require a window to be placed on data columns.

> **TIP**
> Professional ETL developers can use Data Factory mapping data flows to clean, transform, and integrate data without the need to write code.

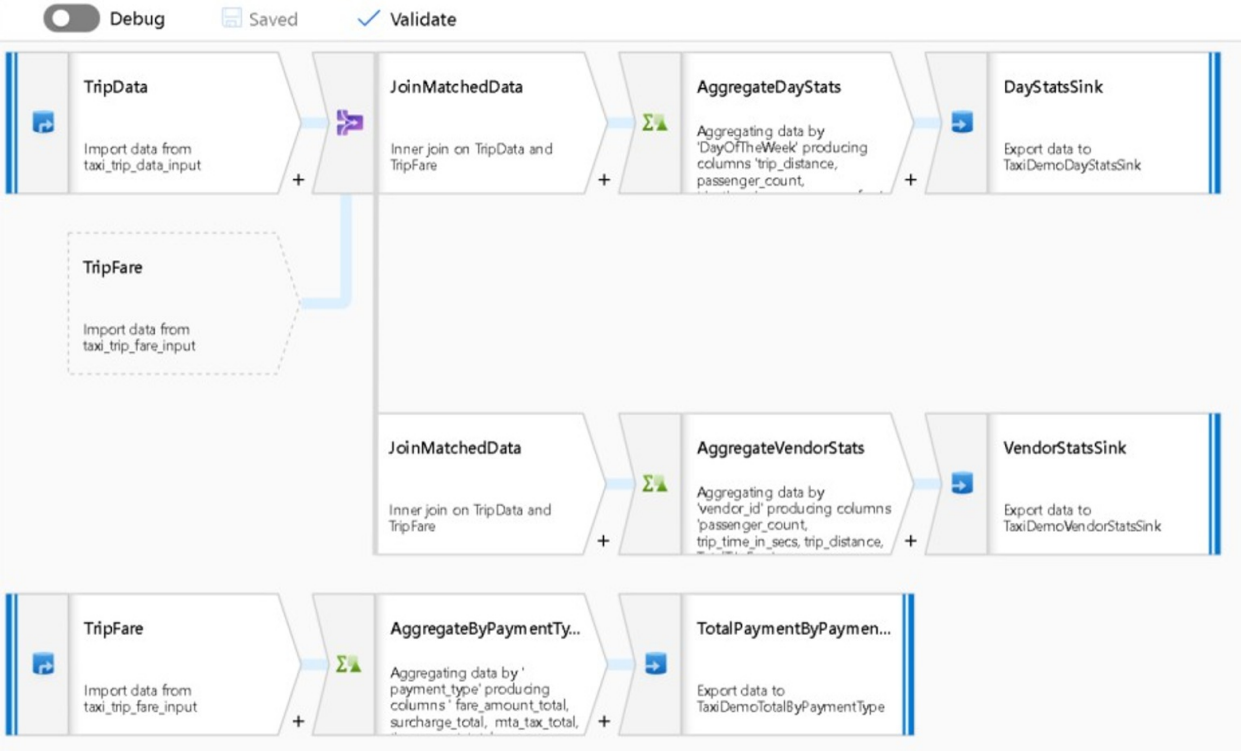
You can run mapping data flows that transform data as activities in a Data Factory pipeline, and if necessary, you can include multiple mapping data flows in a single pipeline. In this way, you can manage the complexity by breaking up challenging data transformation and integration tasks into smaller mapping dataflows that can be combined. And, you can add custom code when needed. In addition to this functionality, Data Factory mapping data flows include the ability to:

- Define expressions to clean and transform data, compute aggregations, and enrich data. For example, these expressions can perform feature engineering on a date field to break it into multiple fields to create training data during machine learning model development. You can construct expressions from a rich set of functions that include mathematical, temporal, split, merge, string concatenation, conditions, pattern match, replace, and many other functions.
- Automatically handle schema drift so that data transformation pipelines can avoid being impacted by schema changes in data sources. This ability is especially important for streaming IoT data, where schema changes can happen without notice if devices are upgraded or when readings are missed by gateway devices collecting IoT data.

- Partition data to enable transformations to run in parallel at scale.
- Inspect streaming data to view the metadata of a stream you're transforming.

> **TIP**
> Data Factory supports the ability to automatically detect and manage schema changes in inbound data, such as in streaming data.

The following screenshot shows an example Data Factory mapping data flow.



Data engineers can profile data quality and view the results of individual data transforms by enabling debug capability during development.

> **TIP**
> Data Factory can also partition data to enable ETL processing to run at scale.

If necessary, you can extend Data Factory transformational and analytical functionality by adding a linked service that contains your code into a pipeline. For example, an Azure Synapse Spark pool notebook might contain Python code that uses a trained model to score the data integrated by a mapping data flow.

You can store integrated data and any results from analytics within a Data Factory pipeline in one or more data stores, such as Data Lake Storage, Azure Synapse, or Hive tables in HDInsight. You can also invoke other activities to act on insights produced by a Data Factory analytical pipeline.

> **TIP**
> Data Factory pipelines are extensible because Data Factory lets you write your own code and run it as part of a pipeline.

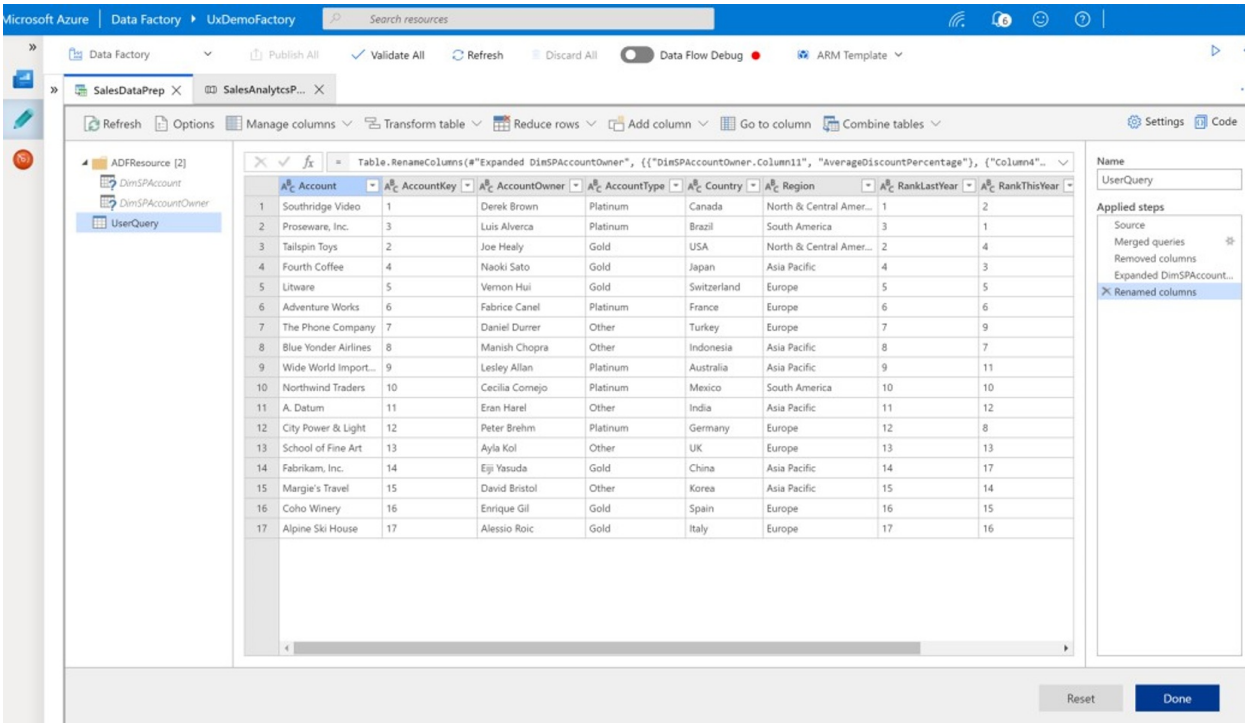
Utilize Spark to scale data integration

At run time, Data Factory internally uses Azure Synapse Spark pools, which are Microsoft's Spark as a service offering, to clean and integrate data in the Azure cloud. You can clean, integrate, and analyze high-volume, high-

velocity data, such as click-stream data, at scale. Microsoft's intention is to also run Data Factory pipelines on other Spark distributions. In addition to running ETL jobs on Spark, Data Factory can invoke Pig scripts and Hive queries to access and transform data stored in HDInsight.

Link self-service data prep and Data Factory ETL processing using wrangling data flows

Data wrangling lets business users, also known as citizen data integrators and data engineers, make use of the platform to visually discover, explore, and prepare data at scale without writing code. This Data Factory capability is easy to use and is similar to Microsoft Excel Power Query or Microsoft Power BI dataflows, where self-service business users use a spreadsheet-style UI with drop-down transforms to prepare and integrate data. The following screenshot shows an example Data Factory wrangling data flow.



Unlike Excel and Power BI, Data Factory wrangling data flows use Power Query to generate M code and then translate it into a massively parallel in-memory Spark job for cloud-scale execution. The combination of mapping data flows and wrangling data flows in Data Factory lets professional ETL developers and business users collaborate to prepare, integrate, and analyze data for a common business purpose. The preceding Data Factory mapping data flows diagram shows how both Data Factory and Azure Synapse Spark pool notebooks can be combined in the same Data Factory pipeline. The combination of mapping and wrangling data flows in Data Factory helps IT and business users stay aware of what data flows each has created, and supports data flow reuse to minimize reinvention and maximize productivity and consistency.

> **TIP**
>
> Data Factory supports both wrangling data flows and mapping data flows, so business users and IT users can integrate data collaboratively on a common platform.

Link data and analytics in analytical pipelines

In addition to cleaning and transforming data, Data Factory can combine data integration and analytics in the same pipeline. You can use Data Factory to create both data integration and analytical pipelines, the latter being an extension of the former. You can drop an analytical model into a pipeline to create an analytical pipeline that generates clean, integrated data for predictions or recommendations. Then, you can act on the predictions or recommendations immediately, or store them in your data warehouse to provide new insights and recommendations that can be viewed in BI tools.

To batch score your data, you can develop an analytical model that you invoke as a service within a Data Factory pipeline. You can develop analytical models code-free with Azure Machine Learning studio, or with the Azure

Machine Learning SDK using Azure Synapse Spark pool notebooks or R in RStudio. When you run Spark machine learning pipelines on Azure Synapse Spark pool notebooks, analysis happens at scale.

You can store integrated data and any Data Factory analytical pipeline results in one or more data stores, such as Data Lake Storage, Azure Synapse, or Hive tables in HDInsight. You can also invoke other activities to act on insights produced by a Data Factory analytical pipeline.

Use a lake database to share consistent trusted data

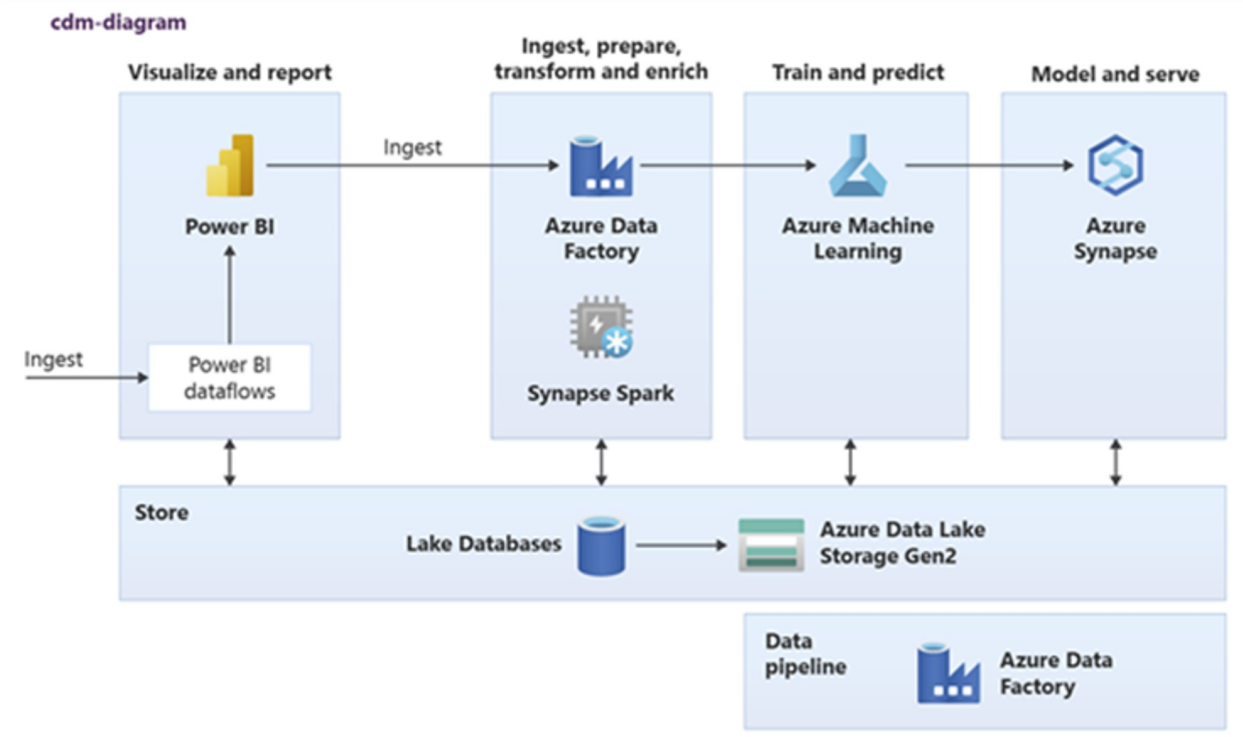
A key objective of any data integration setup is the ability to integrate data once and reuse it everywhere, not just in a data warehouse. For example, you might want to use integrated data in data science. Reuse avoids reinvention and ensures consistent, commonly understood data that everyone can trust.

Common Data Model describes core data entities that can be shared and reused across the enterprise. To achieve reuse, Common Data Model establishes a set of common data names and definitions that describe logical data entities. Examples of common data names include Customer, Account, Product, Supplier, Orders, Payments, and Returns. IT and business professionals can use data integration software to create and store common data assets to maximize their reuse and drive consistency everywhere.

Azure Synapse provides industry-specific database templates to help standardize data in the lake. Lake database templates provide schemas for predefined business areas, enabling data to be loaded into a lake database in a structured way. The power comes when you use data integration software to create lake database common data assets, resulting in self-describing trusted data that can be consumed by applications and analytical systems. You can create common data assets in Data Lake Storage by using Data Factory.

> **TIP**
> Data Lake Storage is shared storage that underpins Microsoft Azure Synapse, Azure Machine Learning, Azure Synapse Spark, and HDInsight.

Power BI, Azure Synapse Spark, Azure Synapse, and Azure Machine Learning can consume common data assets. The following diagram shows how a lake database can be used in Azure Synapse.



> **TIP**
> Integrate data to create lake database logical entities in shared storage to maximize the reuse of common data assets.

Integration with Microsoft data science technologies on Azure

Another key objective when modernizing a data warehouse is to produce insights for competitive advantage. You can produce insights by integrating your migrated data warehouse with Microsoft and third-party data science technologies in Azure. The following sections describe machine learning and data science technologies offered by Microsoft to see how they can be used with Azure Synapse in a modern data warehouse environment.

Microsoft technologies for data science on Azure

Microsoft offers a range of technologies that support advance analysis. With these technologies, you can build predictive analytical models using machine learning or analyze unstructured data using deep learning. The technologies include:

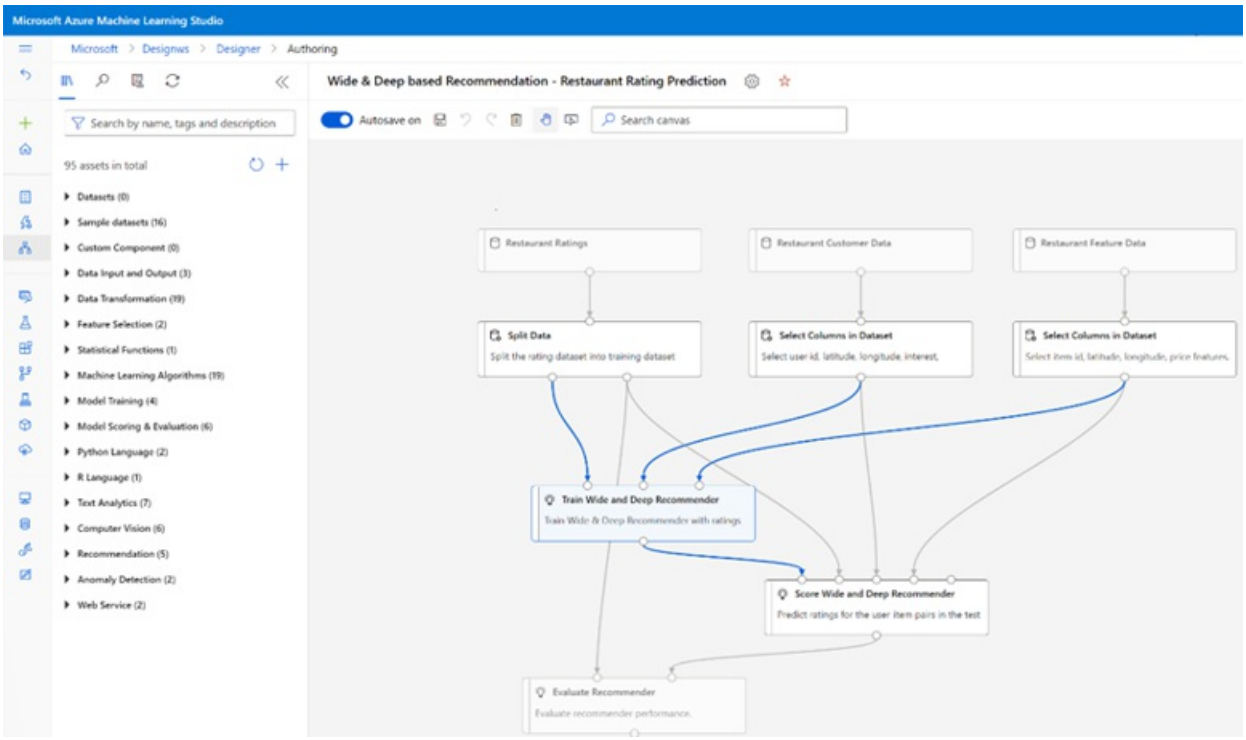
- Azure Machine Learning studio
- Azure Machine Learning
- Azure Synapse Spark pool notebooks
- ML.NET (API, CLI, or ML.NET Model Builder for Visual Studio)
- .NET for Apache Spark

Data scientists can use RStudio (R) and Jupyter Notebooks (Python) to develop analytical models, or they can use frameworks such as Keras or TensorFlow.

> **TIP**
> Develop machine learning models using a no/low-code approach or by using programming languages like Python, R, and .NET.

Azure Machine Learning studio

Azure Machine Learning studio is a fully managed cloud service that lets you build, deploy, and share predictive analytics using a drag-and-drop, web-based UI. The following screenshot shows the Azure Machine Learning studio UI.



Azure Machine Learning

Azure Machine Learning provides an SDK and services for Python that supports can help you quickly prepare data and also train and deploy machine learning models. You can use Azure Machine Learning in Azure notebooks using Jupyter Notebook, with open-source frameworks, such as PyTorch, TensorFlow, scikit-learn, or Spark MLlib—the machine learning library for Spark. Azure Machine Learning provides an AutoML capability that automatically tests multiple algorithms to identify the most accurate algorithms to expedite model development.

> **TIP**
> Azure Machine Learning provides an SDK for developing machine learning models using several open-source frameworks.

You can also use Azure Machine Learning to build machine learning pipelines that manage end-to-end workflow, programmatically scale in the cloud, and deploy models both to the cloud and the edge. Azure Machine Learning contains workspaces, which are logical spaces that you can programmatically or manually create in the Azure portal. These workspaces keep compute targets, experiments, data stores, trained machine learning models, Docker images, and deployed services all in one place to enable teams to work together. You can use Azure Machine Learning in Visual Studio with the Visual Studio for AI extension.

> **TIP**
> Organize and manage related data stores, experiments, trained models, Docker images, and deployed services in workspaces.

Azure Synapse Spark pool notebooks

An Azure Synapse Spark pool notebook is an Azure-optimized Apache Spark service. With Azure Synapse Spark pool notebooks:

- Data engineers can build and run scalable data preparation jobs using Data Factory.
- Data scientists can build and run machine learning models at scale using notebooks written in languages such as Scala, R, Python, Java, and SQL to visualize results.

> **TIP**
> Azure Synapse Spark is a dynamically scalable Spark as a service offering from Microsoft, Spark offers scalable execution of data preparation, model development, and deployed model execution.

Jobs running in Azure Synapse Spark pool notebooks can retrieve, process, and analyze data at scale from Azure Blob Storage, Data Lake Storage, Azure Synapse, HDInsight, and streaming data services such as Apache Kafka.

> **TIP**
> Azure Synapse Spark can access data in a range of Microsoft analytical ecosystem data stores on Azure.

Azure Synapse Spark pool notebooks support autoscaling and auto-termination to reduce total cost of ownership (TCO). Data scientists can use the MLflow open-source framework to manage the machine learning lifecycle.

ML.NET

ML.NET is an open-source, cross-platform machine learning framework for Windows, Linux, macOS. Microsoft created ML.NET so that .NET developers can use existing tools, such as ML.NET Model Builder for Visual Studio, to develop custom machine learning models and integrate them into their .NET applications.

> **TIP**
> Microsoft has extended its machine learning capability to .NET developers.

.NET for Apache Spark

.NET for Apache Spark extends Spark support beyond R, Scala, Python, and Java to .NET and aims to make Spark accessible to .NET developers across all Spark APIs. While .NET for Apache Spark is currently only available on Apache Spark in HDInsight, Microsoft intends to make .NET for Apache Spark available on Azure Synapse Spark pool notebooks.

Use Azure Synapse Analytics with your data warehouse

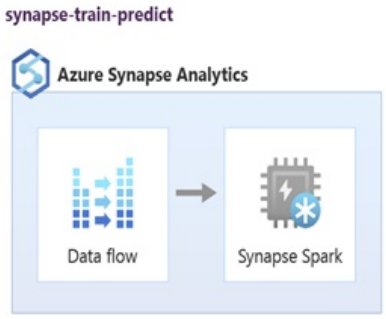
To combine machine learning models with Azure Synapse, you can:

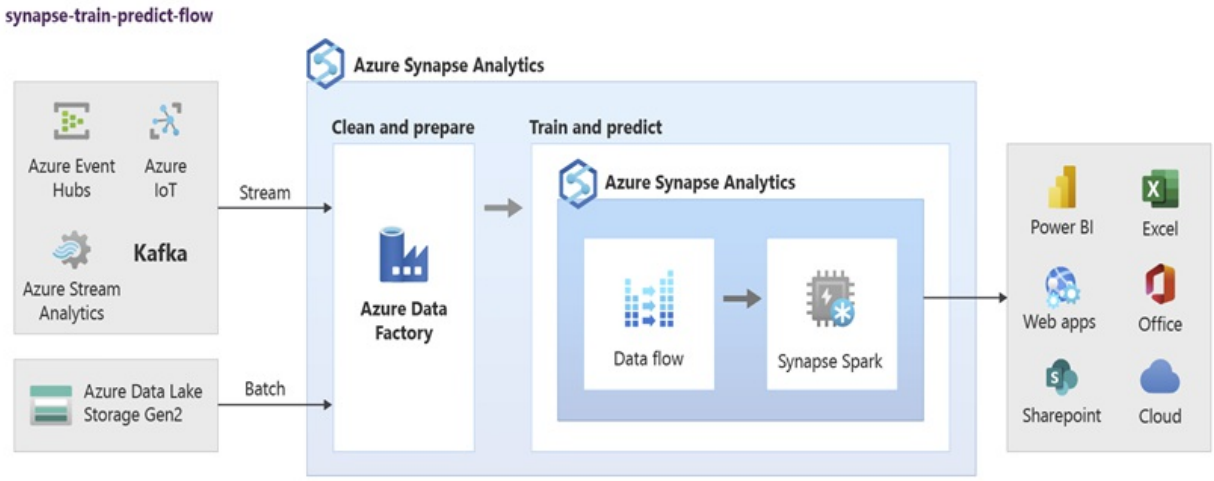
- Use machine learning models in batch or in real-time on streaming data to produce new insights, and add those insights to what you already know in Azure Synapse.
- Use the data in Azure Synapse to develop and train new predictive models for deployment elsewhere, such as in other applications.
- Deploy machine learning models, including models trained elsewhere, in Azure Synapse to analyze data in your data warehouse and drive new business value.

> **TIP**
> Train, test, evaluate, and run machine learning models at scale on Azure Synapse Spark pool notebooks by using data in Azure Synapse.

Data scientists can use RStudio, Jupyter Notebooks, and Azure Synapse Spark pool notebooks together with Azure Machine Learning to develop machine learning models that run at scale on Azure Synapse Spark pool notebooks using data in Azure Synapse. For example, data scientists could create an unsupervised model to segment customers to drive different marketing campaigns. Use supervised machine learning to train a model to predict a specific outcome, such as to predict a customer's propensity to churn, or to recommend the next

best offer for a customer to try to increase their value. The following diagram shows how Azure Synapse can be used for Azure Machine Learning.





In another scenario, you can ingest social network or review website data into Data Lake Storage, then prepare and analyze the data at scale on an Azure Synapse Spark pool notebook using natural language processing to score customer sentiment about your products or brand. You can then add those scores to your data warehouse. By using big data analytics to understand the effect of negative sentiment on product sales, you add to what you already know in your data warehouse.

> **TIP**
> Produce new insights using machine learning on Azure in batch or in real-time and add to what you know in your data warehouse.

Integrate live streaming data into Azure Synapse Analytics

When analyzing data in a modern data warehouse, you must be able to analyze streaming data in real-time and join it with historical data in your data warehouse. An example is combining IoT data with product or asset data.

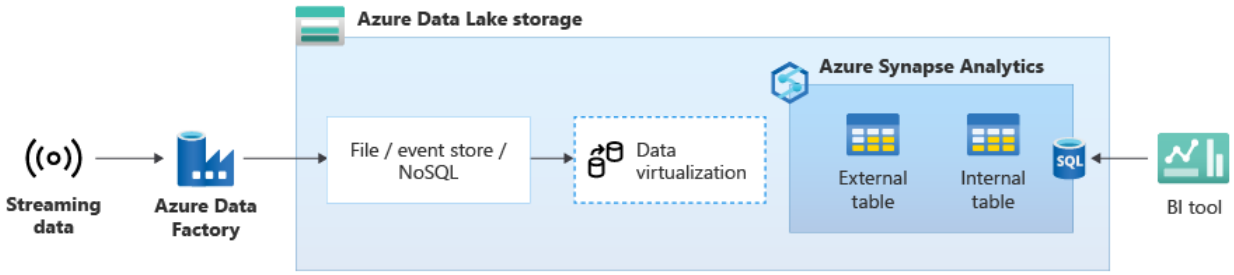
> **TIP**
> Integrate your data warehouse with streaming data from IoT devices or clickstreams.

Once you've successfully migrated your data warehouse to Azure Synapse, you can introduce live streaming data integration as part of a data warehouse modernization exercise by taking advantage of the extra functionality in Azure Synapse. To do so, ingest streaming data via Event Hubs, other technologies like Apache Kafka, or potentially your existing ETL tool if it supports the streaming data sources. Store the data in Data Lake Storage. Then, create an external table in Azure Synapse using PolyBase and point it at the data being streamed into Data Lake Storage so that your data warehouse now contains new tables that provide access to the real-time streaming data. Query the external table as if the data was in the data warehouse by using standard T-SQL

from any BI tool that has access to Azure Synapse. You can also join the streaming data to other tables with historical data to create views that join live streaming data to historical data to make it easier for business users to access the data.

> **TIP**
> Ingest streaming data into Data Lake Storage from Event Hubs or Apache Kafka, and access the data from Azure Synapse using PolyBase external tables.

In the following diagram, a real-time data warehouse on Azure Synapse is integrated with streaming data in Data Lake Storage.



Create a logical data warehouse using PolyBase

With PolyBase, you can create a logical data warehouse to simplify user access to multiple analytical data stores. Many companies have adopted "workload optimized" analytical data stores over the last several years in addition to their data warehouses. The analytical platforms on Azure include:

- Data Lake Storage with Azure Synapse Spark pool notebook (Spark as a service), for big data analytics.
- HDInsight (Hadoop as a service), also for big data analytics.
- NoSQL Graph databases for graph analysis, which could be done in Azure Cosmos DB.
- Event Hubs and Stream Analytics, for real-time analysis of data in motion.

You might have non-Microsoft equivalents of these platforms, or a master data management (MDM) system that needs to be accessed for consistent trusted data on customers, suppliers, products, assets, and more.

> **TIP**
> PolyBase simplifies access to multiple underlying analytical data stores on Azure for ease of access by business users.

Those analytical platforms emerged because of the explosion of new data sources inside and outside the enterprise and the demand by business users to capture and analyze the new data. The new data sources include:

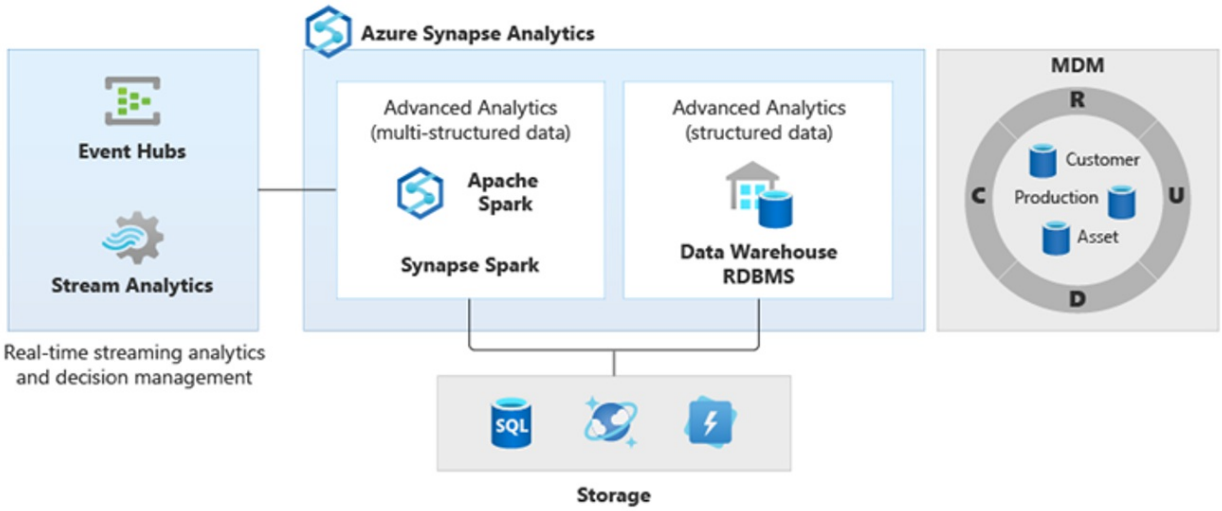
- Machine generated data, such as IoT sensor data and clickstream data.
- Human generated data, such as social network data, review web site data, customer inbound email, images, and video.
- Other external data, such as open government data and weather data.

This new data goes beyond the structured transaction data and main data sources that typically feed data warehouses and often includes:

- Semi-structured data like JSON, XML, or Avro.

- Unstructured data like text, voice, image, or video, which is more complex to process and analyze.
- High volume data, high velocity data, or both.

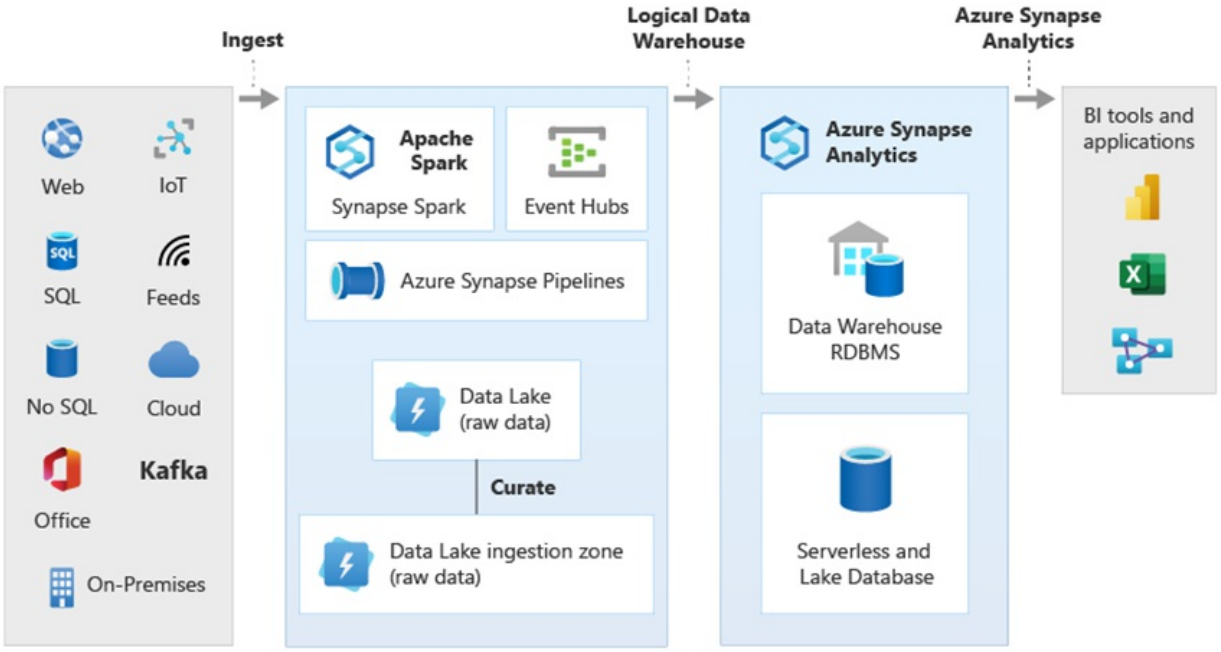
As a result, new more complex kinds of analysis have emerged, such as natural language processing, graph analysis, deep learning, streaming analytics, or complex analysis of large volumes of structured data. These kinds of analysis typically don't happen in a data warehouse, so it's not surprising to see different analytical platforms for different types of analytical workloads, as shown in the following diagram.



> **TIP**
> The ability to make data in multiple analytical data stores look like it's all in one system and join it to Azure Synapse is known as a logical data warehouse architecture.

Because these platforms produce new insights, it's normal to see a requirement to combine the new insights with what you already know in Azure Synapse, which is what PolyBase makes possible.

By using PolyBase data virtualization inside Azure Synapse, you can implement a logical data warehouse where data in Azure Synapse is joined to data in other Azure and on-premises analytical data stores like HDInsight, Azure Cosmos DB, or streaming data flowing into Data Lake Storage from Stream Analytics or Event Hubs. This approach lowers the complexity for users, who access external tables in Azure Synapse and don't need to know that the data they're accessing is stored in multiple underlying analytical systems. The following diagram shows a complex data warehouse structure accessed through comparatively simpler yet still powerful UI methods.



The diagram shows how other technologies in the Microsoft analytical ecosystem can be combined with the capability of the logical data warehouse architecture in Azure Synapse. For example, you can ingest data into Data Lake Storage and curate the data using Data Factory to create trusted data products that represent Microsoft lake database logical data entities. This trusted, commonly understood data can then be consumed and reused in different analytical environments such as Azure Synapse, Azure Synapse Spark pool notebooks, or Azure Cosmos DB. All insights produced in these environments are accessible via a logical data warehouse data virtualization layer made possible by PolyBase.

> **TIP**
> A logical data warehouse architecture simplifies business user access to data and adds new value to what you already know in your data warehouse.

Conclusions

After you migrate your data warehouse to Azure Synapse, you can take advantage of other technologies in the Microsoft analytical ecosystem. By doing so, you not only modernize your data warehouse, but bring insights produced in other Azure analytical data stores into an integrated analytical architecture.

You can broaden your ETL processing to ingest data of any type into Data Lake Storage, and then prepare and integrate the data at scale using Data Factory to produce trusted, commonly understood data assets. Those assets can be consumed by your data warehouse and accessed by data scientists and other applications. You can build real-time and batch oriented analytical pipelines and create machine learning models to run in batch, in real-time on streaming data, and on-demand as a service.

You can use PolyBase or `COPY INTO` to go beyond your data warehouse to simplify access to insights from multiple underlying analytical platforms on Azure. To do so, create holistic integrated views in a logical data warehouse that support access to streaming, big data, and traditional data warehouse insights from BI tools and applications.

By migrating your data warehouse to Azure Synapse, you can take advantage of the rich Microsoft analytical ecosystem running on Azure to drive new value in your business.

Next steps

To learn about migrating to a dedicated SQL pool, see Migrate a data warehouse to a dedicated SQL pool in

Azure Synapse Analytics.

Design and performance for Oracle migrations

8/19/2022 • 26 minutes to read • Edit Online

This article is part one of a seven-part series that provides guidance on how to migrate from Oracle to Azure Synapse Analytics. The focus of this article is best practices for design and performance.

Overview

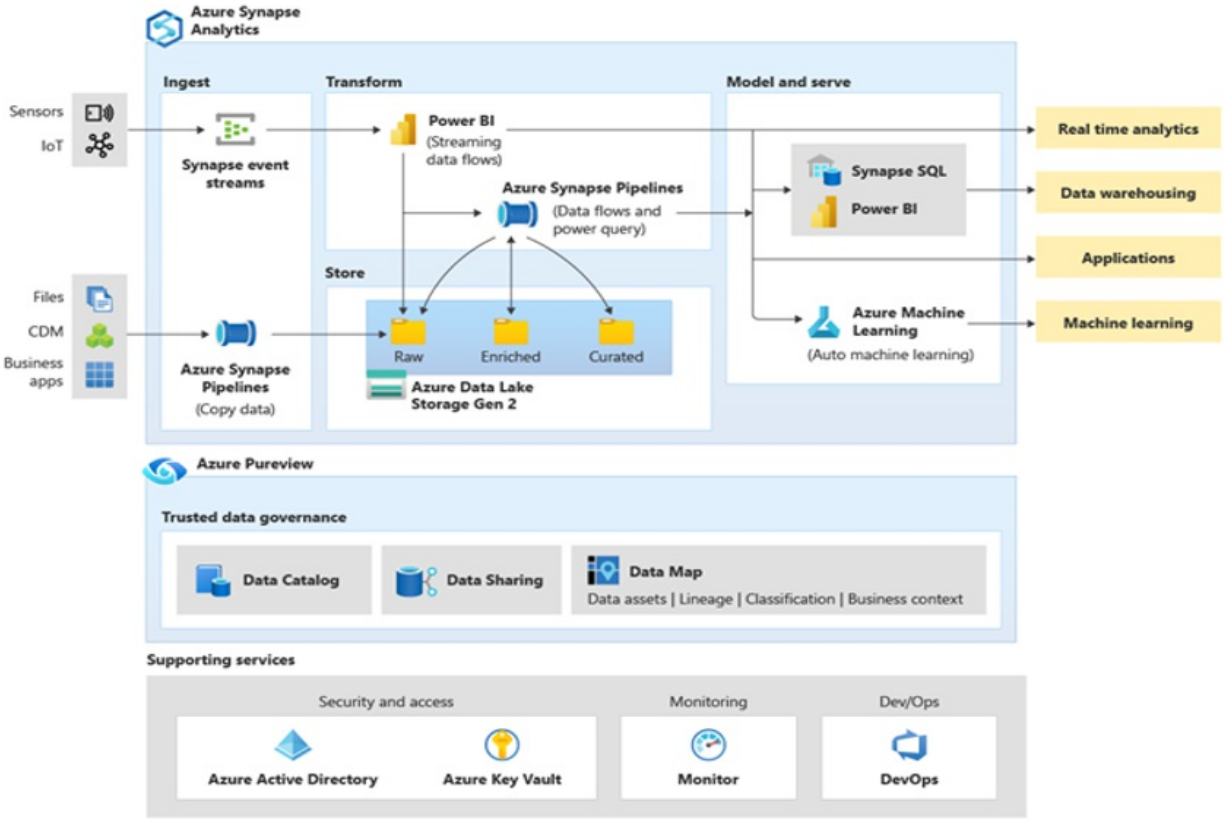
Due to the cost and complexity of maintaining and upgrading legacy on-premises Oracle environments, many existing Oracle users want to take advantage of the innovations provided by modern cloud environments. Infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS) cloud environments let you delegate tasks like infrastructure maintenance and platform development to the cloud provider.

> **TIP**
> More than just a database—the Azure environment includes a comprehensive set of capabilities and tools.

Although Oracle and Azure Synapse Analytics are both SQL databases that use massively parallel processing (MPP) techniques to achieve high query performance on exceptionally large data volumes, there are some basic differences in approach:

- Legacy Oracle systems are often installed on-premises and use relatively expensive hardware, while Azure Synapse is cloud-based and uses Azure storage and compute resources.
- Upgrading an Oracle configuration is a major task involving extra physical hardware and potentially lengthy database reconfiguration, or dump and reload. Because storage and compute resources are separate in the Azure environment and have elastic scaling capability, those resources can be scaled upwards or downwards independently.
- You can pause or resize Azure Synapse as needed to reduce resource utilization and cost.

Microsoft Azure is a globally available, highly secure, scalable cloud environment that includes Azure Synapse and an ecosystem of supporting tools and capabilities. The next diagram summarizes the Azure Synapse ecosystem.



Azure Synapse provides best-of-breed relational database performance by using techniques such as MPP and automatic in-memory caching. You can see the results of these techniques in independent benchmarks such as the one run recently by GigaOm, which compares Azure Synapse to other popular cloud data warehouse offerings. Customers who migrate to the Azure Synapse environment see many benefits, including:

- Improved performance and price/performance.
- Increased agility and shorter time to value.
- Faster server deployment and application development.
- Elastic scalability—only pay for actual usage.
- Improved security/compliance.
- Reduced storage and disaster recovery costs.
- Lower overall TCO, better cost control, and streamlined operational expenditure (OPEX).

To maximize these benefits, migrate new or existing data and applications to the Azure Synapse platform. In many organizations, migration includes moving an existing data warehouse from a legacy on-premises platform, such as Oracle, to Azure Synapse. At a high level, the migration process includes these steps:

Preparation

- Define scope—what is to be migrated.
- Build inventory of data and processes for migration.
- Define data model changes (if any).
- Define source data extract mechanism.
- Identify the appropriate Azure and third-party tools and features to be used.
- Train staff early on the new platform.

- Set up the Azure target platform.

Migration □

- Start small and simple.
- Automate wherever possible.
- Leverage Azure built-in tools and features to reduce migration effort.
- Migrate metadata for tables and views.
- Migrate historical data to be maintained.
- Migrate or refactor stored procedures and business processes.
- Migrate or refactor ETL/ELT incremental load processes.

Post migration

- Monitor and document all stages of the process.
- Use the experience gained to build a template for future migrations.
- Re-engineer the data model if required (using new platform performance and scalability).
- Test applications and query tools.
- Benchmark and optimize query performance.

This article provides general information and guidelines for performance optimization when migrating a data warehouse from an existing Oracle environment to Azure Synapse. The goal of performance optimization is to achieve the same or better data warehouse performance in Azure Synapse after the migration.

Design considerations

Migration scope

When you're preparing to migrate from an Oracle environment, consider the following migration choices.

Choose the workload for the initial migration

Typically, legacy Oracle environments have evolved over time to encompass multiple subject areas and mixed workloads. When you're deciding where to start on a migration project, choose an area where you'll be able to:

- Prove the viability of migrating to Azure Synapse by quickly delivering the benefits of the new environment.
- Allow your in-house technical staff to gain relevant experience with the processes and tools that they'll use when they migrate other areas.
- Create a template for further migrations that's specific to the source Oracle environment and the current tools and processes that are already in place.

A good candidate for an initial migration from an Oracle environment supports the preceding items, and:

- Implements a BI/Analytics workload rather than an online transaction processing (OLTP) workload.
- Has a data model, such as a star or snowflake schema, that can be migrated with minimal modification.

> **TIP**
> Create an inventory of objects that need to be migrated, and document the migration process.

The volume of migrated data in an initial migration should be large enough to demonstrate the capabilities and benefits of the Azure Synapse environment but not too large to quickly demonstrate value. A size in the 1-10 terabyte range is typical.

An initial approach to a migration project is to minimize the risk, effort, and time needed so that you quickly see the benefits of the Azure cloud environment. The following approaches limit the scope of the initial migration to just the data marts and doesn't address broader migration aspects, such as ETL migration and historical data migration. However, you can address those aspects in later phases of the project once the migrated data mart layer is backfilled with data and the required build processes.

Lift and shift migration vs. Phased approach

In general, there are two types of migration regardless of the purpose and scope of the planned migration: lift and shift as-is and a phased approach that incorporates changes.

Lift and shift

In a lift and shift migration, an existing data model, like a star schema, is migrated unchanged to the new Azure Synapse platform. This approach minimizes risk and migration time by reducing the work needed to realize the benefits of moving to the Azure cloud environment. Lift and shift migration is a good fit for these scenarios:

- You have an existing Oracle environment with a single data mart to migrate, or
- You have an existing Oracle environment with data that's already in a well-designed star or snowflake schema, or
- You're under time and cost pressures to move to a modern cloud environment.

> **TIP**
> Lift and shift is a good starting point, even if subsequent phases implement changes to the data model.

Phased approach that incorporates changes

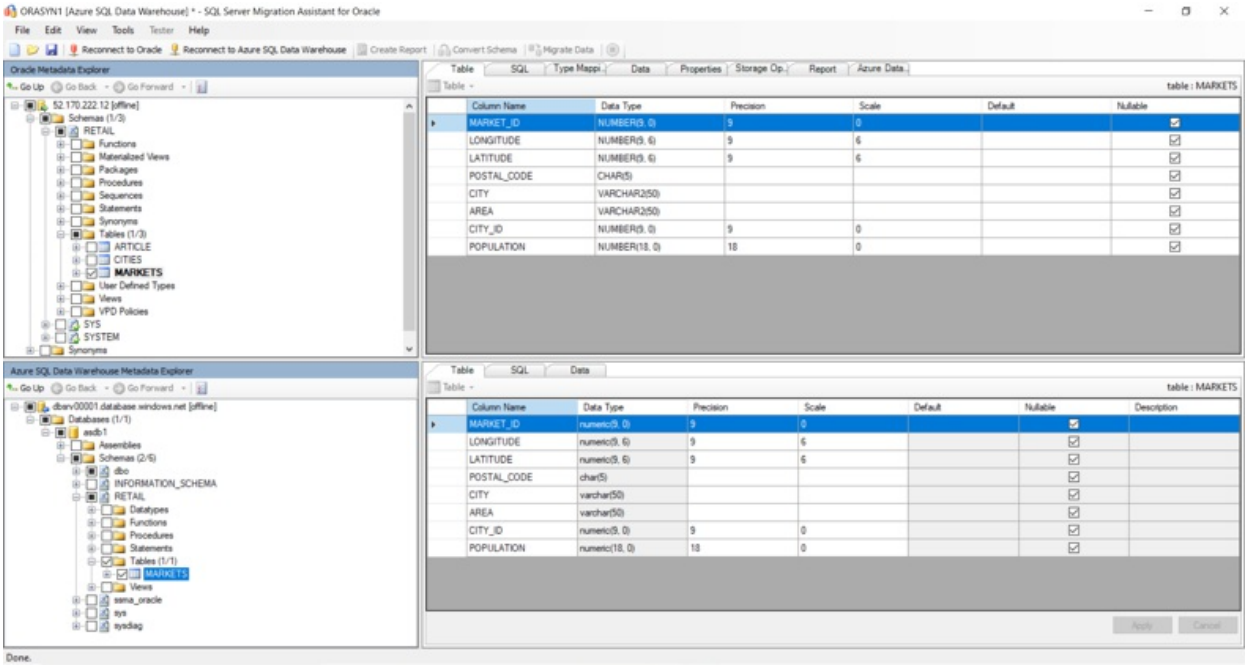
If a legacy data warehouse has evolved over a long period of time, you might need to re-engineer it to maintain the required performance levels. You might also have to re-engineer to support new data like Internet of Things (IoT) streams. As part of the re-engineering process, migrate to Azure Synapse to get the benefits of a scalable cloud environment. Migration can include a change in the underlying data model, such as a move from an Inmon model to a data vault.

Microsoft recommends moving your existing data model as-is to Azure and using the performance and flexibility of the Azure environment to apply the re-engineering changes. That way, you can use Azure's capabilities to make the changes without impacting the existing source system.

Use Microsoft facilities to implement a metadata-driven migration

You can automate and orchestrate the migration process by using the capabilities of the Azure environment. This approach minimizes the performance hit on the existing Oracle environment, which may already be running close to capacity.

The SQL Server Migration Assistant (SSMA) for Oracle can automate many parts of the migration process, including in some cases functions and procedural code. SSMA supports Azure Synapse as a target environment.



SSMA for Oracle can help you migrate an Oracle data warehouse or data mart to Azure Synapse. SSMA is designed to automate the process of migrating tables, views, and data from an existing Oracle environment.

Azure Data Factory is a cloud-based data integration service that supports creating data-driven workflows in the cloud that orchestrate and automate data movement and data transformation. You can use Data Factory to create and schedule data-driven workflows (pipelines) that ingest data from disparate data stores. Data Factory can process and transform data by using compute services such as Azure HDInsight Hadoop, Spark, Azure Data Lake Analytics, and Azure Machine Learning.

Data Factory can be used to migrate data at source to Azure SQL target. This offline data movement helps to reduce migration downtime significantly.

Azure Database Migration Services can help you plan and perform a migration from environments like Oracle.

When you're planning to use Azure facilities to manage the migration process, create metadata that lists all the data tables to be migrated and their location.

Design differences between Oracle and Azure Synapse

As mentioned earlier, there are some basic differences in approach between Oracle and Azure Synapse Analytics databases. SSMA for Oracle not only helps bridge these gaps but also automates the migration. Although SSMA isn't the most efficient approach for very high volumes of data, it's useful for smaller tables.

Multiple databases vs. single database and schemas

The Oracle environment often contains multiple separate databases. For instance, there could be separate databases for: data ingestion and staging tables, core warehouse tables, and data marts—sometimes referred to as the semantic layer. Processing in ETL or ELT pipelines can implement cross-database joins and move data between the separate databases.

In contrast, the Azure Synapse environment contains a single database and uses schemas to separate tables into logically separate groups. We recommend that you use a series of schemas within the target Azure Synapse database to mimic the separate databases migrated from the Oracle environment. If the Oracle environment already uses schemas, you may need to use a new naming convention when you move the existing Oracle tables and views to the new environment. For example, you could concatenate the existing Oracle schema and table names into the new Azure Synapse table name, and use schema names in the new environment to maintain the original separate database names. Although you can use SQL views on top of the underlying tables to maintain the logical structures, there are potential downsides to that approach:

- Views in Azure Synapse are read-only, so any updates to the data must take place on the underlying base

tables.

- There may already be one or more layers of views in existence and adding an extra layer of views could affect performance.

> **TIP**
> Combine multiple databases into a single database within Azure Synapse and use schema names to logically separate the tables.

Table considerations

When you migrate tables between different environments, typically only the raw data and the metadata that describes it physically migrate. Other database elements from the source system, such as indexes, usually aren't migrated because they might be unnecessary or implemented differently in the new environment.

Performance optimizations in the source environment, such as indexes, indicate where you might add performance optimization in the new environment. For example, if queries in the source Oracle environment frequently use bit-mapped indexes, that suggests that a non-clustered index should be created within Azure Synapse. Other native performance optimization techniques like table replication may be more applicable than straight like-for-like index creation. SSMA for Oracle can be used to provide migration recommendations for table distribution and indexing.

> **TIP**
> Existing indexes indicate candidates for indexing in the migrated warehouse.

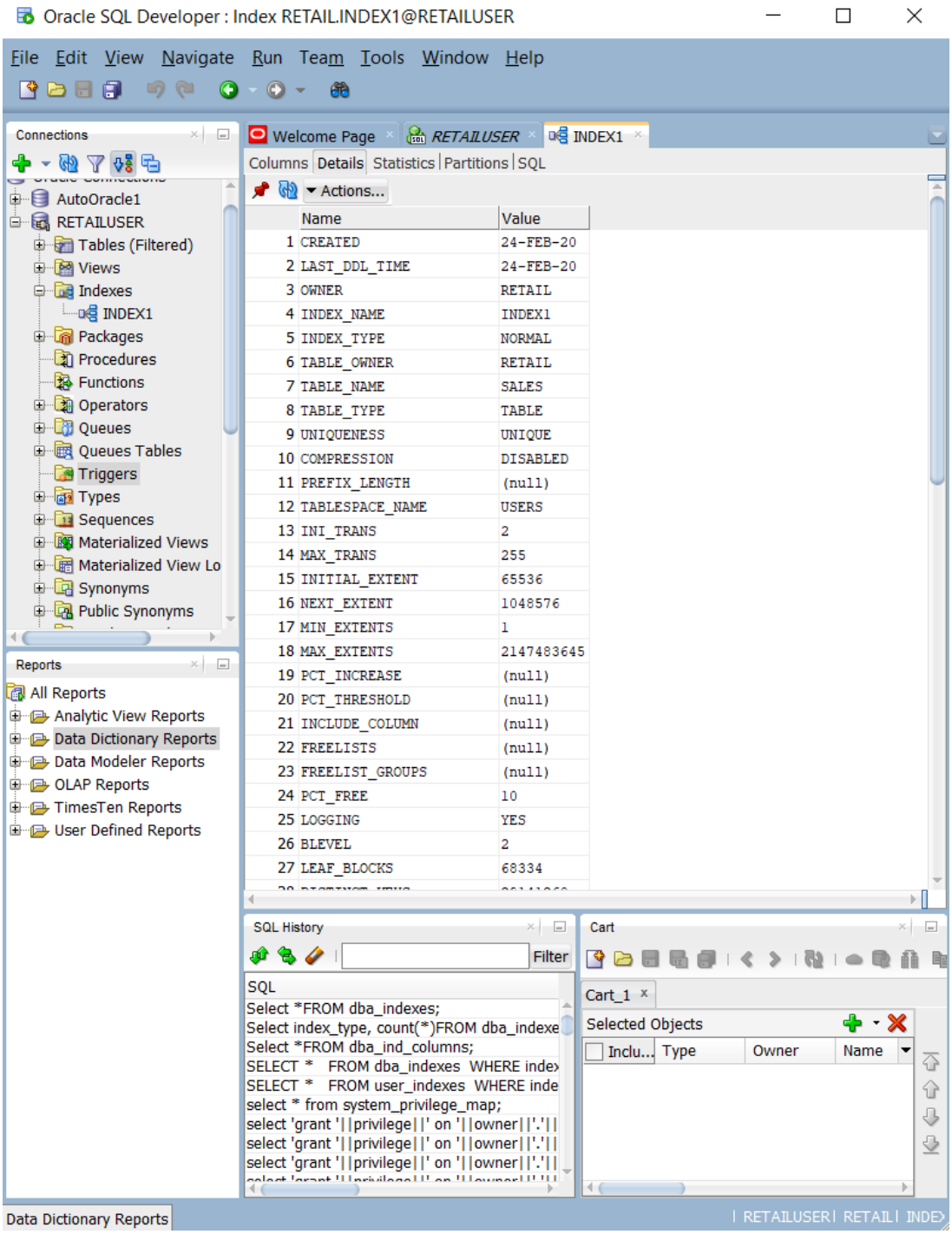
Unsupported Oracle database object types

Oracle-specific features can often be replaced by Azure Synapse features. However, some Oracle database objects aren't directly supported in Azure Synapse. The following list of unsupported Oracle database objects describes how you can achieve an equivalent functionality in Azure Synapse.

- **Various indexing options**: in Oracle, several indexing options, such as bit-mapped indexes, function-based indexes, and domain indexes, have no direct equivalent in Azure Synapse.

  You can find out which columns are indexed and the index type by:

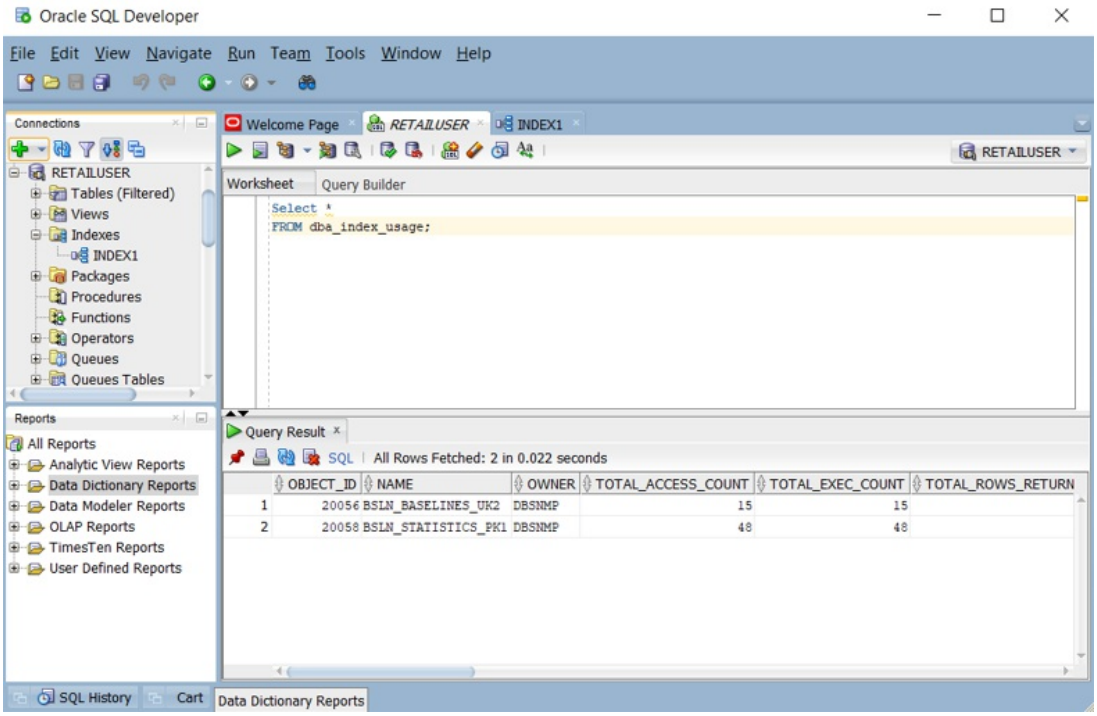
  - Querying system catalog tables and views, such as `ALL_INDEXES`, `DBA_INDEXES`, `USER_INDEXES`, and `DBA_IND_COL`. You can use the built-in queries in Oracle SQL Developer, as shown in the following screenshot.



Or, run the following query to find all indexes of a given type:

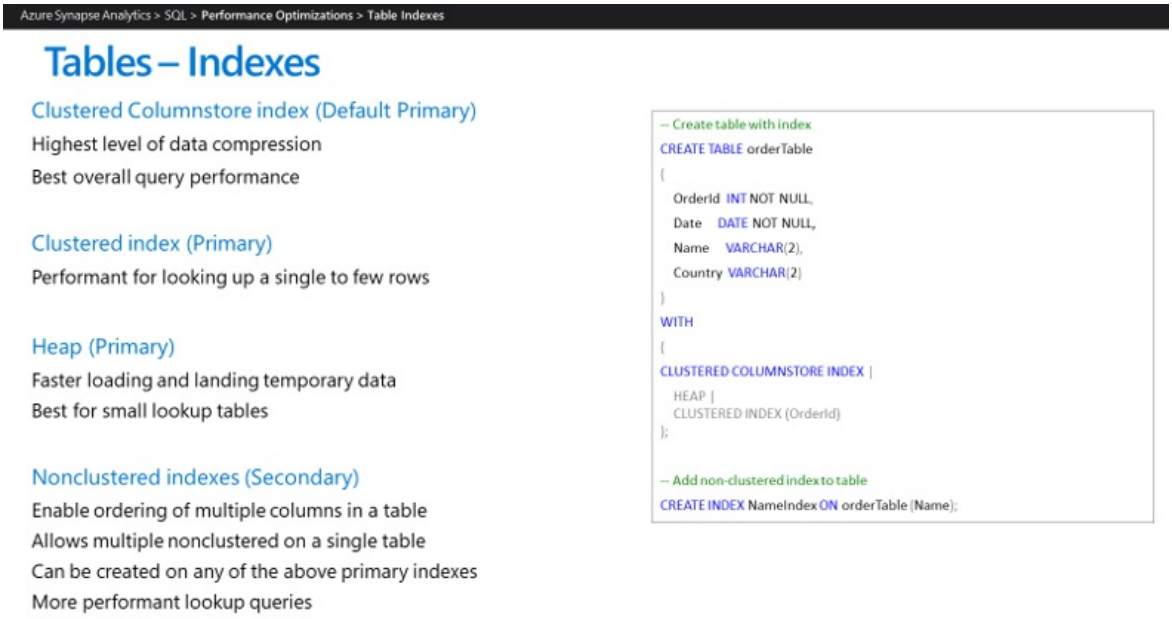
```
SELECT * FROM dba_indexes WHERE index_type LIKE 'FUNCTION-BASED%';
```

- Querying the `dba_index_usage` or `v$object_usage` views when monitoring is enabled. You can query those views in Oracle SQL Developer, as shown in the following screenshot.



Function-based indexes, where the index contains the result of a function on the underlying data columns, have no direct equivalent in Azure Synapse. We recommend that you first migrate the data, then in Azure Synapse run the Oracle queries that use function-based indexes to gauge performance. If the performance of those queries in Azure Synapse isn't acceptable, consider creating a column that contains the pre-calculated value and then index that column.

When you configure the Azure Synapse environment, it makes sense to only implement in-use indexes. Azure Synapse currently supports the index types shown here:



Azure Synapse features, such as parallel query processing and in-memory caching of data and results, make it likely that fewer indexes are required for data warehouse applications to achieve performance goals. We recommend that you use the following index types in Azure Synapse:

- **Clustered columnstore indexes**: when no index options are specified for a table, Azure Synapse by default creates a clustered columnstore index. Clustered columnstore tables offer the highest level of data compression, best overall query performance, and generally outperform clustered index or heap tables. A clustered columnstore index is usually the best choice for large tables.

When you create a table, choose clustered columnstore if you're unsure how to index your table. However, there are some scenarios where clustered columnstore indexes aren't the best option:

  - Tables with varchar(max), nvarchar(max), or varbinary(max) data types, because a clustered columnstore index doesn't support those data types. Instead, consider using a heap or clustered index.
  - Tables with transient data, because columnstore tables might be less efficient than heap or temporary tables.
  - Small tables with less than 100 million rows. Instead, consider using heap tables.
- **Clustered and nonclustered indexes**: clustered indexes can outperform clustered columnstore indexes when a single row needs to be quickly retrieved. For queries where a single row lookup, or just a few row lookups, must perform at extreme speed, consider using a cluster index or nonclustered secondary index. The disadvantage of using a clustered index is that only queries with a highly selective filter on the clustered index column will benefit. To improve filtering on other columns, you can add a nonclustered index to the other columns. However, each index that you add to a table uses more space and increases the processing time to load.

- **Heap tables**: when you're temporarily landing data on Azure Synapse, you might find that using a heap table makes the overall process faster. This is because loading data to heap tables is faster than loading data to index tables, and in some cases subsequent reads can be done from cache. If you're loading data only to stage it before running more transformations, it's much faster to load it to a heap table than a clustered columnstore table. Also, loading data to a temporary table is faster than loading a table to permanent storage. For small lookup tables with less than 100 million rows, heap tables are usually the right choice. Cluster columnstore tables begin to achieve optimal compression when they contain more than 100 million rows.

- **Clustered tables**: Oracle tables can be organized so that table rows that are frequently accessed together (based on a common value) are physically stored together to reduce disk I/O when data is retrieved. Oracle also provides a hash-cluster option for individual tables, which applies a hash value to the cluster key and physically stores rows with the same hash value together. To list clusters within an Oracle database, use the `SELECT * FROM DBA_CLUSTERS;` query. To determine whether a table is within a cluster, use the `SELECT * FROM TAB;` query, which shows the table name and cluster ID for each table.

  In Azure Synapse, you can achieve similar results by using materialized and/or replicated tables, because those table types minimize the I/O required at query run time.

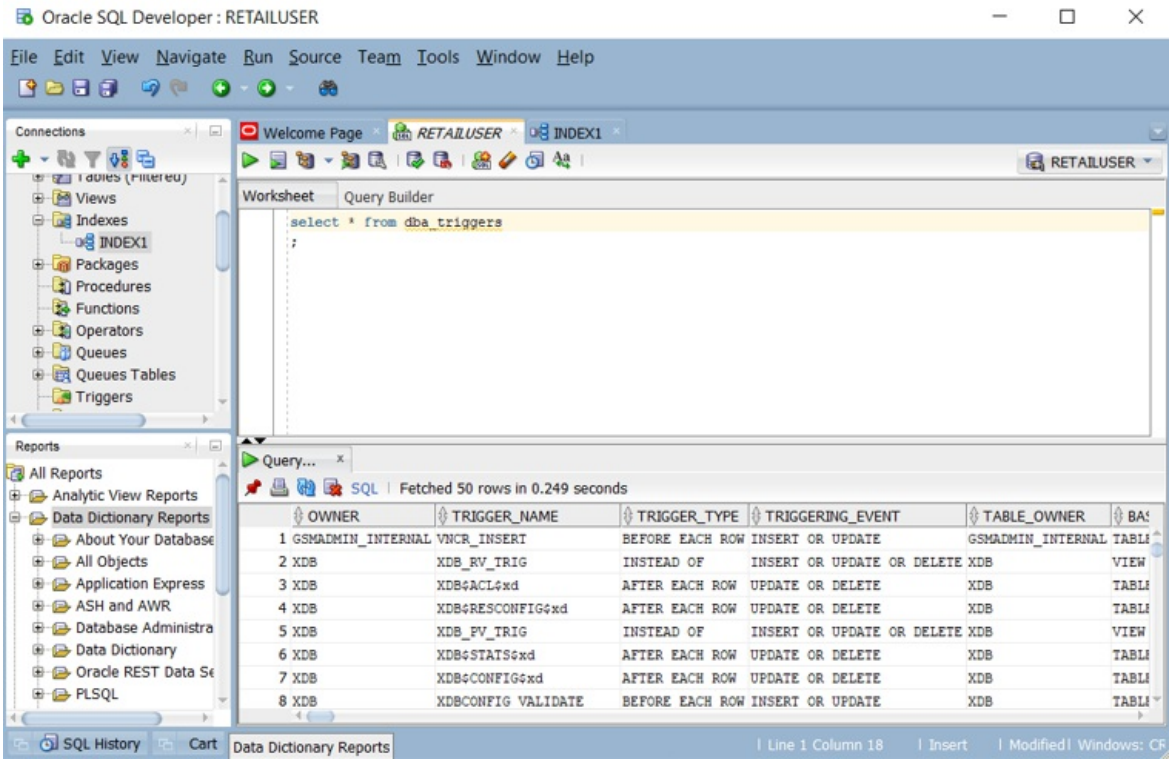
- **Materialized views**: Oracle supports materialized views and recommends using one or more for large tables with many columns where only a few columns are regularly used in queries. Materialized views are automatically refreshed by the system when data in the base table is updated.

  In 2019, Microsoft announced that Azure Synapse will support materialized views with the same functionality as in Oracle. Materialized views are now a preview feature in Azure Synapse.

- **In-database triggers**: in Oracle, a trigger can be configured to automatically run when a triggering event occurs. Triggering events can be:

  - A data manipulation language (DML) statement, such as `INSERT`, `UPDATE`, or `DELETE`, runs on a table. If you defined a trigger that fires before an `INSERT` statement on a customer table, the trigger will fire once before a new row is inserted into the customer table.
  - A DDL statement, such as `CREATE` or `ALTER`, runs. This trigger is often used for auditing purposes to record schema changes.
  - A system event, such as startup or shutdown of the Oracle database.
  - A user event, such as sign in or sign out.

You can get a list of the triggers defined in an Oracle database by querying the `ALL_TRIGGERS`, `DBA_TRIGGERS`, or `USER_TRIGGERS` views. The following screenshot shows a `DBA_TRIGGERS` query in Oracle SQL Developer.



Azure Synapse doesn't support Oracle database triggers. However, you can add equivalent functionality by using Data Factory, although doing so will require you to refactor the processes that use triggers.

- **Synonyms**: Oracle supports defining synonyms as alternative names for several database object types. Those object types include: tables, views, sequences, procedures, stored functions, packages, materialized views, Java class schema objects, user-defined objects, or another synonym.

  Azure Synapse doesn't currently support defining synonyms, although if a synonym in Oracle refers to a table or view, then you can define a view in Azure Synapse to match the alternative name. If a synonym in Oracle refers to a function or stored procedure, then in Azure Synapse you can create another function or stored procedure, with a name to match the synonym, that calls the target.

- **User-defined types**: Oracle supports user-defined objects that can contain a series of individual fields, each with their own definition and default values. Those objects can be referenced within a table definition in the same way as built-in data types like `NUMBER` or `VARCHAR`. You can get a list of user-defined types within an Oracle database by querying the `ALL_TYPES`, `DBA_TYPES`, or `USER_TYPES` views.

  Azure Synapse doesn't currently support user-defined types. If the data you need to migrate includes user-defined data types, either "flatten" them into a conventional table definition, or if they're arrays of data, normalize them in a separate table.

Oracle data type mapping

Most Oracle data types have a direct equivalent in Azure Synapse. The following table shows the recommended approach for mapping Oracle data types to Azure Synapse.

ORACLE DATA TYPE	AZURE SYNAPSE DATA TYPE
BFILE	Not supported. Map to VARBINARY (MAX).
BINARY_FLOAT	Not supported. Map to FLOAT.

ORACLE DATA TYPE	AZURE SYNAPSE DATA TYPE
BINARY_DOUBLE	Not supported. Map to DOUBLE.
BLOB	Not directly supported. Replace with VARBINARY(MAX).
CHAR	CHAR
CLOB	Not directly supported. Replace with VARCHAR(MAX).
DATE	DATE in Oracle can also contain time information. Depending on usage map to DATE or TIMESTAMP.
DECIMAL	DECIMAL
DOUBLE	PRECISION DOUBLE
FLOAT	FLOAT
INTEGER	INT
INTERVAL YEAR TO MONTH	INTERVAL data types aren't supported. Use date comparison functions, such as DATEDIFF or DATEADD, for date calculations.
INTERVAL DAY TO SECOND	INTERVAL data types aren't supported. Use date comparison functions, such as DATEDIFF or DATEADD, for date calculations.
LONG	Not supported. Map to VARCHAR(MAX).
LONG RAW	Not supported. Map to VARBINARY(MAX).
NCHAR	NCHAR
NVARCHAR2	NVARCHAR
NUMBER	NUMBER
NCLOB	Not directly supported. Replace with NVARCHAR(MAX).
NUMERIC	NUMERIC
ORD media data types	Not supported
RAW	Not supported. Map to VARBINARY.
REAL	REAL
ROWID	Not supported. Map to GUID, which is similar.
SDO Geospatial data types	Not supported

ORACLE DATA TYPE	AZURE SYNAPSE DATA TYPE
SMALLINT	SMALLINT
TIMESTAMP	DATETIME2 or the CURRENT_TIMESTAMP() function
TIMESTAMP WITH LOCAL TIME ZONE	Not supported. Map to DATETIMEOFFSET.
TIMESTAMP WITH TIME ZONE	Not supported because TIME is stored using wall-clock time without a time zone offset.
URIType	Not supported. Store in a VARCHAR.
UROWID	Not supported. Map to GUID, which is similar.
VARCHAR	VARCHAR
VARCHAR2	VARCHAR
XMLType	Not supported. Store XML data in a VARCHAR.

Oracle also supports defining user-defined objects that can contain a series of individual fields, each with their own definition and default values. Those objects can then be referenced within a table definition in the same way as built-in data types like `NUMBER` or `VARCHAR`. Azure Synapse doesn't currently support user-defined types. If the data you need to migrate includes user-defined data types, either "flatten" them into a conventional table definition, or if they're arrays of data, normalize them in a separate table.

> **TIP**
> Assess the number and type of unsupported data types during the migration preparation phase.

Third-party vendors offer tools and services to automate migration, including the mapping of data types. If a third-party ETL tool is already in use in the Oracle environment, use that tool to implement any required data transformations.

SQL DML syntax differences

SQL DML syntax differences exist between Oracle SQL and Azure Synapse T-SQL. Those differences are discussed in detail in Minimize SQL issues for Oracle migrations. In some cases, you can automate DML migration by using Microsoft tools like SSMA for Oracle and Azure Database Migration Services, or third-party migration products and services.

Functions, stored procedures, and sequences

When migrating a data warehouse from a mature environment like Oracle, you probably need to migrate elements other than simple tables and views. Check whether tools within the Azure environment can replace the functionality of functions, stored procedures, and sequences because it's usually more efficient to use built-in Azure tools than to recode them for Azure Synapse.

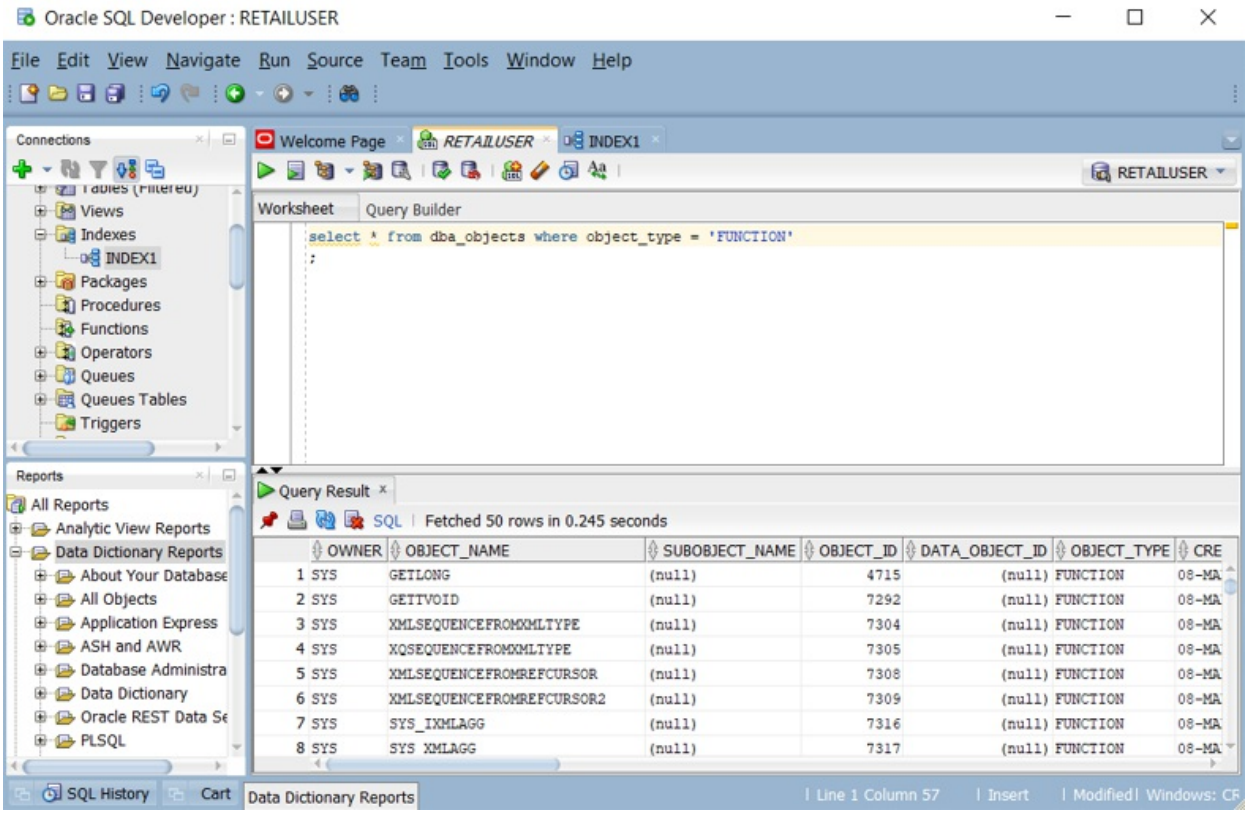
As part of your preparation phase, create an inventory of objects that need to be migrated, define a method for handling them, and allocate appropriate resources in your migration plan.

Microsoft tools like SSMA for Oracle and Azure Database Migration Services, or third-party migration products and services, can automate the migration of functions, stored procedures, and sequences.

The following sections further discuss the migration of functions, stored procedures, and sequences.

Functions

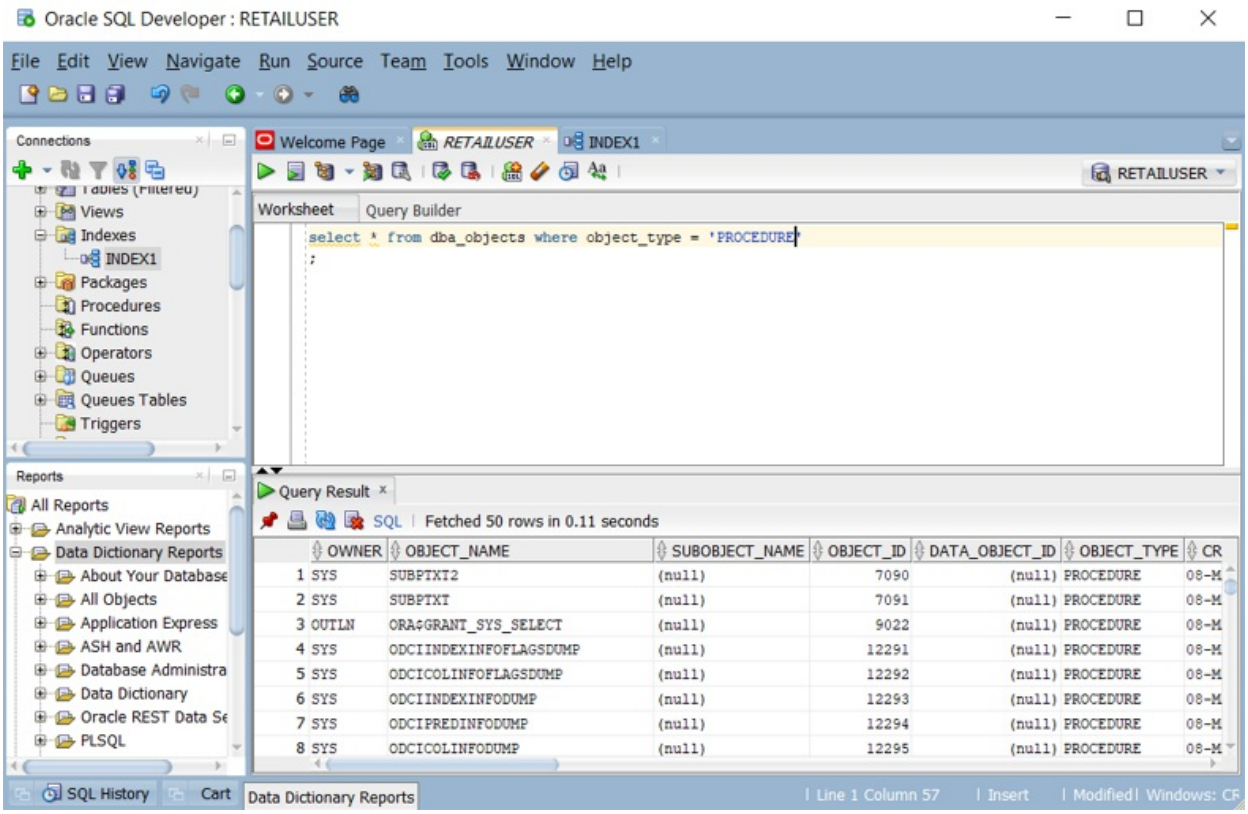
As with most database products, Oracle supports system and user-defined functions within a SQL implementation. When you migrate a legacy database platform to Azure Synapse, common system functions can usually be migrated without change. Some system functions might have a slightly different syntax, but any required changes can be automated. You can get a list of functions within an Oracle database by querying the `ALL_OBJECTS` view with the appropriate `WHERE` clause. You can use Oracle SQL Developer to get a list of functions, as shown in the following screenshot.



For Oracle system functions or arbitrary user-defined functions that have no equivalent in Azure Synapse, recode those functions using a target environment language. Oracle user-defined functions are coded in PL/SQL, Java, or C. Azure Synapse uses the Transact-SQL language to implement user-defined functions.

Stored procedures

Most modern database products support storing procedures within the database. Oracle provides the PL/SQL language for this purpose. A stored procedure typically contains both SQL statements and procedural logic, and returns data or a status. You can get a list of stored procedures within an Oracle database by querying the `ALL_OBJECTS` view with the appropriate `WHERE` clause. You can use Oracle SQL Developer to get a list of stored procedures, as shown in the next screenshot.



Azure Synapse supports stored procedures using T-SQL, so you'll need to recode any migrated stored procedures in that language.

Sequences

In Oracle, a sequence is a named database object, created using `CREATE SEQUENCE`. A sequence provides unique numeric values via the `CURRVAL` and `NEXTVAL` methods. You can use the generated unique numbers as surrogate key values for primary keys.

Azure Synapse doesn't implement `CREATE SEQUENCE`, but you can implement sequences using IDENTITY columns or SQL code that generates the next sequence number in a series.

Extracting metadata and data from an Oracle environment

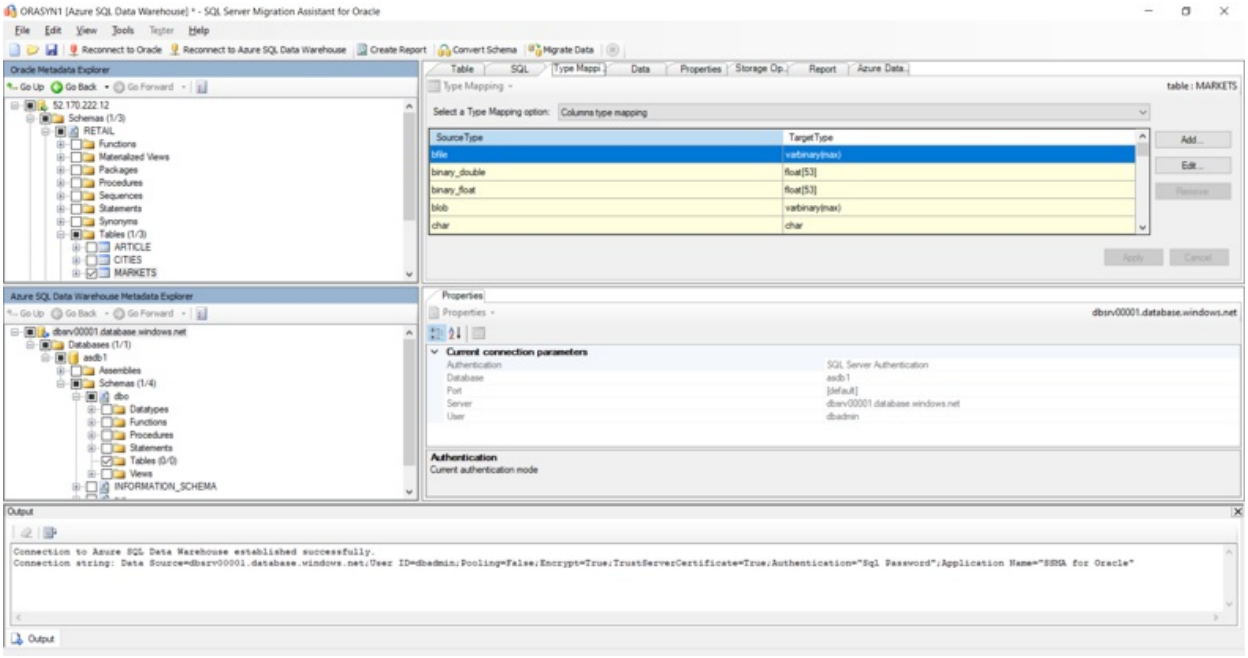
Data Definition Language generation

The ANSI SQL standard defines the basic syntax for Data Definition Language (DDL) commands. Some DDL commands, such as `CREATE TABLE` and `CREATE VIEW`, are common to both Oracle and Azure Synapse but also provide implementation-specific features such as indexing, table distribution, and partitioning options.

You can edit existing Oracle `CREATE TABLE` and `CREATE VIEW` scripts to achieve equivalent definitions in Azure Synapse. To do so, you might need to use modified data types and remove or modify Oracle-specific clauses such as `TABLESPACE`.

Within the Oracle environment, system catalog tables specify the current table and view definition. Unlike user-maintained documentation, system catalog information is always complete and in sync with current table definitions. You can access system catalog information using utilities such as Oracle SQL Developer. Oracle SQL Developer can generate `CREATE TABLE` DDL statements that you can edit to create equivalent tables in Azure Synapse.

Or, you can use SSMA for Oracle to migrate tables from an existing Oracle environment to Azure Synapse. SSMA for Oracle will apply the appropriate data type mappings and recommended table and distribution types, as shown in the following screenshot.



You can also use third-party migration and ETL tools that process system catalog information to achieve similar results.

Data extraction from Oracle

You can extract raw table data from Oracle tables to flat delimited files, such as CSV files, using standard Oracle utilities like Oracle SQL Developer, SQL*Plus, and SCLcl. Then, you can compress the flat delimited files using gzip, and upload the compressed files to Azure Blob Storage using AzCopy or Azure data transport tools like Azure Data Box.

Extract table data as efficiently as possible—especially when migrating large fact tables. For Oracle tables, use parallelism to maximize extraction throughput. You can achieve parallelism by running multiple processes that individually extract discrete segments of data, or by using tools capable of automating parallel extraction through partitioning.

> **TIP**
>
> Use parallelism for the most efficient data extraction.

If sufficient network bandwidth is available, you can extract data from an on-premises Oracle system directly into Azure Synapse tables or Azure Blob Data Storage. To do so, use Data Factory processes, Azure Database Migration Service, or third-party data migration or ETL products.

Extracted data files should contain delimited text in CSV, Optimized Row Columnar (ORC), or Parquet format.

For more information on migrating data and ETL from an Oracle environment, see Data migration, ETL, and load for Oracle migrations.

Performance recommendations for Oracle migrations

The goal of performance optimization is same or better data warehouse performance after migration to Azure Synapse.

Similarities in performance tuning approach concepts

Many performance tuning concepts for Oracle databases hold true for Azure Synapse databases. For example:

- Use data distribution to collocate data-to-be-joined onto the same processing node.
- Use the smallest data type for a given column to save storage space and accelerate query processing.

- Ensure that columns to be joined have the same data type in order to optimize join processing and reduce the need for data transforms.
- To help the optimizer produce the best execution plan, ensure statistics are up to date.
- Monitor performance using built-in database capabilities to ensure that resources are being used efficiently.

> **TIP**
> Prioritize familiarity with Azure Synapse tuning options at the start of a migration.

Differences in performance tuning approach

This section highlights low-level performance tuning implementation differences between Oracle and Azure Synapse.

Data distribution options

For performance, Azure Synapse was designed with multi-node architecture and uses parallel processing. To optimize table performance in Azure Synapse, you can define a data distribution option in `CREATE TABLE` statements using the `DISTRIBUTION` statement. For example, you can specify a hash-distributed table, which distributes table rows across compute nodes by using a deterministic hash function. Many Oracle implementations, especially older on-premises systems, don't support this feature.

Unlike Oracle, Azure Synapse supports local joins between a small table and a large table through small table replication. For instance, consider a small dimension table and a large fact table within a star schema model. Azure Synapse can replicate the smaller dimension table across all nodes to ensure that the value of any join key for the large table has a matching, locally available dimension row. The overhead of dimension table replication is relatively low for a small dimension table. For large dimension tables, a hash distribution approach is more appropriate. For more information on data distribution options, see Design guidance for using replicated tables and Guidance for designing distributed tables.

> **TIP**
> Hash distribution improves query performance on large fact tables. Round-robin distribution is useful for improving loading speed.

Hash distribution can be applied on multiple columns for a more even distribution of the base table. Multi-column distribution will allow you to choose up to eight columns for distribution. This not only reduces the data skew over time but also improves query performance.

> **NOTE**
> Multi-column distribution is currently in preview for Azure Synapse Analytics. You can use multi-column distribution with CREATE MATERIALIZED VIEW, CREATE TABLE, and CREATE TABLE AS SELECT.

Distribution Advisor

In Azure Synapse SQL, the way each table is distributed can be customized. The table distribution strategy affects query performance substantially.

The distribution advisor is a new feature in Synapse SQL that analyzes queries and recommends the best distribution strategies for tables to improve query performance. Queries to be considered by the advisor can be provided by you or pulled from your historic queries available in DMV.

For details and examples on how to use the distribution advisor, visit Distribution Advisor in Azure Synapse SQL.

Data indexing

Azure Synapse supports several user-definable indexing options that have a different operation and usage compared to system-managed zone maps in Oracle. For more information about the different indexing options in Azure Synapse, see Indexes on dedicated SQL pool tables.

Index definitions within a source Oracle environment provide a useful indication of data usage and the candidate columns for indexing in the Azure Synapse environment. Typically, you won't need to migrate every index from a legacy Oracle environment because Azure Synapse doesn't over-rely on indexes and implements the following features to achieve outstanding performance:

- Parallel query processing.
- In-memory data and result set caching.
- Data distribution, such as replication of small dimension tables, to reduce I/O.

Data partitioning

In an enterprise data warehouse, fact tables can contain billions of rows. Partitioning optimizes the maintenance and querying of these tables by splitting them into separate parts to reduce the amount of data processed. In Azure Synapse, the `CREATE TABLE` statement defines the partitioning specification for a table.

You can only use one field per table for partitioning. That field is frequently a date field because many queries are filtered by date or a date range. It's possible to change the partitioning of a table after initial load by using the `CREATE TABLE AS` (CTAS) statement to recreate the table with a new distribution. For a detailed discussion of partitioning in Azure Synapse, see Partitioning tables in dedicated SQL pool.

PolyBase or COPY INTO for data loading

PolyBase supports efficient loading of large amounts of data to a data warehouse by using parallel loading streams. For more information, see PolyBase data loading strategy.

COPY INTO also supports high-throughput data ingestion, and:

- Data retrieval from all files within a folder and subfolders.
- Data retrieval from multiple locations in the same storage account. You can specify multiple locations by using comma separated paths.
- Azure Data Lake Storage (ADLS) and Azure Blob Storage.
- CSV, PARQUET, and ORC file formats.

> **TIP**
> The recommended method for data loading is to use `COPY INTO` along with PARQUET file format.

Workload management

Running mixed workloads can pose resource challenges on busy systems. A successful workload management scheme effectively manages resources, ensures highly efficient resource utilization, and maximizes return on investment (ROI). Workload classification, workload importance, and workload isolation give more control over how workload utilizes system resources.

The workload management guide describes the techniques to analyze the workload, manage and monitor workload importance, and the steps to convert a resource class to a workload group. Use the Azure portal and T-SQL queries on DMVs to monitor the workload to ensure that the applicable resources are efficiently utilized.

Next steps

To learn about ETL and load for Oracle migration, see the next article in this series: Data migration, ETL, and load for Oracle migrations.

Data migration, ETL, and load for Oracle migrations

8/19/2022 • 19 minutes to read • Edit Online

This article is part two of a seven-part series that provides guidance on how to migrate from Oracle to Azure Synapse Analytics. The focus of this article is best practices for ETL and load migration.

Data migration considerations

There are many factors to consider when migrating data, ETL, and loads from a legacy Oracle data warehouse and data marts to Azure Synapse.

Initial decisions about data migration from Oracle

When you're planning a migration from an existing Oracle environment, consider the following data-related questions:

- Should unused table structures be migrated?
- What's the best migration approach to minimize risk and impact for users?
- When migrating data marts: stay physical or go virtual?

The next sections discuss these points within the context of a migration from Oracle.

Migrate unused tables?

It makes sense to only migrate tables that are in use. Tables that aren't active can be archived rather than migrated, so that the data is available if needed in the future. It's best to use system metadata and log files rather than documentation to determine which tables are in use, because documentation can be out of date.

Oracle system catalog tables and logs contain information that can be used to determine when a given table was last accessed—which in turn can be used to decide whether or not a table is a candidate for migration.

If you've licensed the Oracle Diagnostic Pack, then you have access to Active Session History, which you can use to determine when a table was last accessed.

> **TIP**
> In legacy systems, it's not unusual for tables to become redundant over time—these don't need to be migrated in most cases.

Here's an example query that looks for the usage of a specific table within a given time window:

```
SELECT du.username,
    s.sql_text,
    MAX(ash.sample_time) AS last_access ,
    sp.object_owner ,
    sp.object_name ,
    sp.object_alias as aliased_as ,
    sp.object_type ,
    COUNT(*) AS access_count
FROM v$active_session_history ash
    JOIN v$sql s ON ash.force_matching_signature = s.force_matching_signature
    LEFT JOIN v$sql_plan sp ON s.sql_id = sp.sql_id
    JOIN DBA_USERS du ON ash.user_id = du.USER_ID
WHERE ash.session_type = 'FOREGROUND'
    AND ash.SQL_ID IS NOT NULL
    AND sp.object_name IS NOT NULL
    AND ash.user_id <> 0
GROUP BY du.username,
    s.sql_text,
    sp.object_owner,
    sp.object_name,
    sp.object_alias,
    sp.object_type
ORDER BY 3 DESC;
```

This query may take a while to run if you have been running numerous queries.

What's the best migration approach to minimize risk and impact on users?

This question comes up frequently because companies may want to lower the impact of changes on the data warehouse data model to improve agility. Companies often see an opportunity to further modernize or transform their data during an ETL migration. This approach carries a higher risk because it changes multiple factors simultaneously, making it difficult to compare the outcomes of the old system versus the new. Making data model changes here could also affect upstream or downstream ETL jobs to other systems. Because of that risk, it's better to redesign on this scale after the data warehouse migration.

Even if a data model is intentionally changed as part of the overall migration, it's good practice to migrate the existing model as-is to Azure Synapse, rather than do any re-engineering on the new platform. This approach minimizes the effect on existing production systems, while benefiting from the performance and elastic scalability of the Azure platform for one-off re-engineering tasks.

> **TIP**
>
> Migrate the existing model as-is initially, even if a change to the data model is planned in the future.

Data mart migration: stay physical or go virtual?

In legacy Oracle data warehouse environments, it's common practice to create many data marts that are structured to provide good performance for ad hoc self-service queries and reports for a given department or business function within an organization. A data mart typically consists of a subset of the data warehouse that contains aggregated versions of the data in a form that enables users to easily query that data with fast response times. Users can use user-friendly query tools like Microsoft Power BI, which supports business user interactions with data marts. The form of the data in a data mart is generally a dimensional data model. One use of data marts is to expose the data in a usable form even if the underlying warehouse data model is something different, such as a data vault.

You can use separate data marts for individual business units within an organization to implement robust data security regimes. Restrict access to specific data marts that are relevant to users, and eliminate, obfuscate, or anonymize sensitive data.

If these data marts are implemented as physical tables, they'll require extra storage resources and processing to

build and refresh them regularly. Also, the data in the mart will only be as up to date as the last refresh operation, and so may be unsuitable for highly volatile data dashboards.

> **TIP**
> Virtualizing data marts can save on storage and processing resources.

With the advent of lower-cost scalable MPP architectures, such as Azure Synapse, and their inherent performance characteristics, you can provide data mart functionality without instantiating the mart as a set of physical tables. One method is to effectively virtualize the data marts via SQL views onto the main data warehouse. Another way is to virtualize the data marts via a virtualization layer using features like views in Azure or third-party virtualization products. This approach simplifies or eliminates the need for extra storage and aggregation processing and reduces the overall number of database objects to be migrated.

There's another potential benefit of this approach. By implementing the aggregation and join logic within a virtualization layer, and presenting external reporting tools via a virtualized view, the processing required to create these views is pushed down into the data warehouse. The data warehouse is generally the best place to run joins, aggregations, and other related operations on large data volumes.

The primary drivers for implementing a virtual data mart over a physical data mart are:

- More agility: a virtual data mart is easier to change than physical tables and the associated ETL processes.
- Lower total cost of ownership: a virtualized implementation requires fewer data stores and copies of data.
- Elimination of ETL jobs to migrate and simplify data warehouse architecture in a virtualized environment.
- Performance: although physical data marts have historically performed better, virtualization products now implement intelligent caching techniques to mitigate this difference.

> **TIP**
> The performance and scalability of Azure Synapse enables virtualization without sacrificing performance.

Data migration from Oracle

Understand your data

As part of migration planning, you should understand in detail the volume of data to be migrated since that can affect decisions about the migration approach. Use system metadata to determine the physical space taken up by the raw data within the tables to be migrated. In this context, raw data means the amount of space used by the data rows within a table, excluding overhead such as indexes and compression. The largest fact tables will typically comprise more than 95% of the data.

This query will give you the total database size in Oracle:

```
SELECT
  ( SELECT SUM(bytes)/1024/1024/1024 data_size
    FROM sys.dba_data_files ) +
  ( SELECT NVL(sum(bytes),0)/1024/1024/1024 temp_size
    FROM sys.dba_temp_files ) +
  ( SELECT SUM(bytes)/1024/1024/1024 redo_size
    FROM sys.v_$log ) +
  ( SELECT SUM(BLOCK_SIZE*FILE_SIZE_BLKS)/1024/1024/1024 controlfile_size
    FROM v$controlfile ) "Size in GB"
FROM dual
```

The database size equals the size of `(data files + temp files + online/offline redo log files + control files)`.

Overall database size includes used space and free space.

The following example query gives a breakdown of the disk space used by table data and indexes:

```
SELECT
   owner, "Type", table_name "Name", TRUNC(sum(bytes)/1024/1024) Meg
FROM
  ( SELECT segment_name table_name, owner, bytes, 'Table' as "Type"
    FROM dba_segments
    WHERE segment_type in  ('TABLE','TABLE PARTITION','TABLE SUBPARTITION' )
UNION ALL
    SELECT i.table_name, i.owner, s.bytes, 'Index' as "Type"
    FROM dba_indexes i, dba_segments s
    WHERE s.segment_name = i.index_name
    AND   s.owner = i.owner
    AND   s.segment_type in ('INDEX','INDEX PARTITION','INDEX SUBPARTITION')
UNION ALL
    SELECT l.table_name, l.owner, s.bytes, 'LOB' as "Type"
    FROM dba_lobs l, dba_segments s
    WHERE s.segment_name = l.segment_name
    AND   s.owner = l.owner
    AND   s.segment_type IN ('LOBSEGMENT','LOB PARTITION','LOB SUBPARTITION')
UNION ALL
    SELECT l.table_name, l.owner, s.bytes, 'LOB Index' as "Type"
    FROM dba_lobs l, dba_segments s
    WHERE s.segment_name = l.index_name
    AND   s.owner = l.owner
    AND   s.segment_type = 'LOBINDEX')
    WHERE owner in UPPER('&owner')
GROUP BY table_name, owner, "Type"
HAVING SUM(bytes)/1024/1024 > 10  /* Ignore really small tables */
ORDER BY SUM(bytes) desc;
```

In addition, the Microsoft database migration team provides many resources, including the Oracle Inventory Script Artifacts. The Oracle Inventory Script Artifacts tool includes a PL/SQL query that accesses Oracle system tables and provides a count of objects by schema type, object type, and status. The tool also provides a rough estimate of raw data in each schema and the sizing of tables in each schema, with results stored in a CSV format. An included calculator spreadsheet takes the CSV as input and provides sizing data.

For any table, you can accurately estimate the volume of data that needs to be migrated by extracting a representative sample of the data, such as one million rows, to an uncompressed delimited flat ASCII data file. Then, use the size of that file to get an average raw data size per row. Finally, multiply that average size by the total number of rows in the full table to give a raw data size for the table. Use that raw data size in your planning.

Use SQL queries to find data types

By querying the Oracle static data dictionary `DBA_TAB_COLUMNS` view, you can determine which data types are in use in a schema and whether any of those data types need to be changed. Use SQL queries to find the columns in any Oracle schema with data types that don't map directly to data types in Azure Synapse. Similarly, you can use queries to count the number of occurrences of each Oracle data type that doesn't map directly to Azure Synapse. By using the results from these queries in combination with the data type comparison table, you can determine which data types need to be changed in an Azure Synapse environment.

To find the columns with data types that don't map to data types in Azure Synapse, run the following query after you replace `<owner_name>` with the relevant owner of your schema:

```
SELECT owner, table_name, column_name, data_type
FROM dba_tab_columns
WHERE owner in ('<owner_name>')
AND data_type NOT IN
    ('BINARY_DOUBLE', 'BINARY_FLOAT', 'CHAR', 'DATE', 'DECIMAL', 'FLOAT', 'LONG', 'LONG RAW', 'NCHAR',
'NUMERIC', 'NUMBER', 'NVARCHAR2', 'SMALLINT', 'RAW', 'REAL', 'VARCHAR2', 'XML_TYPE')
ORDER BY 1,2,3;
```

To count the number of non-mappable data types, use the following query:

```
SELECT data_type, count(*)
FROM dba_tab_columns
WHERE data_type NOT IN
    ('BINARY_DOUBLE', 'BINARY_FLOAT', 'CHAR', 'DATE', 'DECIMAL', 'FLOAT', 'LONG', 'LONG RAW', 'NCHAR',
'NUMERIC', 'NUMBER', 'NVARCHAR2', 'SMALLINT', 'RAW', 'REAL', 'VARCHAR2', 'XML_TYPE')
GROUP BY data_type
ORDER BY data_type;
```

Microsoft offers SQL Server Migration Assistant (SSMA) for Oracle to automate migration of data warehouses from legacy Oracle environments, including the mapping of data types. You can also use Azure Database Migration Services to help plan and perform a migration from environments like Oracle. Third-party vendors also offer tools and services to automate migration. If a third-party ETL tool is already in use in the Oracle environment, you can use that tool to implement any required data transformations. The next section explores migration of existing ETL processes.

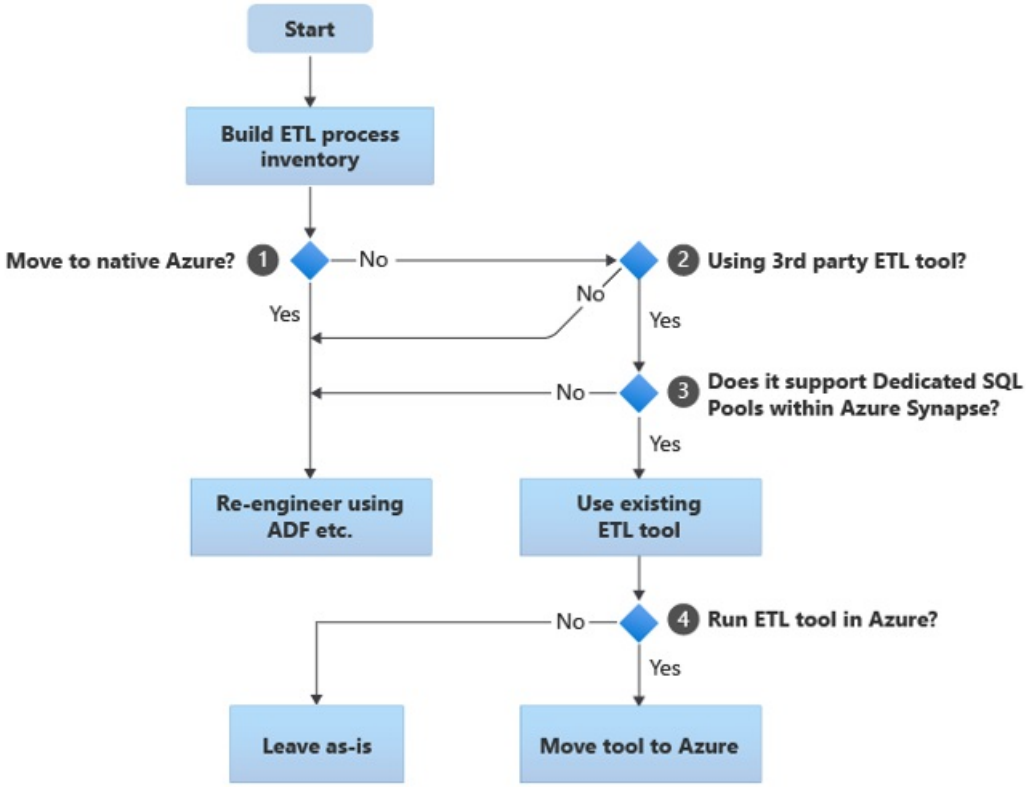
ETL migration considerations

Initial decisions about Oracle ETL migration

For ETL/ELT processing, legacy Oracle data warehouses often use custom-built scripts, third-party ETL tools, or a combination of approaches that has evolved over time. When you're planning a migration to Azure Synapse, determine the best way to implement the required ETL/ELT processing in the new environment while also minimizing cost and risk.

> **TIP**
> Plan the approach to ETL migration ahead of time and leverage Azure facilities where appropriate.

The following flowchart summarizes one approach:



As shown in the flowchart, the initial step is always to build an inventory of ETL/ELT processes that need to be migrated. With the standard built-in Azure features, some existing processes might not need to move. For planning purposes, it's important that you understand the scale of the migration. Next, consider the questions in the flowchart decision tree:

1. **Move to native Azure?** Your answer depends on whether you're migrating to a completely Azure-native environment. If so, we recommend that you re-engineer the ETL processing using Pipelines and activities in Azure Data Factory or Azure Synapse pipelines.

2. **Using a third-party ETL tool?** If you're not moving to a completely Azure-native environment, then check whether an existing third-party ETL tool is already in use. In the Oracle environment, you might find that some or all of the ETL processing is performed by custom scripts using Oracle-specific utilities such as Oracle SQL Developer, Oracle SQL*Loader, or Oracle Data Pump. The approach in this case is to re-engineer using Azure Data Factory.

3. **Does the third-party support dedicated SQL pools within Azure Synapse?** Consider whether there's a large investment in skills in the third-party ETL tool, or if existing workflows and schedules use that tool. If so, determine whether the tool can efficiently support Azure Synapse as a target environment. Ideally, the tool will include native connectors that can use Azure facilities like PolyBase or COPY INTO for the most efficient data loading. But even without native connectors, there's generally a way that you can call external processes, such as PolyBase or `COPY INTO`, and pass in applicable parameters. In this case, use existing skills and workflows, with Azure Synapse as the new target environment.

   If you're using Oracle Data Integrator (ODI) for ELT processing, then you need ODI Knowledge Modules for Azure Synapse. If those modules aren't available to you in your organization, but you have ODI, then you can use ODI to generate flat files. Those flat files can then be moved to Azure and ingested into Azure Data Lake Storage for loading into Azure Synapse.

4. **Run ETL tools in Azure?** If you decide to retain an existing third-party ETL tool, you can run that tool within the Azure environment (rather than on an existing on-premises ETL server) and have Data Factory handle the overall orchestration of the existing workflows. So, decide whether to leave the existing tool running as-is or move it into the Azure environment to achieve cost, performance, and scalability benefits.

> **TIP**
> Consider running ETL tools in Azure to leverage performance, scalability, and cost benefits.

Re-engineer existing Oracle-specific scripts

If some or all of the existing Oracle warehouse ETL/ELT processing is handled by custom scripts that use Oracle-specific utilities, such as Oracle SQL*Plus, Oracle SQL Developer, Oracle SQL*Loader, or Oracle Data Pump, then you need to recode these scripts for the Azure Synapse environment. Similarly, if ETL processes have been implemented using stored procedures in Oracle, then you need to recode those processes.

Some elements of the ETL process are easy to migrate, for example, by simple bulk data load into a staging table from an external file. It may even be possible to automate those parts of the process, for example, by using Azure Synapse `COPY INTO` or PolyBase instead of SQL*Loader. Other parts of the process that contain arbitrary complex SQL and/or stored procedures will take more time to re-engineer.

> **TIP**
> The inventory of ETL tasks to be migrated should include scripts and stored procedures.

One way of testing Oracle SQL for compatibility with Azure Synapse is to capture some representative SQL statements from a join of Oracle `v$active_session_history` and `v$sql` to get `sql_text`, then prefix those queries with `EXPLAIN`. Assuming a like-for-like migrated data model in Azure Synapse, run those `EXPLAIN` statements in Azure Synapse. Any incompatible SQL will give an error. You can use this information to determine the scale of the recoding task.

> **TIP**
> Use `EXPLAIN` to find SQL incompatibilities.

In the worst case, manual recoding may be necessary. However, there are products and services available from Microsoft partners to assist with re-engineering Oracle-specific code.

> **TIP**
> Partners offer products and skills to assist in re-engineering Oracle-specific code.

Use existing third-party ETL tools

In many cases, the existing legacy data warehouse system will already be populated and maintained by a third-party ETL product. See Azure Synapse Analytics data integration partners for a list of current Microsoft data integration partners for Azure Synapse.

The Oracle community frequently uses several popular ETL products. The following paragraphs discuss the most popular ETL tools for Oracle warehouses. You can run all of those products within a VM in Azure and use them to read and write Azure databases and files.

> **TIP**
> Leverage investment in existing third-party tools to reduce cost and risk.

Data loading from Oracle

Choices available when loading data from Oracle

When you're preparing to migrate data from an Oracle data warehouse, decide how data will be physically moved from the existing on-premises environment into Azure Synapse in the cloud, and which tools will be used to perform the transfer and load. Consider the following questions, which are discussed in the following sections.

- Will you extract the data to files, or move it directly via a network connection?
- Will you orchestrate the process from the source system, or from the Azure target environment?
- Which tools will you use to automate and manage the migration process?

Transfer data via files or network connection?

Once the database tables to be migrated have been created in Azure Synapse, you can move the data that populates those tables out of the legacy Oracle system and into the new environment. There are two basic approaches:

- **File Extract**: extract the data from the Oracle tables to flat delimited files, normally in CSV format. You can extract table data in several ways:
  - Use standard Oracle tools such as SQL*Plus, SQL Developer, and SQLcl.
  - Use Oracle Data Integrator (ODI) to generate flat files.
  - Use Oracle connector in Data Factory to unload Oracle tables in parallel to enable data loading by partitions.
  - Use a third-party ETL tool.

  For examples of how to extract Oracle table data, see the article appendix.

  This approach requires space to land the extracted data files. The space could be local to the Oracle source database if sufficient storage is available, or remote in Azure Blob Storage. The best performance is achieved when a file is written locally since that avoids network overhead.

  To minimize storage and network transfer requirements, compress the extracted data files using a utility like gzip.

  After extraction, move the flat files into Azure Blob Storage. Microsoft provides various options to move large volumes of data, including:

  - AzCopy for moving files across the network into Azure Storage.
  - Azure ExpressRoute for moving bulk data over a private network connection.
  - Azure Data Box for moving files to a physical storage device that you ship to an Azure data center for loading.

  For more information, see Transfer data to and from Azure.

- **Direct extract and load across network**: the target Azure environment sends a data extract request, normally via a SQL command, to the legacy Oracle system to extract the data. The results are sent across the network and loaded directly into Azure Synapse, with no need to land the data into intermediate files. The limiting factor in this scenario is normally the bandwidth of the network connection between the Oracle database and the Azure environment. For exceptionally large data volumes, this approach may not be practical.

> **TIP**
> Understand the data volumes to be migrated and the available network bandwidth, because these factors influence the migration approach decision.

There's also a hybrid approach that uses both methods. For example, you can use the direct network extract

approach for smaller dimension tables and samples of the larger fact tables to quickly provide a test environment in Azure Synapse. For large-volume historical fact tables, you can use the file extract and transfer approach using Azure Data Box.

Orchestrate from Oracle or Azure?

The recommended approach when moving to Azure Synapse is to orchestrate data extraction and loading from the Azure environment using SSMA or Data Factory. Use the associated utilities, such as PolyBase or `COPY INTO`, for the most efficient data loading. This approach benefits from built-in Azure capabilities and reduces the effort to build reusable data load pipelines. You can use metadata-driven data load pipelines to automate the migration process.

The recommended approach also minimizes the performance hit on the existing Oracle environment during the data load process, because the management and load process runs in Azure.

Existing data migration tools

Data transformation and movement is the basic function of all ETL products. If a data migration tool is already in use in the existing Oracle environment and it supports Azure Synapse as a target environment, then consider using that tool to simplify data migration.

Even if an existing ETL tool isn't in place, Azure Synapse Analytics data integration partners offer ETL tools to simplify the task of data migration.

Finally, if you plan to use an ETL tool, consider running that tool within the Azure environment to take advantage of Azure cloud performance, scalability, and cost. This approach also frees up resources in the Oracle data center.

Summary

To summarize, our recommendations for migrating data and associated ETL processes from Oracle to Azure Synapse are:

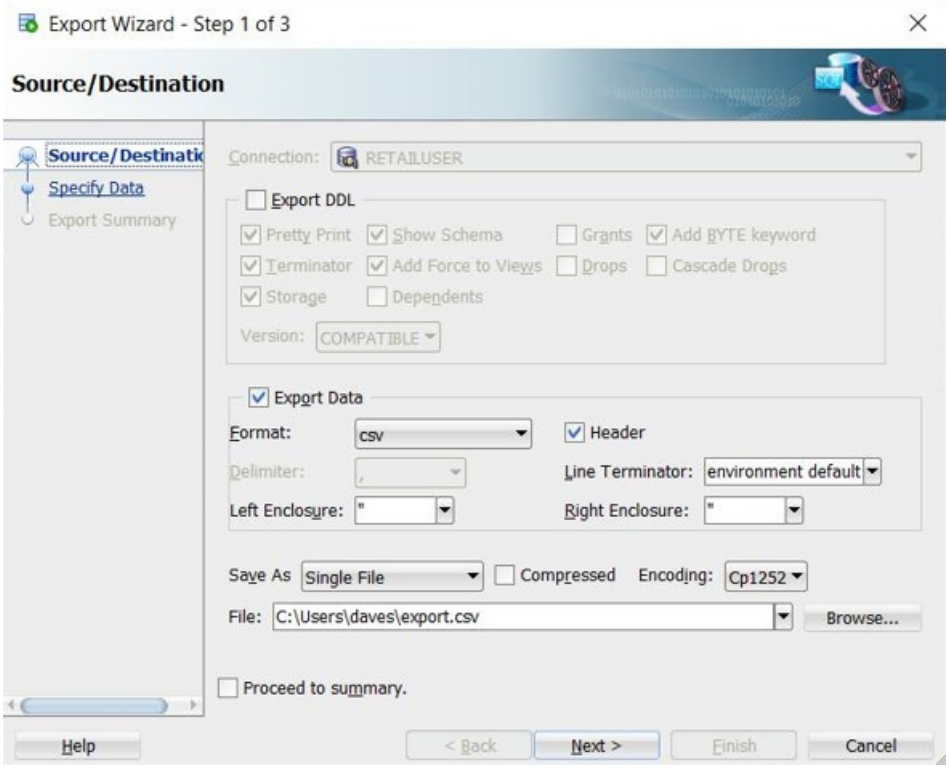
- Plan ahead to ensure a successful migration exercise.
- Build a detailed inventory of data and processes to be migrated as soon as possible.
- Use system metadata and log files to get an accurate understanding of data and process usage. Don't rely on documentation since it may be out of date.
- Understand the data volumes to be migrated, and the network bandwidth between the on-premises data center and Azure cloud environments.
- Consider using an Oracle instance in an Azure VM as a stepping stone to offload migration from the legacy Oracle environment.
- Use standard built-in Azure features to minimize the migration workload.
- Identify and understand the most efficient tools for data extraction and load in both Oracle and Azure environments. Use the appropriate tools in each phase of the process.
- Use Azure facilities, such as Data Factory, to orchestrate and automate the migration process while minimizing impact on the Oracle system.

Appendix: Examples of techniques to extract Oracle data

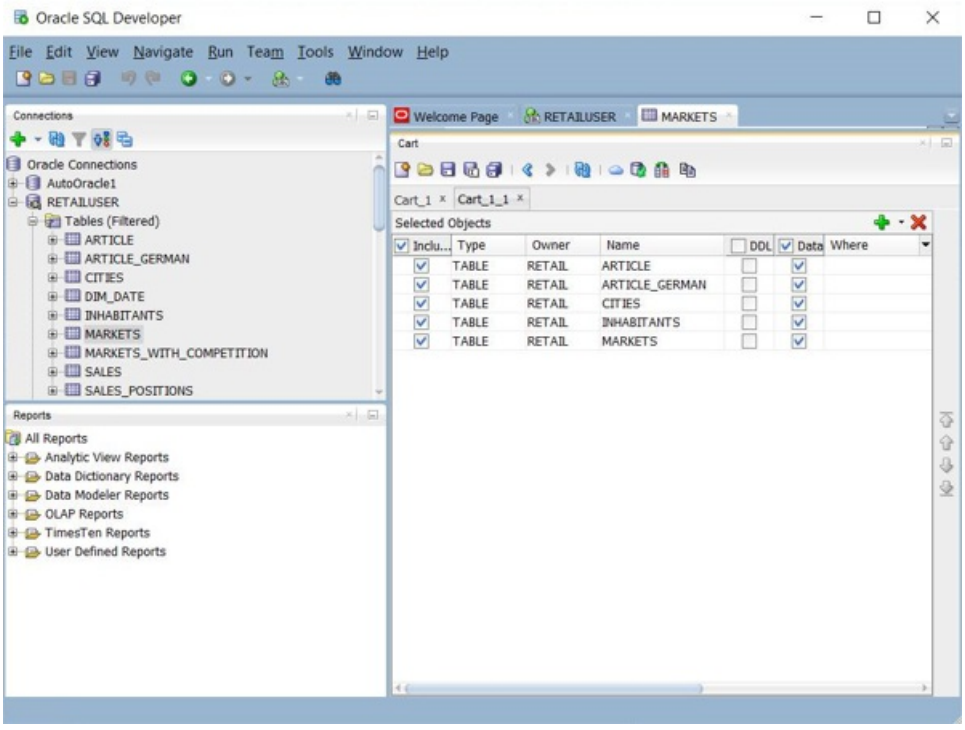
You can use several techniques to extract Oracle data when migrating from Oracle to Azure Synapse. The next sections demonstrate how to extract Oracle data using Oracle SQL Developer and the Oracle connector in Data Factory.

Use Oracle SQL Developer for data extraction

You can use the Oracle SQL Developer UI to export table data to many formats, including CSV, as shown in the following screenshot:



Other export options include JSON and XML. You can use the UI to add a set of table names to a "cart", then apply the export to the entire set in the cart:



You can also use Oracle SQL Developer Command Line (SQLcl) to export Oracle data. This option supports automation using a shell script.

For relatively small tables, you might find this technique useful if you run into problems extracting data through a direct connection.

Use the Oracle connector in Azure Data Factory for parallel copy

You can use the Oracle connector in Data Factory to unload large Oracle tables in parallel. The Oracle connector provides built-in data partitioning to copy data from Oracle in parallel. You can find the data partitioning options

in the *Source* tab of the copy activity.

For information on how to configure the Oracle connector for parallel copy, see Parallel copy from Oracle.

For more information on Data Factory copy activity performance and scalability, see Copy activity performance and scalability guide.

Next steps

To learn about security access operations, see the next article in this series: Security, access, and operations for Oracle migrations.

Security, access, and operations for Oracle migrations

8/19/2022 • 21 minutes to read • Edit Online

This article is part three of a seven-part series that provides guidance on how to migrate from Oracle to Azure Synapse Analytics. The focus of this article is best practices for security access operations.

Security considerations

The Oracle environment offers several methods for access and authentication that you might need to migrate to Azure Synapse with minimal risk and user impact. The article assumes you want to migrate the existing connection methods and the user, role, and permission structures as-is. If that's not the case, then use the Azure portal to create and manage a new security regime.

For more information about Azure Synapse security options, see Azure Synapse Analytics security.

Connection and authentication

Authentication is the process of verifying the identity of a user, device, or other entity in a computer system, generally as a prerequisite to granting access to resources in a system.

> **TIP**
> Authentication in both Oracle and Azure Synapse can be "in database" or via external methods.

Oracle authorization options

The Oracle system offers these authentication methods for database users:

- **Database authentication**: with database authentication, the Oracle database administers the user account and authenticates the user. For the Oracle database to perform authentication, it generates a password for new users and stores passwords in encrypted format. Users can change their password at any time. Oracle recommends password management through account locking, password aging and expiration, password history, and password complexity verification. Database authentication is common in older Oracle installations.
- **External authentication**: with external authentication, the Oracle database maintains the user account, and an external service performs password administration and user authentication. The external service can be an operating system or a network service like Oracle Net. The database relies on the underlying operating system or network authentication service to restrict access to database accounts. This type of sign doesn't use a database password. There are two external authentication options:
  - **Operating system authentication**: by default, Oracle requires a secure connection for logins that the operating system authenticates to prevent a remote user from impersonating an operating system user over a network connection. This requirement precludes the use of Oracle Net and a shared-server configuration.
  - **Network authentication**: several network authentication mechanisms are available, such as smart cards, fingerprints, Kerberos, and the operating system. Many network authentication services, such as Kerberos, support single sign-on so users have fewer passwords to remember.
- **Global authentication and authorization**: with global authentication and authorization, you can centralize management of user-related information, including authorizations, in an LDAP-based directory

service. Users are identified in the database as global users, which means they're authenticated by TLS/SSL and user management occurs outside the database. The centralized directory service performs user management. This approach provides strong authentication using TLS/SSL, Kerberos, or Windows-native authentication, and enables centralized management of users and privileges across the enterprise. Administration is easier because it's not necessary to create a schema for every user in every database in the enterprise. Single sign-on is also supported, so that users only need to sign in once to access multiple databases and services.

- **Proxy authentication and authorization**: you can designate a middle-tier server to proxy clients in a secure fashion. Oracle provides various options for proxy authentication, such as:
  - The middle-tier server can authenticate itself with the database server. A client, which in this case is an application user or another application, authenticates itself with the middle-tier server. Client identities can be maintained all the way through to the database.
  - The client, which in this case is database user, isn't authenticated by the middle-tier server. The client's identity and database password are passed through the middle-tier server to the database server for authentication.
  - The client, which in this case is a global user, is authenticated by the middle-tier server, and passes either a distinguished name (DN) or certificate through the middle tier for retrieving the client's username.

Azure Synapse authorization options

Azure Synapse supports two basic options for connection and authorization:

- **SQL authentication**: SQL authentication uses a database connection that includes a database identifier, user ID, and password, plus other optional parameters. This method of authentication is functionally equivalent to Oracle database authentication.
- **Azure AD authentication**: with Azure AD authentication, you can centrally manage the identities of database users and Microsoft services in one location. Centralized management provides a single place to manage Azure Synapse users and simplifies permission management. Azure AD authentication supports connections to LDAP and Kerberos services. For example, you can use Azure AD authentication to connect to existing LDAP directories if they're to remain in place after migration of the database.

Users, roles, and permissions

Both Oracle and Azure Synapse implement database access control via a combination of users, roles, and permissions. You can use standard SQL statements `CREATE USER` and `CREATE ROLE/GROUP` to define users and roles. `GRANT` and `REVOKE` statements assign or remove permissions to users and/or roles.

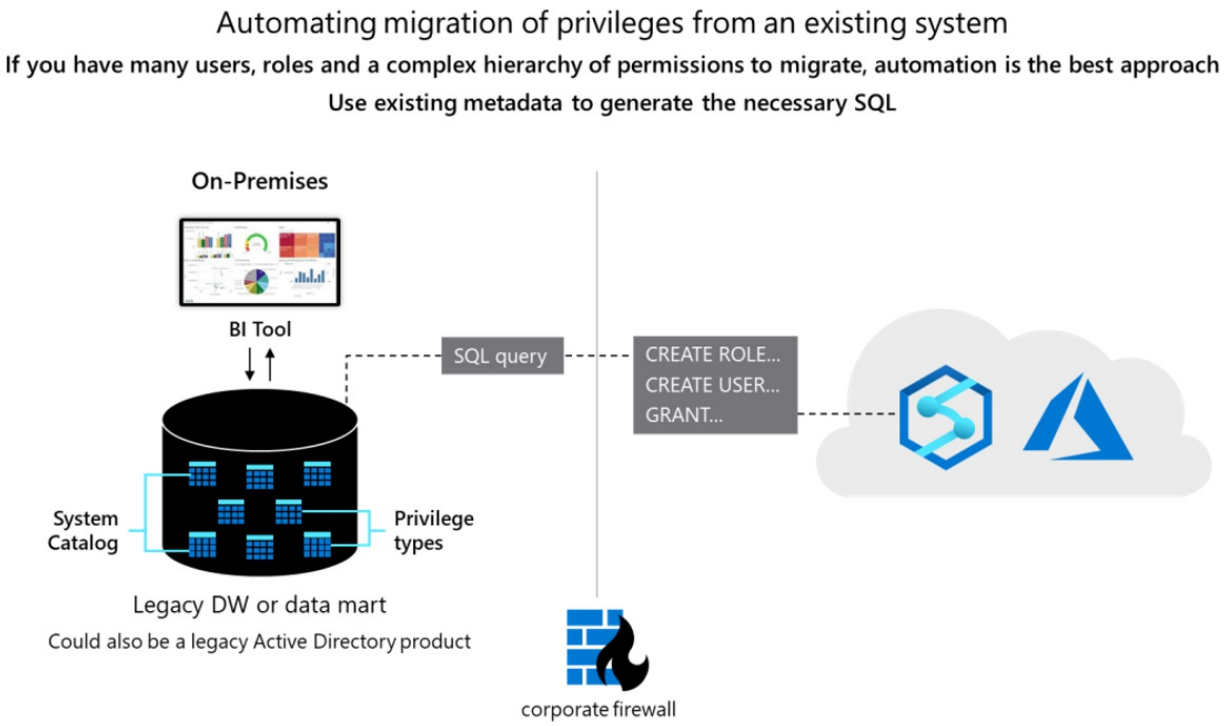
> **TIP**
> Planning is essential for a successful migration project. Start with high-level approach decisions.

Conceptually, Oracle and Azure Synapse databases are similar, and to some degree it's possible to automate the migration of existing user IDs, groups, and permissions. Extract the legacy user and group information from the Oracle system catalog tables, then generate matching equivalent `CREATE USER` and `CREATE ROLE` statements. Run those statements in Azure Synapse to recreate the same user/role hierarchy.

> **TIP**
> If possible, automate migration processes to reduce elapsed time and scope for error.

After data extraction, use Oracle system catalog tables to generate equivalent `GRANT` statements to assign

permissions if an equivalent exists.

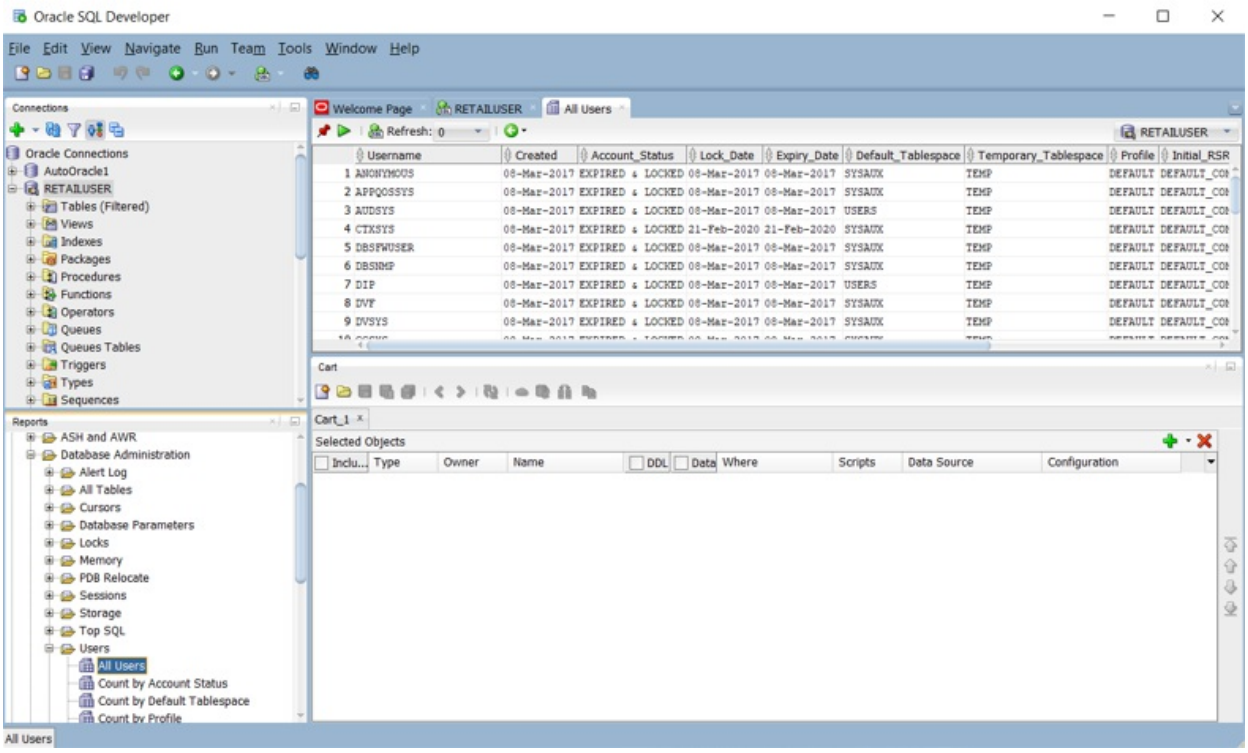


Users and roles

The information about current users and groups in an Oracle system is held in system catalog views, such as `ALL_USERS` and `DBA_USERS`. You can query these views in the normal way via Oracle SQL*Plus or Oracle SQL Developer. The following queries are basic examples:

```
--List of users
select * from dba_users order by username;

--List of roles
select * from dba_roles order by role;

--List of users and their associated roles
select * from user_role_privs order by username, granted_role;
```

Oracle SQL Developer has built-in options to display user and role information in the *Reports* pane, as shown in the following screenshot.



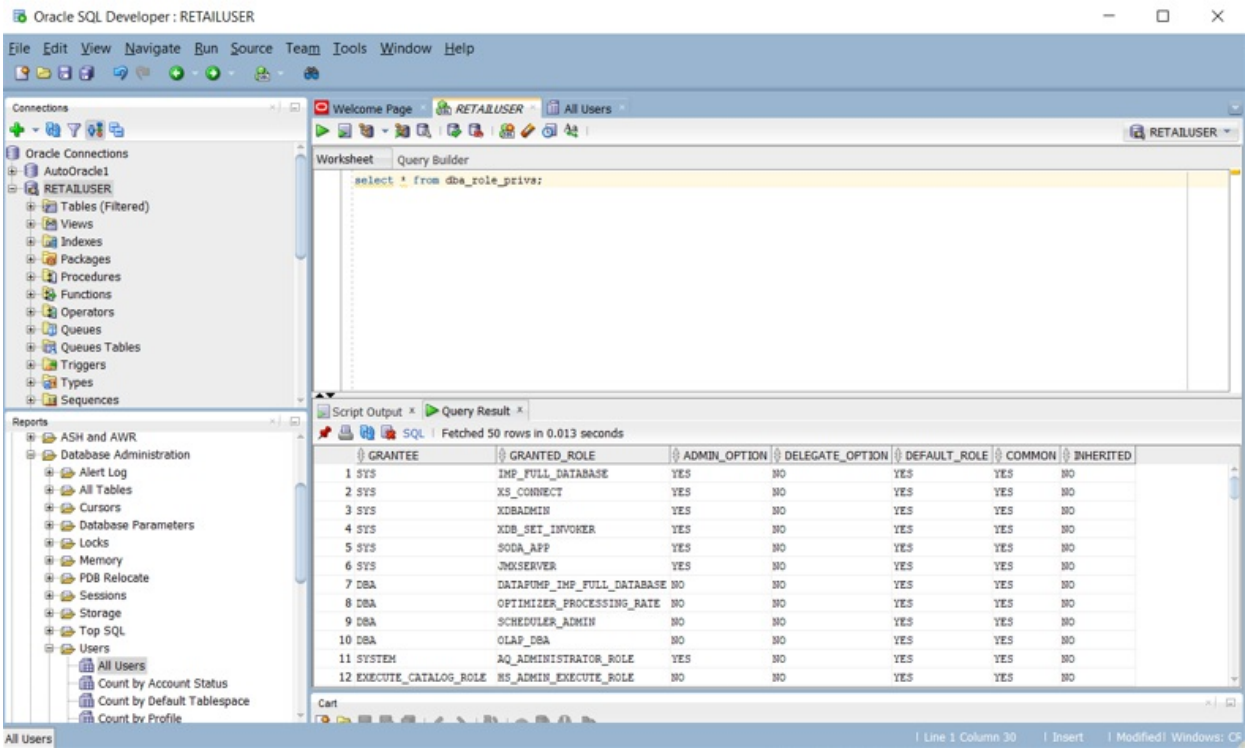
You can modify the example `SELECT` statement to produce a result set that is a series of `CREATE USER` and `CREATE GROUP` statements. To do so, include the appropriate text as a literal within the `SELECT` statement.

There's no way to retrieve existing Oracle passwords, so you need to implement a scheme for allocating new initial passwords on Azure Synapse.

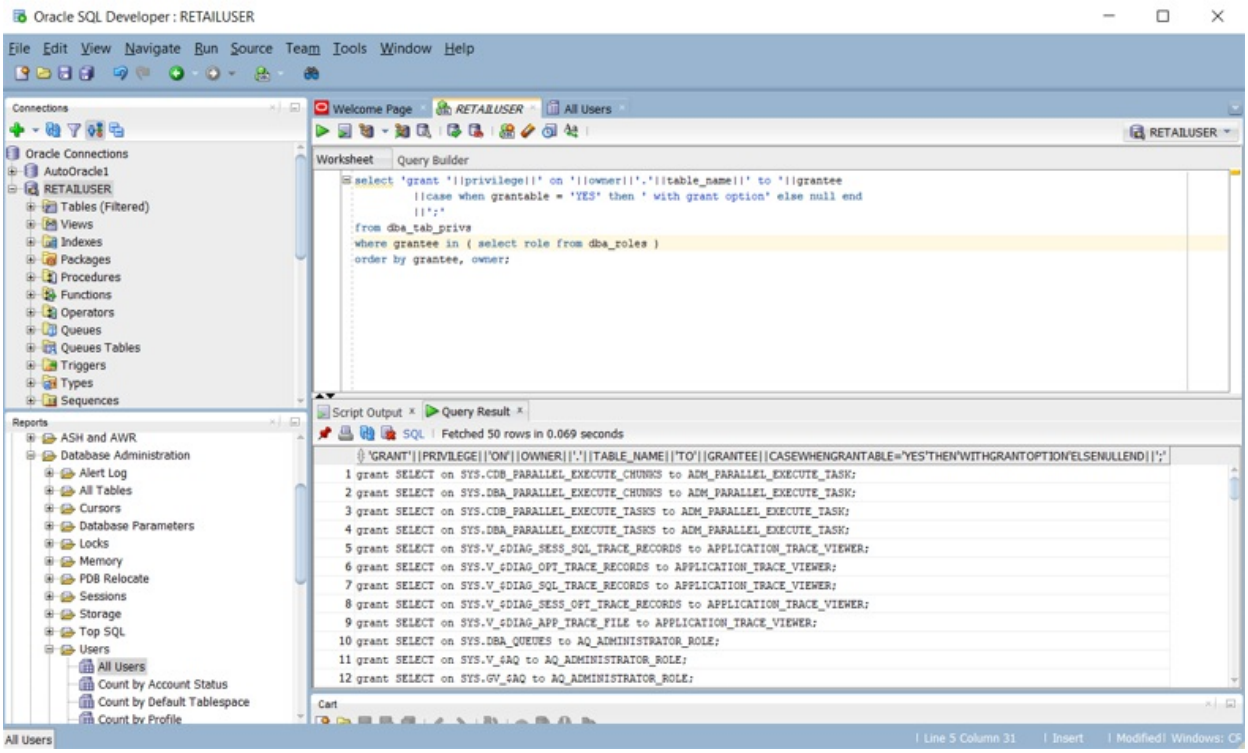
> **TIP**
> Migration of a data warehouse requires migrating more than just tables, views, and SQL statements.

Permissions

In an Oracle system, the system `DBA_ROLE_PRIVS` view holds the access rights for users and roles. If you have `SELECT` access, you can query that view to obtain the current access rights lists defined within the system. The following Oracle SQL Developer screenshot shows an example access rights list.



You can also create queries to produce a script that's a series of `CREATE` and `GRANT` statements for Azure Synapse, based on the existing Oracle privileges. The following Oracle SQL Developer screenshot shows an example of that script.



This table lists and describes the data dictionary views required to view user, role, and privileges information.

VIEW	DESCRIPTION
DBA_COL_PRIVS ALL_COL_PRIVS USER_COL_PRIVS	The DBA view describes all column object grants in the database. The ALL view describes all column object grants for which the current user or PUBLIC is the object owner, grantor, or grantee. The USER view describes column object grants for which the current user is the object owner, grantor, or grantee.

VIEW	DESCRIPTION
ALL_COL_PRIVS_MADE USER_COL_PRIVS_MADE	The ALL view lists column object grants for which the current user is the object owner or grantor. The USER view describes column object grants for which the current user is the grantor.
ALL_COL_PRIVS_RECD USER_COL_PRIVS_RECD	The ALL view describes column object grants for which the current user or PUBLIC is the grantee. The USER view describes column object grants for which the current user is the grantee.
DBA_TAB_PRIVS ALL_TAB_PRIVS USER_TAB_PRIVS	The DBA view lists all grants on all objects in the database. The ALL view lists the grants on objects where the user or PUBLIC is the grantee. The USER view lists grants on all objects where the current user is the grantee.
ALL_TAB_PRIVS_MADE USER_TAB_PRIVS_MADE	The ALL view lists object grants made by the current user or made on the objects owned by the current user. The USER view lists grants on all objects owned by the current user.
ALL_TAB_PRIVS_RECD USER_TAB_PRIVS_RECD	The ALL view lists object grants for which the user or PUBLIC is the grantee. The USER view lists object grants for which the current user is the grantee.
DBA_ROLES	This view lists all roles that exist in the database.
DBA_ROLE_PRIVS USER_ROLE_PRIVS	The DBA view lists roles granted to users and roles. The USER view lists roles granted to the current user.
DBA_SYS_PRIVS USER_SYS_PRIVS	The DBA view lists system privileges granted to users and roles. The USER view lists system privileges granted to the current user.
ROLE_ROLE_PRIVS	This view describes roles granted to other roles. Information is provided only about roles to which the user has access.
ROLE_SYS_PRIVS	This view contains information about system privileges granted to roles. Information is provided only about roles to which the user has access.
ROLE_TAB_PRIVS	This view contains information about object privileges granted to roles. Information is provided only about roles to which the user has access.
SESSION_PRIVS	This view lists the privileges that are currently enabled for the user.
SESSION_ROLES	This view lists the roles that are currently enabled for the user.

Oracle supports various types of privileges:

- **System privileges**: system privileges allow the grantee to perform standard administrator tasks in the database. Typically, these privileges are restricted to trusted users. Many system privileges are specific to Oracle operations.
- **Object privileges**: each type of object has privileges associated with it.

- **Table privileges**: table privileges enable security at the data manipulation language (DML) or data definition language (DDL) level. You can map table privileges directly to their equivalent in Azure Synapse.
- **View privileges**: you can apply DML object privileges to views, similar to tables. You can map view privileges directly to their equivalent in Azure Synapse.
- **Procedure privileges**: procedures privileges allow procedures, including standalone procedures and functions, to be granted the `EXECUTE` privilege. You can map procedure privileges directly to their equivalent in Azure Synapse.
- **Type privileges**: you can grant system privileges to named types, such as object types, `VARRAYs`, and nested tables. Typically, these privileges are specific to Oracle and have no equivalent in Azure Synapse.

> **TIP**
> Azure Synapse has equivalent permissions for basic database operations such as DML and DDL.

The following table lists common Oracle admin privileges that have a direct equivalent in Azure Synapse.

ADMIN PRIVILEGE	DESCRIPTION	SYNAPSE EQUIVALENT
[Create] Database	The user can create databases. Permission to operate on existing databases is controlled by object privileges.	CREATE DATABASE
[Create] External Table	The user can create external tables. Permission to operate on existing tables is controlled by object privileges.	CREATE TABLE
[Create] Function	The user can create user-defined functions (UDFs). Permission to operate on existing UDFs is controlled by object privileges.	CREATE FUNCTION
[Create] Role	The user can create groups. Permission to operate on existing groups is controlled by object privileges.	CREATE ROLE
[Create] Index	For system use only. Users can't create indexes.	CREATE INDEX
[Create] Materialized View	The user can create materialized views.	CREATE VIEW
[Create] Procedure	The user can create stored procedures. Permission to operate on existing stored procedures is controlled by object privileges.	CREATE PROCEDURE
[Create] Schema	The user can create schemas. Permission to operate on existing schemas is controlled by object privileges.	CREATE SCHEMA

ADMIN PRIVILEGE	DESCRIPTION	SYNAPSE EQUIVALENT
[Create] Table	The user can create tables. Permission to operate on existing tables is controlled by object privileges.	CREATE TABLE
[Create] Temporary Table	The user can create temporary tables. Permission to operate on existing tables is controlled by object privileges.	CREATE TABLE
[Create] User	The user can create users. Permission to operate on existing users is controlled by object privileges.	CREATE USER
[Create] View	The user can create views. Permission to operate on existing views is controlled by object privileges.	CREATE VIEW

You can automate the migration of these privileges by generating equivalent scripts for Azure Synapse from the Oracle catalog tables, as described earlier in this section.

The next table lists common Oracle object privileges that have a direct equivalent in Azure Synapse.

OBJECT PRIVILEGE	DESCRIPTION	SYNAPSE EQUIVALENT
Alter	The user can modify object attributes. Applies to all objects.	ALTER
Delete	The user can delete table rows. Applies only to tables.	DELETE
Drop	The user can drop objects. Applies to all object types.	DROP
Execute	The user can run user-defined functions, user-defined aggregates, or stored procedures.	EXECUTE
Insert	The user can insert rows into a table. Applies only to tables.	INSERT
List	The user can display an object name, either in a list or in another manner. Applies to all objects.	LIST
Select	The user can select (or query) rows within a table. Applies to tables and views.	SELECT
Truncate	The user can delete all rows from a table. Applies only to tables.	TRUNCATE
Update	The user can modify table rows. Applies to tables only.	UPDATE

For more information about Azure Synapse permissions, see Database engine permissions.

Migrating users, roles, and privileges

So far, we've described a common approach for migrating users, roles, and privileges to Azure Synapse using `CREATE USER`, `CREATE ROLE`, and `GRANT` SQL commands. However, you don't need to migrate all Oracle operations with grantable privileges to the new environment. For example, system management operations aren't applicable to the new environment or the equivalent functionality is automatic or managed outside the database. For users, roles, and the subset of privileges that do have a direct equivalent in the Azure Synapse environment, the following steps describe the migration process:

1. Migrate Oracle schema, table, and view definitions to the Azure Synapse environment. This step migrates only the table definitions not the data.
2. Extract the existing user IDs that you want to migrate from the Oracle system tables, generate a script of `CREATE USER` statements for Azure Synapse, and then run that script in the Azure Synapse environment. Find a way to create new initial passwords, because passwords can't be extracted from the Oracle environment.
3. Extract the existing roles from the Oracle system tables, generate a script of equivalent `CREATE ROLE` statements for Azure Synapse, and then run that script in the Azure Synapse environment.
4. Extract the user/role combinations from the Oracle system tables, generate a script to `GRANT` roles to users in Azure Synapse, and then run that script in the Azure Synapse environment.
5. Extract the relevant privilege information from the Oracle system tables, then generate a script to `GRANT` the appropriate privileges to users and roles in Azure Synapse, and then run that script in the Azure Synapse environment.

Operational considerations

This section discusses how typical Oracle operational tasks can be implemented in Azure Synapse with minimal risk and user impact.

As with all data warehouse products in production, ongoing management tasks are necessary to keep the system running efficiently and provide data for monitoring and auditing. Other operational considerations include resource utilization, capacity planning for future growth, and backup/restore of data.

> **TIP**
> Operational tasks are necessary to keep any data warehouse operating efficiently.

Oracle administration tasks typically fall into two categories:

- **System administration**: system administration is management of the hardware, configuration settings, system status, access, disk space, usage, upgrades, and other tasks.
- **Database administration**: database administration is management of user databases and their content, data loading, data backup, data recovery, and access to data and permissions.

Oracle offers several methods and interfaces that you can use to perform system and database management tasks:

- Oracle Enterprise Manager is Oracle's on-premises management platform. It provides a single pane of glass for managing all of a customer's Oracle deployments, whether in their data centers or in the Oracle Cloud. Through deep integration with Oracle's product stack, Oracle Enterprise Manager provides management and automation support for Oracle applications, databases, middleware, hardware, and engineered systems.
- Oracle Instance Manager provides a UI for high-level administration of Oracle instances. Oracle Instance Manager enables tasks such as startup, shutdown, and log viewing.

- Oracle Database Configuration Assistant is a UI that allows management and configuration of various database features and functionality.
- SQL commands that support administration tasks and queries within a SQL database session. You can run SQL commands from the Oracle SQL*Plus command interpreter, Oracle SQL Developer UI, or through SQL APIs such as ODBC, JDBC, and OLE DB Provider. You must have a database user account to run SQL commands, with appropriate permissions for the queries and tasks that you perform.

While the management and operations tasks for different data warehouses are similar in concept, the individual implementations can differ. Modern cloud-based products such as Azure Synapse tend to incorporate a more automated and "system managed" approach, compared to the more manual approach in legacy environments like Oracle.

The following sections compare Oracle and Azure Synapse options for various operational tasks.

Housekeeping tasks

In most legacy data warehouse environments, regular housekeeping tasks are time-consuming. You can reclaim disk storage space by removing old versions of updated or deleted rows. Or, you can reclaim disk storage space by reorganizing data, log files, and index blocks for efficiency, for example by running `ALTER TABLE... SHRINK SPACE` in Oracle.

> **TIP**
> Housekeeping tasks keep a production warehouse operating efficiently and optimize storage and other resources.

Collecting statistics is a potentially time-consuming task that's required after bulk data ingestion to provide the query optimizer with up-to-date data for its query execution plans.

Oracle has a built-in feature to help with analyzing the quality of statistics, the Optimizer Statistics Advisor. It works through a list of Oracle rules that represent best practices for optimizer statistics. The advisor checks each rule and, where necessary, generates findings, recommendations, and actions that involve calls to the `DBMS_STATS` package to take corrective measures. Users can see the list of rules in the `V$STATS_ADVISOR_RULES` view, as shown in the following screenshot.

```
SELECT *
FROM   v$stats_advisor_rules
ORDER BY rule_id;

   RULE_ID NAME                            RULE_TYPE DESCRIPTION                                                                  CON_ID
---------- ------------------------------- --------- ------------------------------------------------------------------------- ----------
         0                                 SYSTEM                                                                                       0
         1 UseAutoJob                      SYSTEM    Use Auto Job for Statistics Collection                                             0
         2 CompleteAutoJob                 SYSTEM    Auto Statistics Gather Job should complete successfully                            0
         3 MaintainStatsHistory            SYSTEM    Maintain Statistics History                                                        0
         4 UseConcurrent                   SYSTEM    Use Concurrent preference for Statistics Collection                                0
         5 UseDefaultPreference            SYSTEM    Use Default Preference for Stats Collection                                        0
         6 TurnOnSQLPlanDirective          SYSTEM    SQL Plan Directives should not be disabled                                         0
         7 AvoidSetProcedures              OPERATION Avoid Set Statistics Procedures                                                    0
         8 UseDefaultParams                OPERATION Use Default Parameters in Statistics Collection Procedures                          0
         9 UseGatherSchemaStats            OPERATION Use gather_schema_stats procedure                                                   0
        10 AvoidInefficientStatsOprSeq     OPERATION Avoid inefficient statistics operation sequences                                   0
        11 AvoidUnnecessaryStatsCollection OBJECT    Avoid unnecessary statistics collection                                            0
        12 AvoidStaleStats                 OBJECT    Avoid objects with stale or no statistics                                          0
        13 GatherStatsAfterBulkDML         OBJECT    Do not gather statistics right before bulk DML                                     0
        14 LockVolatileTable               OBJECT    Statistics for objects with volatile data should be locked                         0
        15 UnlockNonVolatileTable          OBJECT    Statistics for objects with non-volatile should not be locked                      0
        16 MaintainStatsConsistency        OBJECT    Statistics of dependent objects should be consistent                               0
        17 AvoidDropRecreate               OBJECT    Avoid drop and recreate object seqauences                                          0
        18 UseIncremental                  OBJECT    Statistics should be maintained incrementally when it is beneficial                0
        19 NotUseIncremental               OBJECT    Statistics should not be maintained incrementally when it is not beneficial        0
        20 AvoidOutOfRange                 OBJECT    Avoid Out of Range Histogram endpoints                                             0
        21 UseAutoDegree                   OBJECT    Use Auto Degree for statistics collection                                          0
        22 UseDefaultObjectPreference      OBJECT    Use Default Object Preference for statistics collection                            0
        23 AvoidAnalyzeTable               OBJECT    Avoid using analyze table commands for statistics collection                       0

24 rows selected.

SQL>
```

An Oracle database contains many log tables in the data dictionary, which accumulate data, either automatically or after certain features are enabled. Because log data grows over time, purge older information to avoid using up permanent space. Oracle provides options to automate log maintenance.

Azure Synapse can automatically create statistics so that they're available when needed. You can defragment indexes and data blocks manually, on a scheduled basis, or automatically. By using native built-in Azure capabilities, you reduce the migration effort.

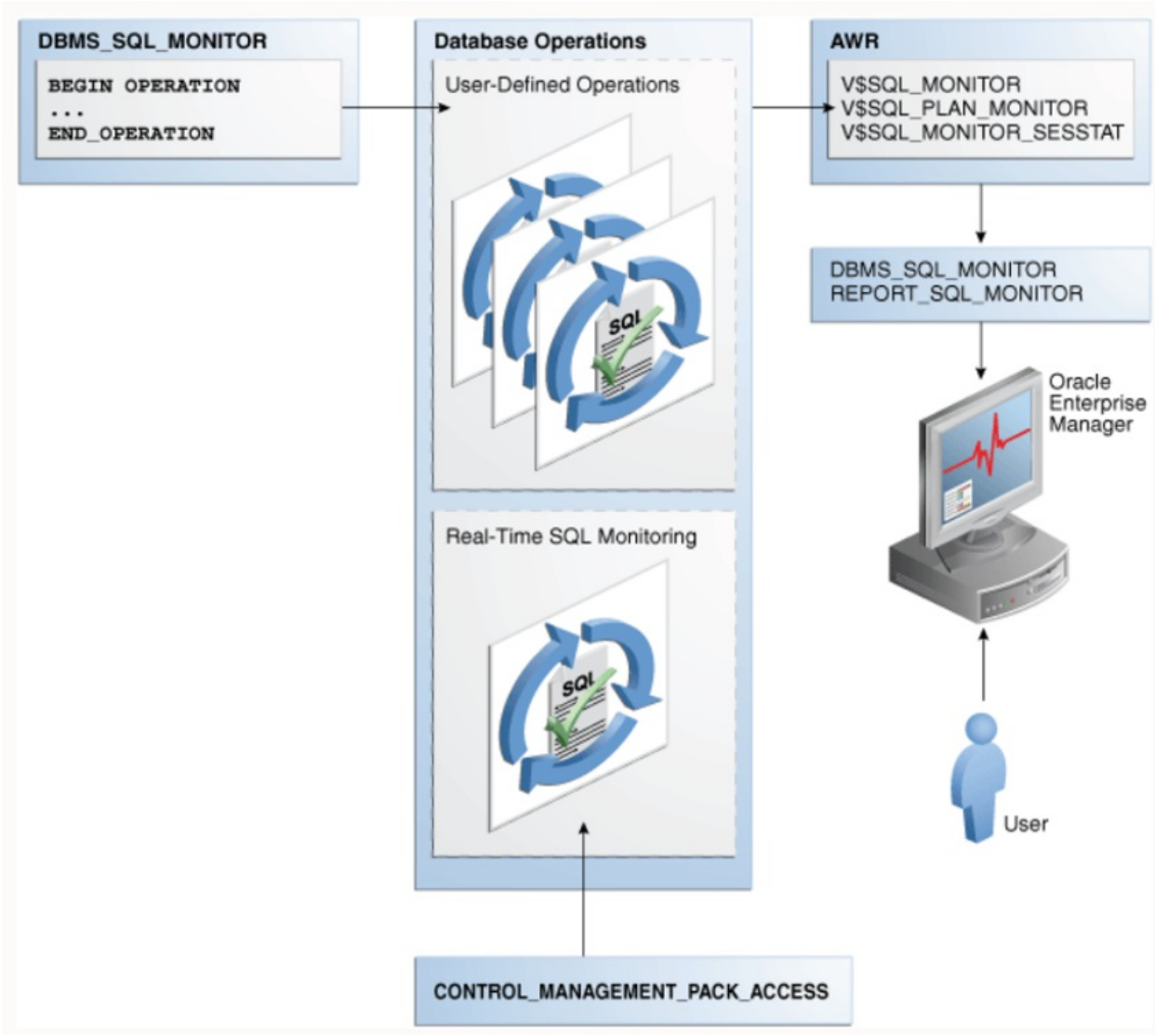
> **TIP**
> Automate and monitor housekeeping tasks in Azure.

Monitoring and auditing

Oracle Enterprise Manager includes tools to monitor various aspects of one or more Oracle systems, such as activity, performance, queuing, and resource utilization. Oracle Enterprise Manager has an interactive UI that lets users drill down into the low-level detail of any chart.

> **TIP**
> Oracle Enterprise Manager is the recommended method of monitoring and logging in Oracle systems.

The following diagram provides an overview of the monitoring environment in an Oracle data warehouse.

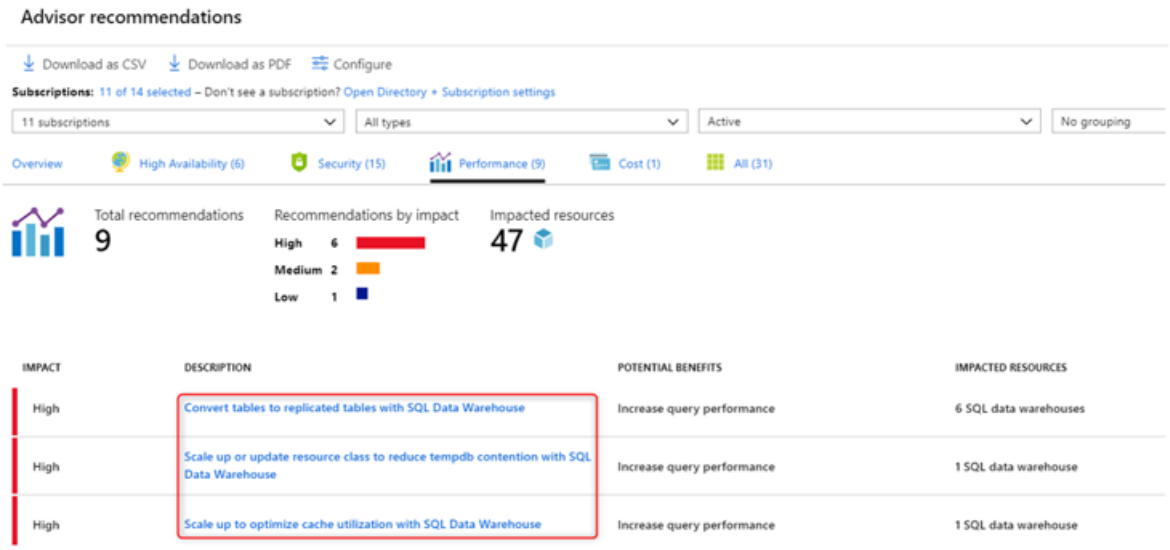


Azure Synapse also provides a rich monitoring experience within the Azure portal to provide insights into your

data warehouse workload. The Azure portal is the recommended tool for monitoring your data warehouse because it provides configurable retention periods, alerts, recommendations, and customizable charts and dashboards for metrics and logs.

> **TIP**
> The Azure portal provides a UI to manage monitoring and auditing tasks for all Azure data and processes.

The Azure portal can also provide recommendations for performance enhancements, as shown in the following screenshot.



The portal supports integration with other Azure monitoring services, such as Operations Management Suite (OMS) and Azure Monitor, to provide an integrated monitoring experience of the data warehouse and the entire Azure analytics platform. For more information, see Azure Synapse operations and management options.

High availability (HA) and disaster recovery (DR)

Since its initial release in 1979, the Oracle environment has evolved to encompass numerous features required by enterprise customers, including options for high availability (HA) and disaster recovery (DR). The latest announcement in this area is Maximum Availability Architecture (MAA), which includes reference architectures for four levels of HA and DR:

- **Bronze tier**: a single-instance HA architecture
- **Silver tier**: HA with automatic failover
- **Gold tier**: comprehensive HA and DR
- **Platinum tier**: zero outage for platinum-ready applications

Azure Synapse uses database snapshots to provide HA of the data warehouse. A data warehouse snapshot creates a restore point that you can use to restore a data warehouse to a previous state. Because Azure Synapse is a distributed system, a data warehouse snapshot consists of many files stored in Azure Storage. Snapshots capture incremental changes to the data stored in your data warehouse.

> **TIP**
> Azure Synapse creates snapshots automatically to ensure fast recovery time.

Azure Synapse automatically takes snapshots throughout the day and creates restore points that are available for seven days. You can't change this retention period. Azure Synapse supports an eight-hour recovery point objective (RPO). You can restore a data warehouse in the primary region from any one of the snapshots taken in

the past seven days.

> **TIP**
> User-defined snapshots can be used to define a recovery point before key updates.

Azure Synapse supports user-defined restore points, which are created from manually triggered snapshots. By creating restore points before and after large data warehouse modifications, you ensure that the restore points are logically consistent. The user-defined restore points augment data protection and reduce recovery time if there are workload interruptions or user errors.

In addition to snapshots, Azure Synapse performs a standard geo-backup once per day to a paired data center. The RPO for a geo-restore is 24 hours. You can restore the geo-backup to a server in any region where Azure Synapse is supported. A geo-backup ensures that a data warehouse can be restored if restore points in the primary region aren't available.

> **TIP**
> Microsoft Azure provides automatic backups to a separate geographical location to enable DR.

Workload management

Oracle provides utilities such as Enterprise Manager and Database Resource Manager (DBRM) for managing workloads. These utilities include features such as load balancing across large clusters, parallel query execution, performance measurement, and prioritization. Many of these features can be automated so that the system becomes to some extent self-tuning.

> **TIP**
> A typical production data warehouse concurrently runs mixed workloads with different resource usage characteristics.

Azure Synapse automatically logs resource utilization statistics. Metrics include usage statistics for CPU, memory, cache, I/O, and temporary workspace for each query. Azure Synapse also logs connectivity information, such as failed connection attempts.

> **TIP**
> Low-level and system-wide metrics are automatically logged within Azure.

In Azure Synapse, resource classes are pre-determined resource limits that govern compute resources and concurrency for query execution. Resource classes help you manage your workload by setting limits on the number of queries that run concurrently and on the compute resources assigned to each query. There's a trade-off between memory and concurrency.

Azure Synapse supports these basic workload management concepts:

- **Workload classification**: you can assign a request to a workload group to set importance levels.
- **Workload importance**: you can influence the order in which a request gets access to resources. By default, queries are released from the queue on a first-in, first-out basis as resources become available. Workload importance allows higher priority queries to receive resources immediately regardless of the queue.
- **Workload isolation**: you can reserve resources for a workload group, assign maximum and minimum

usage for varying resources, limit the resources a group of requests can consume can, and set a timeout value to automatically kill runaway queries.

Running mixed workloads can pose resource challenges on busy systems. A successful workload management scheme effectively manages resources, ensures highly efficient resource utilization, and maximizes return on investment (ROI). The workload classification, workload importance, and workload isolation gives more control over how workload utilizes system resources.

You can use the workload metrics that Azure Synapse collects for capacity planning, for example to determine the resources required for extra users or a larger application workload. You can also use workload metrics to plan scale up/down of compute resources for cost-effective support of peaky workloads.

The workload management guide describes the techniques to analyze the workload, manage and monitor workload importance](../../sql-data-warehouse/sql-data-warehouse-how-to-manage-and-monitor-workload-importance.md), and the steps to convert a resource class to a workload group. Use the Azure portal and T-SQL queries on DMVs to monitor the workload to ensure that the applicable resources are efficiently utilized. Azure Synapse provides a set of Dynamic Management Views (DMVs) for monitoring all aspects of workload management. These views are useful when actively troubleshooting and identifying performance bottlenecks in your workload.

For more information on workload management in Azure Synapse, see Workload management with resource classes.

Scale compute resources

The Azure Synapse architecture separates storage and compute, allowing each to scale independently. As a result, compute resources can be scaled to meet performance demands independent of data storage. You can also pause and resume compute resources. Another benefit of this architecture is that billing for compute and storage is separate. If a data warehouse isn't in use, you can save on compute costs by pausing compute.

> **TIP**
> A major benefit of Azure is the ability to independently scale up and down compute resources on demand to handle peaky workloads cost-effectively.

You can scale compute resources up or down by adjusting the data warehouse units (DWU) setting for a data warehouse. Load and query performance will increase linearly as you allocate more DWUs.

If you increase DWUs, the number of compute nodes increase, which adds more compute power and supports more parallel processing. As the number of compute nodes increase, the number of distributions per compute node decrease, providing more compute power and parallel processing for queries. Similarly, if you decrease DWUs, the number of compute nodes decrease, which reduces the compute resources for queries.

Next steps

To learn more about visualization and reporting, see the next article in this series: Visualization and reporting for Oracle migrations.

Visualization and reporting for Oracle migrations

8/19/2022 • 20 minutes to read • Edit Online

This article is part four of a seven-part series that provides guidance on how to migrate from Oracle to Azure Synapse Analytics. The focus of this article is best practices for visualization and reporting.

Access Azure Synapse Analytics using Microsoft and third-party BI tools

Organizations access data warehouses and data marts using a range of business intelligence (BI) tools and applications. Some examples of BI products are:

- Microsoft BI tools, such as Power BI.
- Office applications, such as Microsoft Excel spreadsheets.
- Third-party BI tools from different vendors.
- Custom analytics applications with embedded BI tool functionality.
- Operational applications that support on-demand BI by running queries and reports on a BI platform that in turn queries data in a data warehouse or data mart.
- Interactive data science development tools, such as Azure Synapse Spark Notebooks, Azure Machine Learning, RStudio, and Jupyter Notebooks.

If you migrate visualization and reporting as part of your data warehouse migration, all existing queries, reports, and dashboards generated by BI products need to run in the new environment. Your BI products must yield the same results on Azure Synapse as they did in your legacy data warehouse environment.

For consistent results after migration, all BI tools and application dependencies must work after you've migrated your data warehouse schema and data to Azure Synapse. The dependencies include less visible aspects, such as access and security. When you address access and security, ensure that you migrate:

- Authentication so users can sign into the data warehouse and data mart databases on Azure Synapse.
- All users to Azure Synapse.
- All user groups to Azure Synapse.
- All roles to Azure Synapse.
- All authorization privileges governing access control to Azure Synapse.
- User, role, and privilege assignments to mirror what you had in your existing data warehouse before migration. For example:
    - Database object privileges assigned to roles
    - Roles assigned to user groups
    - Users assigned to user groups and/or roles

Access and security are important considerations for data access in the migrated system and are discussed in more detail in Security, access, and operations for Oracle migrations.

TIP

Existing users, user groups, roles, and assignments of access security privileges need to be migrated first for migration of reports and visualizations to succeed.

Migrate all required data to ensure that the reports and dashboards that query data in the legacy environment produce the same results in Azure Synapse.

Business users will expect a seamless migration, with no surprises that destroy their confidence in the migrated system on Azure Synapse. Take care to allay any fears that your users might have through good communication. Your users will expect that:

- Table structure remains the same when directly referred to in queries.
- Table and column names remain the same when directly referred to in queries. For instance, calculated fields defined on columns in BI tools shouldn't fail when aggregate reports are produced.
- Historical analysis remains the same.
- Data types remain the same, if possible.
- Query behavior remains the same.
- ODBC/JDBC drivers are tested to ensure that query behavior remains the same.

TIP

Communication and business user involvement are critical to success.

If BI tools query views in the underlying data warehouse or data mart database, will those views still work after the migration? Some views might not work if there are proprietary SQL extensions specific to your legacy data warehouse DBMS that have no equivalent in Azure Synapse. If so, you need to know about those incompatibilities and find a way to resolve them.

TIP

Views and SQL queries using proprietary SQL query extensions are likely to result in incompatibilities that impact BI reports and dashboards.

Other issues, like the behavior of `NULL` values or data type variations across DBMS platforms, need to be tested to ensure that even slight differences don't exist in calculation results. Minimize those issues and take all necessary steps to shield business users from being affected by them. Depending on your legacy data warehouse environment, third-party tools that can help hide the differences between the legacy and new environments so that BI tools and applications run unchanged.

Testing is critical to visualization and report migration. You need a test suite and agreed-on test data to run and rerun tests in both environments. A test harness is also useful, and a few are mentioned in this guide. Also, it's important to involve business users in the testing aspect of the migration to keep confidence high and to keep them engaged and part of the project.

TIP

Use repeatable tests to ensure reports, dashboards, and other visualizations migrate successfully.

You might be thinking about switching BI tools, for example to migrate to Power BI. The temptation is to make such changes at the same time you're migrating your schema, data, ETL processing, and more. However, to minimize risk, it's better to migrate to Azure Synapse first and get everything working before undertaking further modernization.

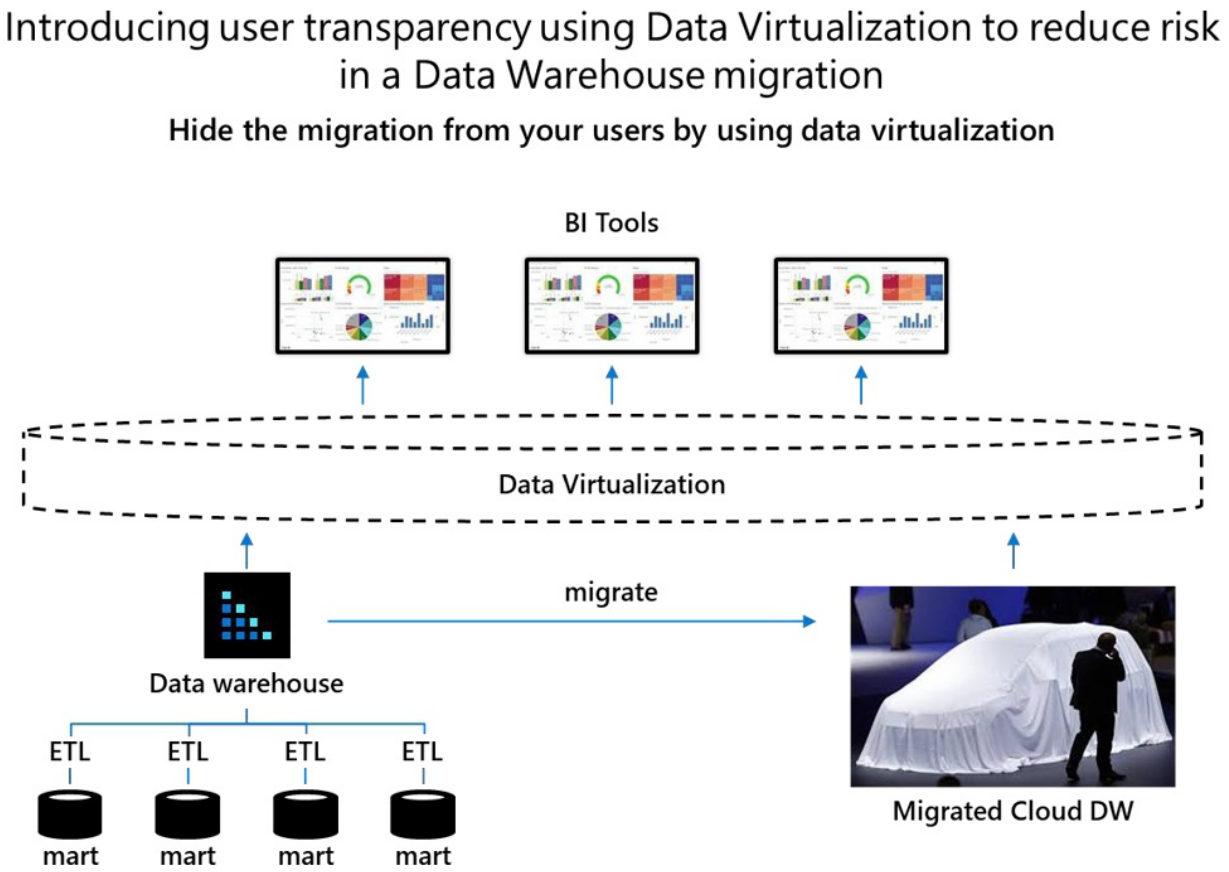
If your existing BI tools run on-premises, ensure they can connect to Azure Synapse through your firewall so you're able to run comparisons against both environments. Alternatively, if the vendor of your existing BI tools offers their product on Azure, you can try it there. The same applies for applications running on-premises that embed BI or call your BI server on demand, for example by requesting a "headless report" with XML or JSON data.

There's a lot to think about here, so let's take a closer look.

Use data virtualization to minimize the impact of migration on BI tools and reports

During migration, you might be tempted to fulfill long-term requirements like opening business requests, adding missing data, or implementing new features. However, such changes can affect BI tool access to your data warehouse, especially if the change involves structural changes to your data model. If you want to adopt an agile data modeling technique or implement structural changes, do so *after* migration.

One way to minimize the effect of schema changes or other structural changes on your BI tools is to introduce data virtualization between the BI tools and your data warehouse and data marts. The following diagram shows how data virtualization can hide a migration from users.



Data virtualization breaks the dependency between business users utilizing self-service BI tools and the physical schema of the underlying data warehouse and data marts that are being migrated.

> **TIP**
> Data virtualization allows you to shield business users from structural changes during migration so they remain unaware of those changes. Structural changes include schema alterations that tune your data model for Azure Synapse.

With data virtualization, any schema alterations made during a migration to Azure Synapse, for example to optimize performance, can be hidden from business users because they only have access to virtual tables in the data virtualization layer. And, if you make structural changes, you only need to update the mappings between the data warehouse or data marts and any virtual tables. With data virtualization, users remain unaware of structural changes. Microsoft partners provide data virtualization software.

Identify high-priority reports to migrate first

A key question when migrating your existing reports and dashboards to Azure Synapse is which ones to migrate first. Several factors might drive that decision, such as:

- Usage
- Business value
- Ease of migration
- Data migration strategy

The following sections discuss these factors.

Whatever your decision, it must involve your business users because they produce the reports, dashboards, and other visualizations, and make business decisions based on insights from those items. Everyone benefits when you can:

- Migrate reports and dashboards seamlessly,
- Migrate reports and dashboards with minimal effort, and
- Point your BI tool(s) at Azure Synapse instead of your legacy data warehouse system, and get like-for-like reports, dashboards, and other visualizations.

Migrate reports based on usage

Usage is often an indicator of business value. Unused reports and dashboards clearly don't contribute to business decisions or offer current value. If you don't have a way to find out which reports and dashboards are unused, you can use one of the several BI tools that provide usage statistics.

If your legacy data warehouse has been up and running for years, there's a good chance you have hundreds, if not thousands, of reports in existence. It's worth compiling an inventory of reports and dashboards and identifying their business purpose and usage statistics.

For unused reports, determine whether to decommission them to reduce your migration effort. A key question when deciding whether to decommission an unused report is whether the report is unused because people don't know it exists, because it offers no business value, or because it's been superseded by another report.

Migrate reports based on business value

Usage alone isn't always a good indicator of business value. You might want to consider the extent to which a report's insights contribute to business value. One way to do that is to evaluate the profitability of every business decision that relies on the report and the extent of the reliance. However, that information is unlikely to be readily available in most organizations.

Another way to evaluate business value is to look at the alignment of a report with business strategy. The business strategy set by your executive typically lays out strategic business objectives (SBOs), key performance

indicators (KPIs), KPI targets that need to be achieved, and who is accountable for achieving them. You can classify a report by which SBOs the report contributes to, such as fraud reduction, improved customer engagement, and optimized business operations. Then, you can prioritize for migration the reports and dashboards that are associated with high-priority objectives. In this way, the initial migration can deliver business value in a strategic area.

Another way to evaluate business value is to classify reports and dashboards as operational, tactical, or strategic to identify at which business level they're used. SBOs require contributions at all these levels. By knowing which reports and dashboards are used, at what level, and what objectives they're associated with, you're able to focus the initial migration on high-priority business value. You can use the following *business strategy objective* table to evaluate reports and dashboards.

LEVEL	REPORT / DASHBOARD NAME	BUSINESS PURPOSE	DEPARTMENT USED	USAGE FREQUENCY	BUSINESS PRIORITY
Strategic					
Tactical					
Operational					

Metadata discovery tools like Azure Data Catalog let business users tag and rate data sources to enrich the metadata for those data sources to assist with their discovery and classification. You can use the metadata for a report or dashboard to help you understand its business value. Without such tools, understanding the contribution of reports and dashboards to business value is likely to be a time-consuming task, whether you're migrating or not.

Migrate reports based on data migration strategy

If your migration strategy is based on migrating data marts first, then the order of data mart migration will affect which reports and dashboards are migrated first. If your strategy is based on business value, the order in which you migrate data marts to Azure Synapse will reflect business priorities. Metadata discovery tools can help you implement your strategy by showing you which data mart tables supply data for which reports.

> **TIP**
> Your data migration strategy affects which reports and visualizations get migrated first.

Migration incompatibility issues that can affect reports and visualizations

BI tools produce reports, dashboards, and other visualizations by issuing SQL queries that access physical tables and/or views in your data warehouse or data mart. When you migrate your legacy data warehouse to Azure Synapse, several factors can affect the ease of migration of reports, dashboards, and other visualizations. Those factors include:

- Schema incompatibilities between the environments.
- SQL incompatibilities between the environments.

Schema incompatibilities

During a migration, schema incompatibilities in the data warehouse or data mart tables that supply data for reports, dashboards, and other visualizations can be:

- Non-standard table types in your legacy data warehouse DBMS that don't have an equivalent in Azure

Synapse.

- Data types in your legacy data warehouse DBMS that don't have an equivalent in Azure Synapse.

In most cases, there's a workaround to the incompatibilities. For example, you can migrate the data in an unsupported table type into a standard table with appropriate data types and indexed or partitioned on a date/time column. Similarly, it might be possible to represent unsupported data types in another type of column and perform calculations in Azure Synapse to achieve the same results.

> **TIP**
> Schema incompatibilities include legacy warehouse DBMS table types and data types that are unsupported on Azure Synapse.

To identify the reports affected by schema incompatibilities, run queries against the system catalog of your legacy data warehouse to identify the tables with unsupported data types. Then, you can use metadata from your BI tool to identify the reports that access data in those tables. For more information about how to identify object type incompatibilities, see Unsupported Oracle database object types.

> **TIP**
> Query the system catalog of your legacy warehouse DBMS to identify schema incompatibilities with Azure Synapse.

The effect of schema incompatibilities on reports, dashboards, and other visualizations might be less than you think because many BI tools don't support the less generic data types. As a result, your legacy data warehouse might already have views that `CAST` unsupported data types to more generic types.

SQL incompatibilities

During a migration, SQL incompatibilities are likely to affect any report, dashboard, or other visualization in an application or tool that:

- Accesses legacy data warehouse DBMS views that include proprietary SQL functions that have no equivalent in Azure Synapse.
- Issues SQL queries that include proprietary SQL functions, specific to the SQL dialect of your legacy environment, that have no equivalent in Azure Synapse.

Gauge the impact of SQL incompatibilities on your reporting portfolio

Your reporting portfolio might include embedded query services, reports, dashboards, and other visualizations. Don't rely on the documentation associated with those items to gauge the effect of SQL incompatibilities on the migration of your reporting portfolio to Azure Synapse. You need to use a more precise way to assess the effect of SQL incompatibilities.

Use EXPLAIN statements to find SQL incompatibilities

You can find SQL incompatibilities by reviewing the logs of recent SQL activity in your legacy Oracle data warehouse. Use a script to extract a representative set of SQL statements to a file. Then, prefix each SQL statement with an `EXPLAIN` statement, and run those `EXPLAIN` statements in Azure Synapse. Any SQL statements containing proprietary unsupported SQL extensions will be rejected by Azure Synapse when the `EXPLAIN` statements are executed. This approach lets you assess the extent of SQL incompatibilities.

Metadata from your legacy data warehouse DBMS can also help you identify incompatible views. As before, capture a representative set of SQL statements from the applicable logs, prefix each SQL statement with an `EXPLAIN` statement, and run those `EXPLAIN` statements in Azure Synapse to identify views with incompatible SQL.

> **TIP**
> Gauge the impact of SQL incompatibilities by harvesting your DBMS log files and running `EXPLAIN` statements.

Test report and dashboard migration to Azure Synapse Analytics

A key element of data warehouse migration is testing of reports and dashboards in Azure Synapse to verify the migration has worked. Define a series of tests and a set of required outcomes for each test that you will run to verify success. Test and compare the reports and dashboards across your existing and migrated data warehouse systems to:

- Identify whether any schema changes that were made during migration affected the ability of reports to run, report results, or the corresponding report visualizations. An example of a schema change is if you mapped an incompatible data type to an equivalent data type that's supported in Azure Synapse.
- Verify that all users are migrated.
- Verify that all roles are migrated, and users are assigned to those roles.
- Verify that all data access security privileges are migrated to ensure access control list (ACL) migration.
- Ensure consistent results for all known queries, reports, and dashboards.
- Ensure that data and ETL migration is complete and error-free.
- Ensure that data privacy is upheld.
- Test performance and scalability.
- Test analytical functionality.

> **TIP**
> Test and tune performance to minimize compute costs.

For information about how to migrate users, user groups, roles, and privileges, see Security, access, and operations for Oracle migrations.

Automate testing as much as possible to make each test repeatable and to support a consistent approach to evaluating test results. Automation works well for known regular reports, and can be managed via Azure Synapse pipelines or Azure Data Factory orchestration. If you already have a suite of test queries in place for regression testing, you can use the existing testing tools to automate post migration testing.

> **TIP**
> Best practice is to build an automated test suite to make tests repeatable.

Ad-hoc analysis and reporting are more challenging and require compilation of a set of tests to verify that the same reports and dashboards from before and after migration are consistent. If you find inconsistencies, then your ability to compare metadata lineage across the original and migrated systems during migration testing becomes crucial. That comparison can highlight differences and pinpoint where inconsistencies originated, when detection by other means is difficult.

> **TIP**
> Leverage tools that compare metadata lineage to verify results.

Analyze lineage to understand dependencies between reports, dashboards, and data

Your understanding of lineage is a critical factor in the successful migration of reports and dashboards. Lineage is metadata that shows the journey of migrated data so you can track its path from a report or dashboard all the way back to the data source. Lineage shows how data has traveled from point to point, its location in the data warehouse and/or data mart, and which reports and dashboards use it. Lineage can help you understand what happens to data as it travels through different data stores, such as files and databases, different ETL pipelines, and into reports. When business users have access to data lineage, it improves trust, instills confidence, and supports informed business decisions.

> **TIP**
> Your ability to access metadata and data lineage from reports all the way back to a data source is critical for verifying that migrated reports work correctly.

In multi-vendor data warehouse environments, business analysts in BI teams might map out data lineage. For example, if you use different vendors for ETL, data warehouse, and reporting, and each vendor has its own metadata repository, then figuring out where a specific data element in a report came from can be challenging and time-consuming.

> **TIP**
> Tools that automate the collection of metadata and show end-to-end lineage in a multi-vendor environment are valuable during a migration.

To migrate seamlessly from a legacy data warehouse to Azure Synapse, use end-to-end data lineage to prove like-for-like migration when you're comparing the reports and dashboards generated by each environment. To show the end-to-end data journey, you'll need to capture and integrate metadata from several tools. Having access to tools that support automated metadata discovery and data lineage, helps you identify duplicate reports or ETL processes, and find reports that rely on obsolete, questionable, or non-existent data sources. You can use that information to reduce the number of reports and ETL processes that you migrate.

You can also compare the end-to-end lineage of a report in Azure Synapse to the end-to-end lineage of the same report in your legacy environment to check for differences that might have inadvertently occurred during migration. This type of comparison is exceptionally useful when you need to test and verify migration success.

Data lineage visualization not only reduces time, effort, and error in the migration process, but also enables faster migration.

By using automated metadata discovery and data lineage tools that compare lineage, you can verify that a report in Azure Synapse that's produced from migrated data is produced in the same way in your legacy environment. This capability also helps you determine:

- What data needs to be migrated to ensure successful report and dashboard execution in Azure Synapse.
- What transformations have been and should be performed to ensure successful execution in Azure Synapse.

- How to reduce report duplication.

Automated metadata discovery and data lineage tools substantially simplify the migration process because they help businesses become more aware of their data assets and to know what needs to be migrated to Azure Synapse to achieve a solid reporting environment.

Several ETL tools provide end-to-end lineage capability, so check whether your existing ETL tool has that capability if you plan to use it with Azure Synapse. Azure Synapse pipelines or Data Factory both support the ability to view lineage in mapping flows. Microsoft partners also provide automated metadata discovery, data lineage, and lineage comparison tools.

Migrate BI tool semantic layers to Azure Synapse Analytics

Some BI tools have what is known as a semantic metadata layer. That layer simplifies business user access to the underlying physical data structures in a data warehouse or data mart database. The semantic metadata layer simplifies access by providing high-level objects like dimensions, measures, hierarchies, calculated metrics, and joins. The high-level objects use business terms that are familiar to business analysts, and map to physical data structures in your data warehouse or data mart.

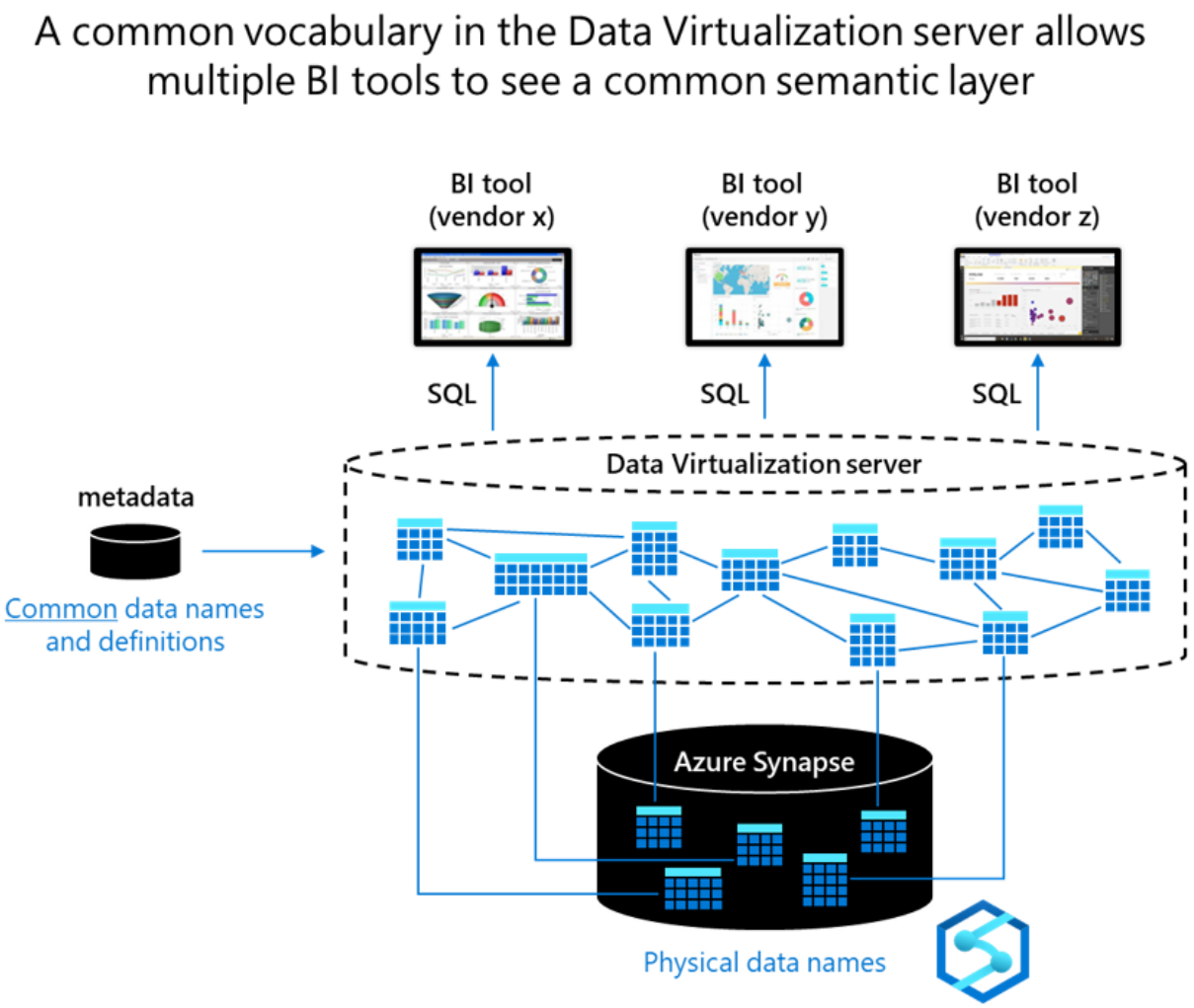
> **TIP**
> Some BI tools have semantic layers that simplify business user access to physical data structures in your data warehouse or data mart.

In a data warehouse migration, you might be forced to change column or table names. For example, Oracle allows a `#` character in table names, but Azure Synapse only allows `#` as a table name prefix to indicate a temporary table. In Oracle, TEMPORARY TABLES do not necessarily have a "#" in the name, but in Synapse they must. You may need to do some rework to change table mappings in such cases.

To achieve consistency across multiple BI tools, create a universal semantic layer by using a data virtualization server that sits between BI tools and applications and Azure Synapse. In the data virtualization server, use common data names for high-level objects like dimensions, measures, hierarchies, and joins. That way you configure everything, including calculated fields, joins, and mappings, only once instead of in every tool. Then, point all BI tools at the data virtualization server.

> **TIP**
> Use data virtualization to create a common semantic layer to guarantee consistency across all BI tools in an Azure Synapse environment.

With data virtualization, you get consistency across all BI tools and break the dependency between BI tools and applications and the underlying physical data structures in Azure Synapse. Microsoft partners can help you achieve consistency in Azure. The following diagram shows how a common vocabulary in the data virtualization server lets multiple BI tools see a common semantic layer.



Conclusions

In a lift and shift data warehouse migration, most reports, dashboards, and other visualizations should migrate easily.

During a migration from a legacy environment, you might find that data in the legacy data warehouse or data mart tables is stored in unsupported data types. Or, you may find legacy data warehouse views that include proprietary SQL with no equivalent in Azure Synapse. If so, you'll need to resolve those issues to ensure a successful migration to Azure Synapse.

Don't rely on user-maintained documentation to identify where issues are located. Instead, use `EXPLAIN` statements because they're a quick, pragmatic way to identify SQL incompatibilities. Rework the incompatible SQL statements to achieve equivalent functionality in Azure Synapse. Also, use automated metadata discovery and lineage tools to understand dependencies, find duplicate reports, and identify invalid reports that rely on obsolete, questionable, or non-existent data sources. Use lineage tools to compare lineage to verify that reports running in your legacy data warehouse environment are produced identically in Azure Synapse.

Don't migrate reports that you no longer use. BI tool usage data can help you determine which reports aren't in use. For the reports, dashboards, and other visualizations that you do want to migrate, migrate all users, user groups, roles, and privileges. If you're using business value to drive your report migration strategy, associate reports with strategic business objectives and priorities to help identify the contribution of report insights to specific objectives. If you're migrating data mart by data mart, use metadata to identify which reports are dependent on which tables and views, so you can make an informed decision about which data marts to migrate first.

> **TIP**
> Identify incompatibilities early to gauge the extent of the migration effort. Migrate your users, group roles, and privilege assignments. Only migrate the reports and visualizations that are used and are contributing to business value.

Structural changes to the data model of your data warehouse or data mart can occur during a migration. Consider using data virtualization to shield BI tools and applications from structural changes. With data virtualization, you can use a common vocabulary to define a common semantic layer. The common semantic layer guarantees consistent common data names, definitions, metrics, hierarchies, and joins across all BI tools and applications in the new Azure Synapse environment.

Next steps

To learn more about minimizing SQL issues, see the next article in this series: Minimizing SQL issues for Oracle migrations.

Minimize SQL issues for Oracle migrations

8/19/2022 • 15 minutes to read • Edit Online

This article is part five of a seven-part series that provides guidance on how to migrate from Oracle to Azure Synapse Analytics. The focus of this article is best practices for minimizing SQL issues.

Overview

Characteristics of Oracle environments

Oracle's initial database product, released in 1979, was a commercial SQL relational database for on-line transaction processing (OLTP) applications—with much lower transaction rates than today. Since that initial release, the Oracle environment has evolved to become far more complex and encompasses numerous features. The features include client-server architectures, distributed databases, parallel processing, data analytics, high availability, data warehousing, data in-memory techniques, and support for cloud-based instances.

> **TIP**
> Oracle pioneered the "data warehouse appliance" concept in the early 2000's.

Due to the cost and complexity of maintaining and upgrading legacy on-premises Oracle environments, many existing Oracle users want to take advantage of the innovations provided by cloud environments. Modern cloud environments, such as cloud, IaaS, and PaaS, let you delegate tasks like infrastructure maintenance and platform development to the cloud provider.

Many data warehouses that support complex analytic SQL queries on large data volumes use Oracle technologies. These data warehouses commonly have a dimensional data model, such as star or snowflake schemas, and use data marts for individual departments.

> **TIP**
> Many existing Oracle installations are data warehouses that use a dimensional data model.

The combination of SQL and dimensional data models in Oracle simplifies migration to Azure Synapse because the SQL and basic data model concepts are transferable. Microsoft recommends moving your existing data model as-is to Azure to reduce risk, effort, and migration time. Although your migration plan might include a change in the underlying data model, such as a move from an Inmon model to a data vault, it makes sense to initially perform an as-is migration. After the initial migration, you can then make changes within the Azure cloud environment to take advantage of its performance, elastic scalability, built-in features, and cost benefits.

Although the SQL language is standardized, individual vendors sometimes implement proprietary extensions. As a result, you might find SQL differences during your migration that require workarounds in Azure Synapse.

Use Azure facilities to implement a metadata-driven migration

You can automate and orchestrate the migration process by using the capabilities of the Azure environment. This approach minimizes the performance hit on the existing Oracle environment, which may already be running close to capacity.

Azure Data Factory is a cloud-based data integration service that supports creating data-driven workflows in the cloud to orchestrate and automate data movement and data transformation. You can use Data Factory to create and schedule data-driven workflows (pipelines) that ingest data from disparate data stores. Data Factory can

process and transform data by using compute services such as Azure HDInsight Hadoop, Spark, Azure Data Lake Analytics, and Azure Machine Learning.

Azure also includes Azure Database Migration Services to help you plan and perform a migration from environments such as Oracle. SQL Server Migration Assistant (SSMA) for Oracle can automate migration of Oracle databases, including in some cases functions and procedural code.

> **TIP**
> Automate the migration process by using Azure Data Factory capabilities.

When you're planning to use Azure facilities, such as Data Factory, to manage the migration process, first create metadata that lists all the data tables that need to be migrated and their location.

SQL DDL differences between Oracle and Azure Synapse

The ANSI SQL standard defines the basic syntax for Data Definition Language (DDL) commands. Some DDL commands, like `CREATE TABLE` and `CREATE VIEW`, are common to both Oracle and Azure Synapse, but have been extended to provide implementation-specific features such as indexing, table distribution, and partitioning options.

> **TIP**
> SQL DDL commands `CREATE TABLE` and `CREATE VIEW` have standard core elements but are also used to define implementation-specific options.

The following sections discuss the Oracle-specific options that need to be considered during a migration to Azure Synapse.

Table/view considerations

When you migrate tables between different environments, typically only the raw data and the metadata that describes it physically migrate. Other database elements from the source system, such as indexes and log files, usually aren't migrated because they might be unnecessary or implemented differently in the new environment. For example, the `TEMPORARY` option within Oracle's `CREATE TABLE` syntax is equivalent to prefixing a table name with the `#` character in Azure Synapse.

Performance optimizations in the source environment, such as indexes, indicate where you might add performance optimization in the new target environment. For example, if bit-mapped indexes are frequently used in queries within the source Oracle environment, that suggests that a non-clustered index should be created within Azure Synapse. Other native performance optimization techniques such as table replication may be more applicable than straight like-for-like index creation. SSMA for Oracle can provide migration recommendations for table distribution and indexing.

> **TIP**
> Existing indexes indicate candidates for indexing in the migrated warehouse.

SQL view definitions contain SQL Data Manipulation Language (DML) statements that define the view, typically with one or more `SELECT` statements. When you migrate `CREATE VIEW` statements, take into account the DML differences between Oracle and Azure Synapse.

Unsupported Oracle database object types

Oracle-specific features can often be replaced by Azure Synapse features. However, some Oracle database

objects aren't directly supported in Azure Synapse. The following list of unsupported Oracle database objects describes how you can achieve equivalent functionality in Azure Synapse:

- **Indexing options**: in Oracle, several indexing options, such as bit-mapped indexes, function-based indexes, and domain indexes, have no direct equivalent in Azure Synapse. Although Azure Synapse doesn't support those index types, you can achieve a similar reduction in disk I/O by using user-defined index types and/or partitioning Reducing disk I/O improves query performance.

  You can find out which columns are indexed and their index type by querying system catalog tables and views, such as `ALL_INDEXES`, `DBA_INDEXES`, `USER_INDEXES`, and `DBA_IND_COL`. Or, you can query the `dba_index_usage` or `v$object_usage` views when monitoring is enabled.

  Azure Synapse features, such as parallel query processing and in-memory caching of data and results, make it likely that fewer indexes are required for data warehouse applications to achieve excellent performance goals.

- **Clustered tables**: Oracle tables can be organized so that table rows that are frequently accessed together (based on a common value) are physically stored together. This strategy reduces disk I/O when data is retrieved. Oracle also has a hash-cluster option for individual tables, which applies a hash value to the cluster key and physically stores rows with the same hash value together.

  In Azure Synapse, you can achieve a similar result by partitioning and/or using other indexes.

- **Materialized views**: Oracle supports materialized views and recommends one or more of them for large tables with many columns where only a few columns are regularly used in queries. Materialized views are automatically refreshed by the system when data in the base table is updated.

  In 2019, Microsoft announced that Azure Synapse will support materialized views with the same functionality as in Oracle. Materialized views are now a preview feature in Azure Synapse.

- **In-database triggers**: in Oracle, a trigger can be configured to automatically run when a triggering event occurs. Triggering events can be:

  - A DML statement, such as `INSERT`, `UPDATE`, or `DELETE`, runs. If you defined a trigger that fires before an `INSERT` statement on a customer table, the trigger will fire once before a new row is inserted into the customer table.
  - A DDL statement, such as `CREATE` or `ALTER`, runs. This triggering event is often used to record schema changes for auditing purposes.
  - A system event such as startup or shutdown of the Oracle database.
  - A user event such as login or logout.

  Azure Synapse doesn't support Oracle database triggers. However, you can achieve equivalent functionality by using Data Factory, although doing so will require you to refactor the processes that use triggers.

- **Synonyms**: Oracle supports defining synonyms as alternative names for several database object types. Those types include tables, views, sequences, procedures, stored functions, packages, materialized views, Java class schema objects, user-defined objects, or other synonyms.

  Azure Synapse doesn't currently support defining synonyms, although if a synonym in Oracle refers to a table or view, then you can define a view in Azure Synapse to match the alternative name. If a synonym in Oracle refers to a function or stored procedure, then you can replace the synonym in Azure Synapse with another function or stored procedure that calls the target.

- **User-defined types**: Oracle supports user-defined objects that can contain a series of individual fields, each with their own definition and default values. Those objects can then be referenced within a table

definition in the same way as built-in data types like `NUMBER` or `VARCHAR`.

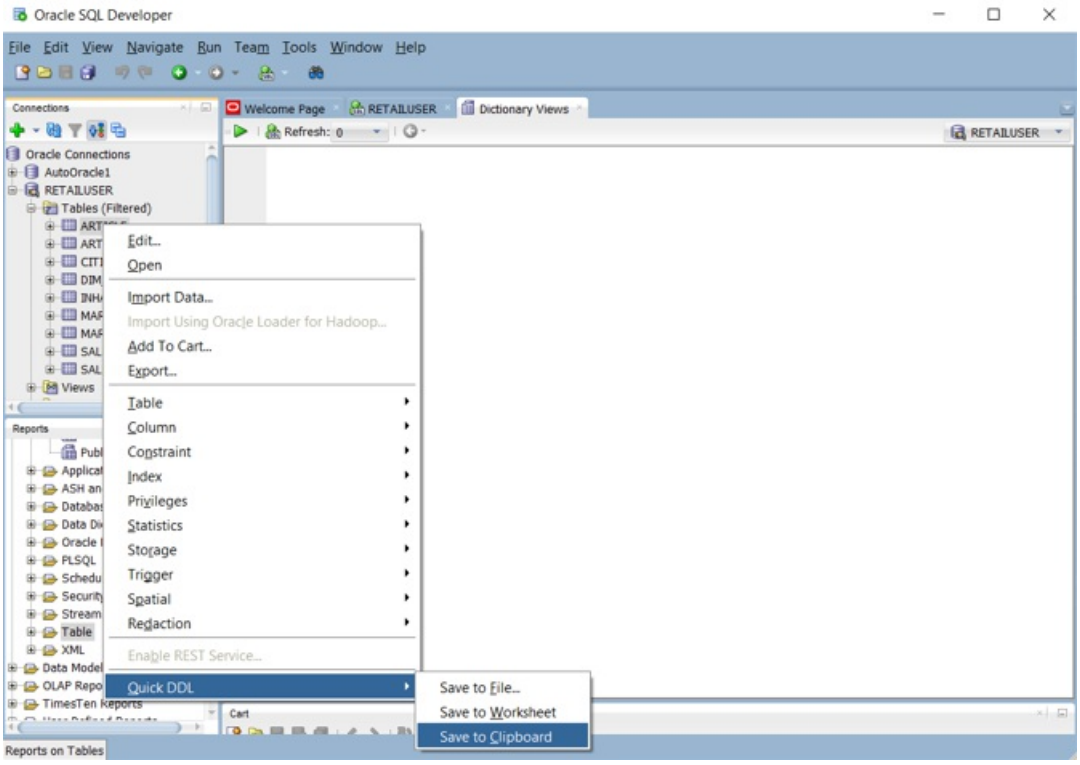
Azure Synapse doesn't currently support user-defined types. If the data you need to migrate includes user-defined data types, either "flatten" them into a conventional table definition, or if they're arrays of data, normalize them in a separate table.

SQL DDL generation

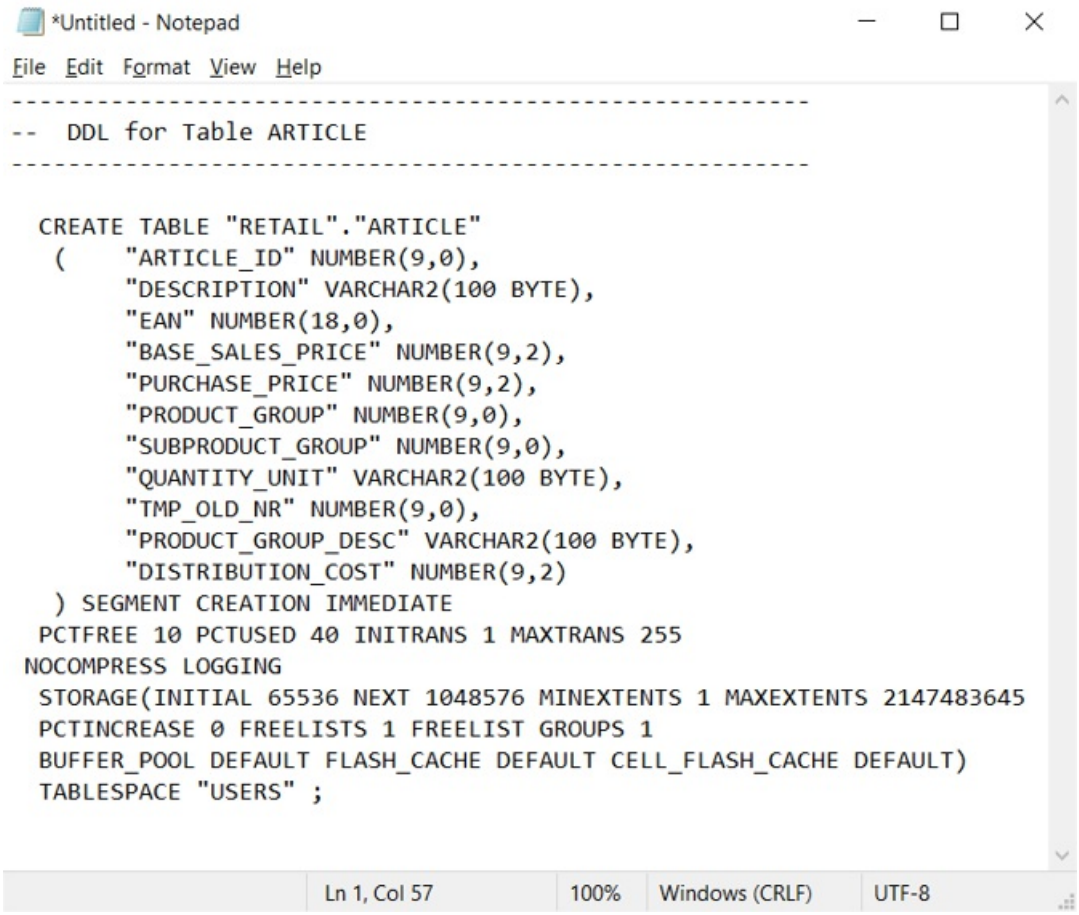
You can edit existing Oracle `CREATE TABLE` and `CREATE VIEW` scripts to achieve equivalent definitions in Azure Synapse. To do so, you might need to use modified data types and remove or modify Oracle-specific clauses, such as `TABLESPACE`.

> **TIP**
> Use existing Oracle metadata to automate the generation of `CREATE TABLE` and `CREATE VIEW` DDL for Azure Synapse.

Within the Oracle environment, system catalog tables specify the current table/view definition. Unlike user-maintained documentation, system catalog information is always complete and in sync with current table definitions. You can access system catalog information by using utilities such as Oracle SQL Developer. Oracle SQL Developer can generate `CREATE TABLE` DDL statements that you can edit to apply to equivalent tables in Azure Synapse, as shown in the next screenshot.



Oracle SQL Developer outputs the following `CREATE TABLE` statement, which contains Oracle-specific clauses that you should remove. Map any unsupported data types before running your modified `CREATE TABLE` statement on Azure Synapse.



Alternatively, you can automatically generate `CREATE TABLE` statements from the information within Oracle catalog tables by using SQL queries, SSMA, or third-party migration tools. This approach is the fastest, most consistent way to generate `CREATE TABLE` statements for many tables.

> **TIP**
> Third-party tools and services can automate data mapping tasks.

Third-party vendors offer tools and services to automate migration, including the mapping of data types. If a third-party ETL tool is already in use in the Oracle environment, use that tool to implement any required data transformations.

SQL DML differences between Oracle and Azure Synapse

The ANSI SQL standard defines the basic syntax for DML commands, such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. Although Oracle and Azure Synapse both support DDL commands, in some cases they implement the same command differently.

> **TIP**
> The standard SQL DML commands `SELECT`, `INSERT`, and `UPDATE` can have additional syntax options in different database environments.

The following sections discuss the Oracle-specific DML commands that need to be considered during a migration to Azure Synapse.

SQL DML syntax differences

There are some SQL DML syntax differences between Oracle SQL and Azure Synapse T-SQL:

- `DUAL` table: Oracle has a system table named `DUAL` that consists of exactly one column named `dummy` and one record with the value `X`. The `DUAL` system table is used when a query requires a table name for syntax reasons, but the table content isn't needed.

  An example Oracle query that uses the `DUAL` table is `SELECT sysdate from dual;`. The Azure Synapse equivalent is `SELECT GETDATE();`. To simplify migration of DML, you could create an equivalent `DUAL` table in Azure Synapse using the following DDL.

  ```
  CREATE TABLE DUAL
  (
    DUMMY VARCHAR(1)
  )
  GO
  INSERT INTO DUAL (DUMMY)
  VALUES ('X')
  GO
  ```

- `NULL` values: a `NULL` value in Oracle is an empty string, represented by a `CHAR` or `VARCHAR` string type of length `0`. In Azure Synapse and most other databases, `NULL` means something else. Be careful when migrating data, or when migrating processes that handle or store data, to ensure that `NULL` values are handled consistently.

- Oracle outer join syntax: although more recent versions of Oracle support ANSI outer join syntax, older Oracle systems use a proprietary syntax for outer joins that uses a plus sign (`+`) within the SQL statement. If you're migrating an older Oracle environment, you might encounter the older syntax. For example:

  ```
  SELECT
      d.deptno, e.job
  FROM
      dept d,
      emp e
  WHERE
      d.deptno = e.deptno (+)
      AND e.job (+) = 'CLERK'
  GROUP BY
      d.deptno, e.job;
  ```

  The equivalent ANSI standard syntax is:

  ```
  SELECT
      d.deptno, e.job
  FROM
      dept d
      LEFT OUTER JOIN emp e ON d.deptno = e.deptno
      and e.job = 'CLERK'
  GROUP BY
      d.deptno,
      e.job
  ORDER BY
      d.deptno, e.job;
  ```

- `DATE` data: in Oracle, the `DATE` data type can store both date and time. Azure Synapse stores date and time in separate `DATE`, `TIME`, and `DATETIME` data types. When you're migrating Oracle `DATE` columns, check whether they store both date and time or just a date. If they only store a date, then map the column to `DATE`, otherwise to `DATETIME`.

- `DATE` arithmetic: Oracle supports subtracting one date from another, for example

`SELECT date '2018-12-31' - date '2018-1201' from dual;`. In Azure Synapse, you can subtract dates by using the `DATEDIFF()` function, for example `SELECT DATEDIFF(day, '2018-12-01', '2018-12-31');`.

Oracle can subtract integers from dates, for example `SELECT hire_date, (hire_date-1) FROM employees;`. In Azure Synapse, you can add or subtract integers from dates by using the `DATEADD()` function.

- Updates via views: in Oracle you can run insert, update, and delete operations against a view to update the underlying table. In Azure Synapse, you run those operations against a base table—not a view. You might have to re-engineer ETL processing if an Oracle table is updated through a view.
- Built-in functions: the following table shows the differences in the syntax and usage of some built-in functions.

ORACLE FUNCTION	DESCRIPTION	SYNAPSE EQUIVALENT
ADD_MONTHS	Add a specified number of months	DATEADD
CAST	Convert one built-in data type into another	CAST
DECODE	Evaluate a list of conditions	CASE expression
EMPTY_BLOB	Create an empty BLOB value	`0x` constant (empty binary string)
EMPTY_CLOB	Create an empty CLOB or NCLOB value	`''` (empty string)
INITCAP	Capitalize the first letter of each word	User-defined function
INSTR	Find position of a substring in a string	CHARINDEX
LAST_DAY	Get the last date of month	EOMONTH
LENGTH	Get string length in characters	LEN
LPAD	Left-pad string to the specified length	Expression using REPLICATE, RIGHT, and LEFT
MOD	Get the remainder of a division of one number by another	`%` operator
MONTHS_BETWEEN	Get the number of months between two dates	DATEDIFF
NVL	Replace `NULL` with expression	ISNULL
SUBSTR	Return a substring from a string	SUBSTRING
TO_CHAR for datetime	Convert datetime to string	CONVERT
TO_DATE	Convert a string to datetime	CONVERT
TRANSLATE	One-to-one single character substitution	Expressions using REPLACE or a user-defined function

ORACLE FUNCTION	DESCRIPTION	SYNAPSE EQUIVALENT
TRIM	Trim leading or trailing characters	LTRIM and RTRIM
TRUNC for datetime	Truncate datetime	Expressions using CONVERT
UNISTR	Convert Unicode code points to characters	Expressions using NCHAR

Functions, stored procedures, and sequences

When migrating a data warehouse from a mature environment like Oracle, you probably need to migrate elements other than simple tables and views. For functions, stored procedures, and sequences, check whether tools within the Azure environment can replace their functionality because it's usually more efficient to use built-in Azure tools than to recode the Oracle functions.

As part of your preparation phase, create an inventory of objects that need to be migrated, define a method for handling them, and allocate appropriate resources in your migration plan.

Microsoft tools like SSMA for Oracle and Azure Database Migration Services, or third-party migration products and services, can automate the migration of functions, stored procedures, and sequences.

> **TIP**
> Third-party products and services can automate migration of non-data elements.

The following sections discuss the migration of functions, stored procedures, and sequences.

Functions

As with most database products, Oracle supports system and user-defined functions within a SQL implementation. When you migrate a legacy database platform to Azure Synapse, you can usually migrate common system functions without change. Some system functions might have a slightly different syntax, but you can automate any required changes.

For Oracle system functions or arbitrary user-defined functions that have no equivalent in Azure Synapse, recode those functions using the target environment language. Oracle user-defined functions are coded in PL/SQL, Java, or C. Azure Synapse uses the Transact-SQL language to implement user-defined functions.

Stored procedures

Most modern database products support storing procedures within the database. Oracle provides the PL/SQL language for this purpose. A stored procedure typically contains both SQL statements and procedural logic, and returns data or a status.

Azure Synapse supports stored procedures using T-SQL, so you'll need to recode any migrated stored procedures in T-SQL.

Sequences

In Oracle, a sequence is a named database object, created using `CREATE SEQUENCE`. A sequence provides unique numeric values via the `CURRVAL` and `NEXTVAL` methods. You can use the generated unique numbers as surrogate key values for primary keys. Azure Synapse doesn't implement `CREATE SEQUENCE`, but you can implement sequences using `IDENTITY` columns or SQL code that generates the next sequence number in a series.

Use EXPLAIN to validate legacy SQL

> **TIP**
> Use real queries from the existing system query logs to find potential migration issues.

Assuming a like-for-like migrated data model in Azure Synapse with the same table and column names, one way to test legacy Oracle SQL for compatibility with Azure Synapse is:

1. Capture some representative SQL statements from the legacy system query history logs.
2. Prefix those queries with the `EXPLAIN` statement.
3. Run the `EXPLAIN` statements in Azure Synapse.

Any incompatible SQL will generate an error, and the error information can be used to determine the scale of the recoding task. This approach doesn't require you to load any data into the Azure environment, you only need to create the relevant tables and views.

Summary

Existing legacy Oracle installations are typically implemented in a way that makes migration to Azure Synapse relatively straightforward. Both environments use SQL for analytical queries on large data volumes, and generally use some form of dimensional data model. These factors make Oracle installations a good candidate for migration to Azure Synapse.

To summarize, our recommendations for minimizing the task of migrating SQL code from Oracle to Azure Synapse are:

- Migrate your existing data model as-is to minimize risk, effort, and migration time, even if a different data model is planned, such as a data vault.
- Understand the differences between the Oracle SQL implementation and the Azure Synapse implementation.
- Use the metadata and query logs from the existing Oracle implementation to assess the impact of changing the environment. Plan an approach to mitigate the differences.
- Automate the migration process to minimize risk, effort, and migration time. You can use Microsoft tools such as Azure Database Migration Services and SSMA.
- Consider using specialist third-party tools and services to streamline the migration.

Next steps

To learn more about Microsoft and third-party tools, see the next article in this series: Tools for Oracle data warehouse migration to Azure Synapse Analytics.

Tools for Oracle data warehouse migration to Azure Synapse Analytics

8/19/2022 • 6 minutes to read • Edit Online

This article is part six of a seven-part series that provides guidance on how to migrate from Oracle to Azure Synapse Analytics. The focus of this article is best practices for Microsoft and third-party tools.

Data warehouse migration tools

By migrating your existing data warehouse to Azure Synapse, you benefit from:

- A globally secure, scalable, low-cost, cloud-native, pay-as-you-use analytical database.
- The rich Microsoft analytical ecosystem that exists on Azure. This ecosystem consists of technologies to help modernize your data warehouse once it's migrated and extend your analytical capabilities to drive new value.

Several tools from both Microsoft and third-party partners can help you migrate your existing data warehouse to Azure Synapse. This article discusses the following types of tools:

- Microsoft data and database migration tools.
- Third-party data warehouse automation tools to automate and document the migration to Azure Synapse.
- Third-party data warehouse migration tools to migrate schema and data to Azure Synapse.
- Third-party tools to bridge the SQL differences between your existing data warehouse DBMS and Azure Synapse.

Microsoft data migration tools

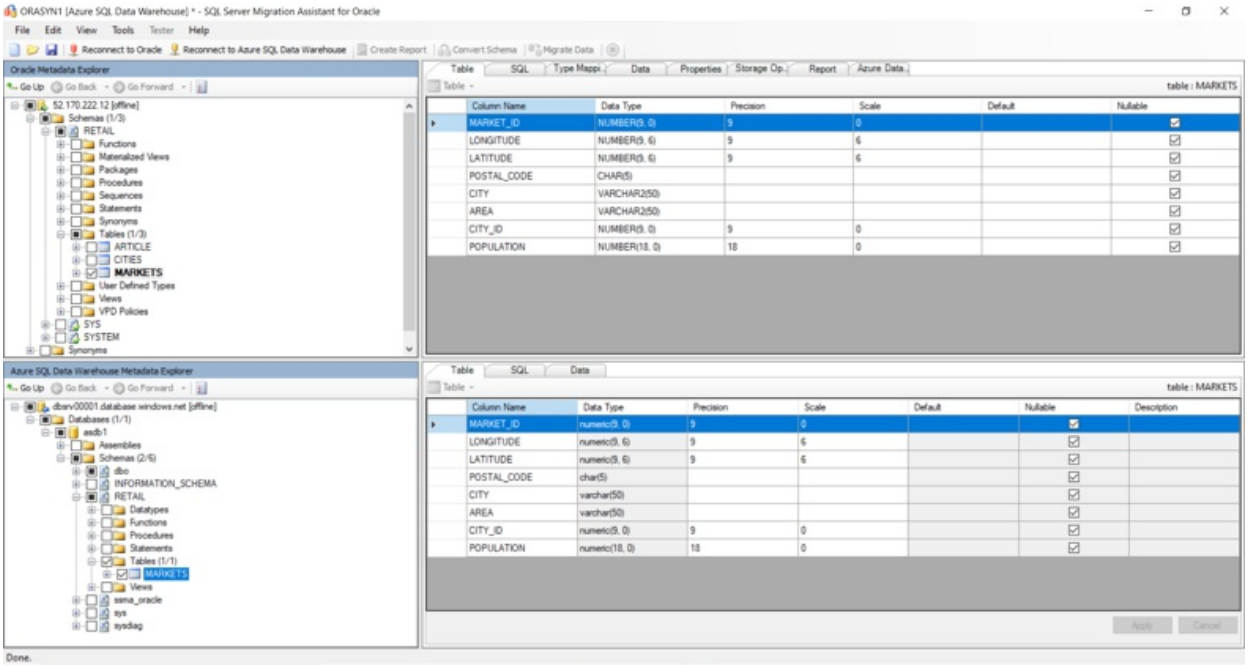
Microsoft offers several tools to help you migrate your existing data warehouse to Azure Synapse, such as:

- SQL Server Migration Assistant (SSMA)
- Azure Data Factory.
- Microsoft services for physical data transfer.
- Microsoft services for data ingestion.

The next sections discuss these tools in more detail.

SQL Server Migration Assistant (SSMA)

SQL Server Migration Assistant (SSMA) for Oracle can automate many parts of the migration process, including in some cases functions and procedural code. SSMA supports Azure Synapse as a target environment.



SSMA for Oracle can help you migrate an Oracle data warehouse or data mart to Azure Synapse. SSMA is designed to automate the process of migrating tables, views, and data from an existing Oracle environment.

Microsoft Azure Data Factory

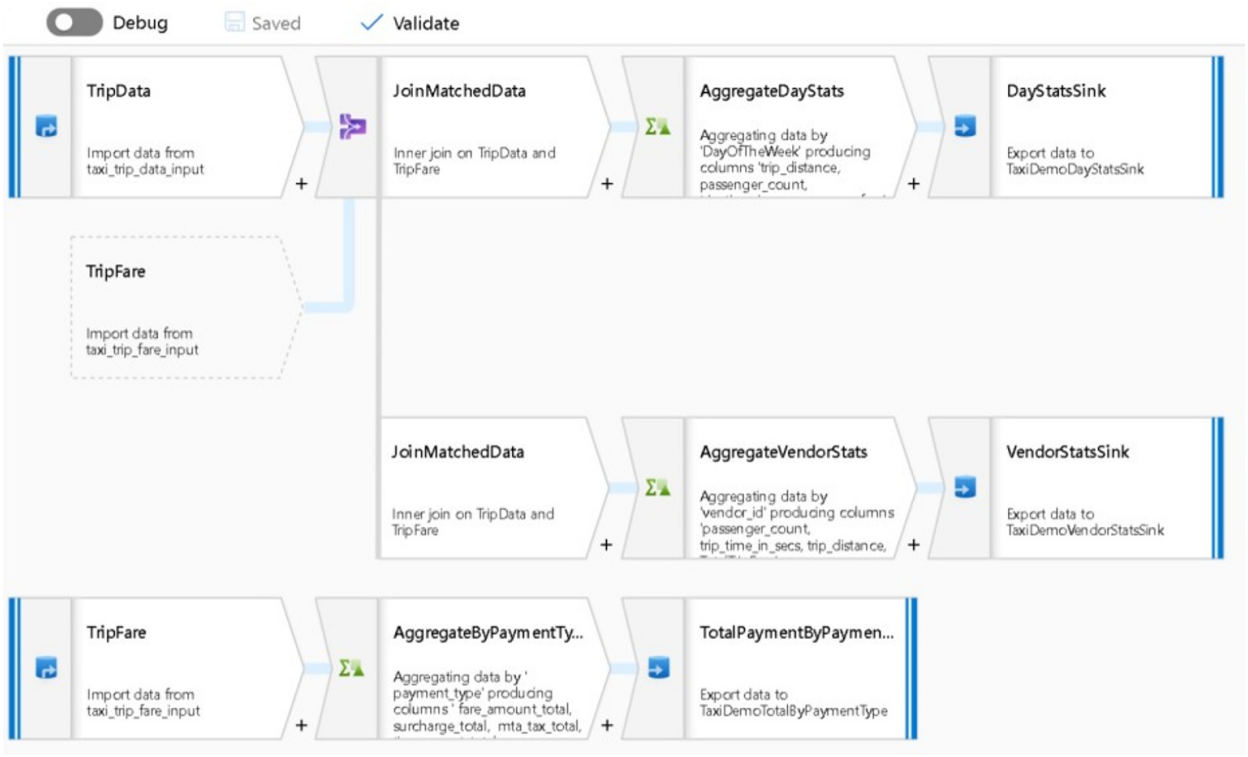
Data Factory is a fully managed, pay-as-you-use, hybrid data integration service for highly scalable ETL and ELT processing. It uses Apache Spark to process and analyze data in parallel and in-memory to maximize throughput.

> **TIP**
> Data Factory allows you to build scalable data integration pipelines code-free.

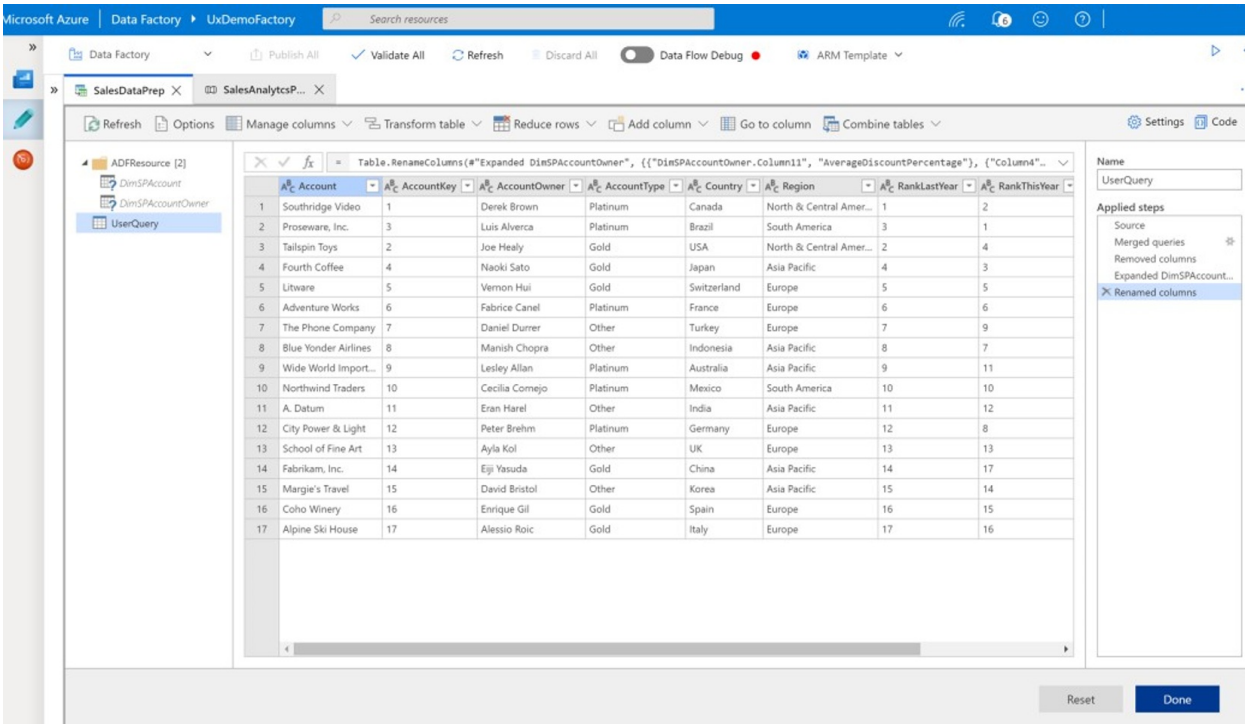
Data Factory connectors support connections to external data sources and databases and include templates for common data integration tasks. A visual front-end, browser-based UI enables non-programmers to create and run pipelines to ingest, transform, and load data. More experienced programmers can incorporate custom code, such as Python programs.

> **TIP**
> Data Factory enables collaborative development between business and IT professionals.

Data Factory is also an orchestration tool and is the best Microsoft tool to automate the end-to-end migration process. Automation reduces the risk, effort, and time to migrate, and makes the migration process easily repeatable. The following diagram shows a mapping data flow in Data Factory.



The next screenshot shows a wrangling data flow in Data Factory.



	Account	AccountKey	AccountOwner	AccountType	Country	Region	RankLastYear	RankThisYear
1	Southridge Video	1	Derek Brown	Platinum	Canada	North & Central Amer...	1	2
2	Proseware, Inc.	3	Luis Alverca	Platinum	Brazil	South America	3	1
3	Tailspin Toys	2	Joe Healy	Gold	USA	North & Central Amer...	2	4
4	Fourth Coffee	4	Naoki Sato	Gold	Japan	Asia Pacific	4	3
5	Litware	5	Vernon Hui	Gold	Switzerland	Europe	5	5
6	Adventure Works	6	Fabrice Canel	Platinum	France	Europe	6	6
7	The Phone Company	7	Daniel Durrer	Other	Turkey	Europe	7	9
8	Blue Yonder Airlines	8	Manish Chopra	Other	Indonesia	Asia Pacific	8	7
9	Wide World Import...	9	Lesley Allan	Platinum	Australia	Asia Pacific	9	11
10	Northwind Traders	10	Cecilia Comejo	Platinum	Mexico	South America	10	10
11	A. Datum	11	Eran Harel	Other	India	Asia Pacific	11	12
12	City Power & Light	12	Peter Brehm	Platinum	Germany	Europe	12	8
13	School of Fine Art	13	Ayla Kol	Other	UK	Europe	13	13
14	Fabrikam, Inc.	14	Eiji Yasuda	Gold	China	Asia Pacific	14	17
15	Margie's Travel	15	David Bristol	Other	Korea	Asia Pacific	15	14
16	Coho Winery	16	Enrique Gil	Gold	Spain	Europe	16	15
17	Alpine Ski House	17	Alessio Roic	Gold	Italy	Europe	17	16

In Data Factory, you can develop simple or comprehensive ETL and ELT processes without coding or maintenance with just a few clicks. ETL/ELT processes ingest, move, prepare, transform, and process your data. You can design and manage scheduling and triggers in Data Factory to build an automated data integration and loading environment. In Data Factory, you can define, manage, and schedule PolyBase bulk data load processes.

> **TIP**
> Data Factory includes tools to help migrate both your data and your entire data warehouse to Azure.

You can use Data Factory to implement and manage a hybrid environment with on-premises, cloud, streaming, and SaaS data in a secure and consistent way. SaaS data might come from applications such as Salesforce.

Wrangling data flows is a new capability in Data Factory. This capability opens up Data Factory to business users who want to visually discover, explore, and prepare data at scale without writing code. Wrangling data flows offers self-service data preparation, similar to Microsoft Excel, Power Query, and Microsoft Power BI dataflows. Business users can prepare and integrate data through a spreadsheet-style UI with drop-down transform options.

Data migration at scale capability helps migration of data at source to Azure SQL target using Data Factory.

Data Factory is the recommended approach for implementing data integration and ETL/ELT processes in the Azure Synapse environment, especially if you want to refactor existing legacy processes.

Microsoft services for physical data transfer

The following sections discuss a range of products and services that Microsoft offers to assist customers with data transfer. This offline data movement will allow end users to reduce the migration downtime significantly.

Azure ExpressRoute

Azure ExpressRoute creates private connections between Azure data centers and infrastructure on your premises or in a collocation environment. ExpressRoute connections don't go over the public internet, and offer more reliability, faster speeds, and lower latencies than typical internet connections. In some cases, you gain significant cost benefits by using ExpressRoute connections to transfer data between on-premises systems and Azure.

AzCopy

AzCopy is a command line utility that copies files to Azure Blob Storage over a standard internet connection. In a warehouse migration project, you can use AzCopy to upload extracted, compressed, delimited text files before loading them into Azure Synapse using PolyBase. AzCopy can upload individual files, file selections, or file folders. If the exported files are in Parquet format, use a native Parquet reader instead.

Azure Data Box

Azure Data Box is a Microsoft service that provides you with a proprietary physical storage device that you can copy migration data onto. You then ship the device to an Azure data center for data upload to cloud storage. This service can be cost-effective for large volumes of data, such as tens or hundreds of terabytes, or where network bandwidth isn't readily available. Azure Data Box is typically used for a large one-off historical data load into Azure Synapse.

Azure Data Box Gateway

Azure Data Box Gateway is a virtualized cloud storage gateway device that resides on your premises and sends your images, media, and other data to Azure. Use Data Box Gateway for one-off migration tasks or ongoing incremental data uploads.

Microsoft services for data ingestion

The following sections discuss the products and services that Microsoft offers to assist customers with data ingestion.

COPY INTO

The COPY INTO statement provides the most flexibility for high-throughput data ingestion into Azure Synapse. For more information about `COPY INTO` capabilities, see COPY (Transact-SQL).

PolyBase

PolyBase is the fastest, most scalable method for bulk data load into Azure Synapse. PolyBase uses the massively parallel processing (MPP) architecture of Azure Synapse for parallel loading of data to achieve the fastest throughput. PolyBase can read data from flat files in Azure Blob Storage, or directly from external data sources and other relational databases via connectors.

> **TIP**
> PolyBase can load data in parallel from Azure Blob Storage into Azure Synapse.

PolyBase can also directly read from files compressed with gzip to reduce the physical volume of data during a load process. PolyBase supports popular data formats such as delimited text, ORC, and Parquet.

> **TIP**
> You can invoke PolyBase from Data Factory as part of a migration pipeline.

PolyBase is tightly integrated with Data Factory to support rapid development of data load ETL/ELT processes. You can schedule data load processes through a visual UI for higher productivity and fewer errors than hand-written code. Microsoft recommends PolyBase for data ingestion into Azure Synapse, especially for high-volume data ingestion.

PolyBase uses `CREATE TABLE AS` or `INSERT...SELECT` statements to load data. `CREATE TABLE AS` minimizes logging to achieve the highest throughput. The most efficient input format for data load is compressed delimited text files. For maximum throughput, split large input files into multiple smaller files and load them in parallel. For fastest loading to a staging table, define the target table as `HEAP` type and use round-robin distribution.

PolyBase has some limitations, it requires data row length to be less than 1 megabyte and doesn't support fixed-width nested formats like JSON and XML.

Microsoft tools for Oracle migrations

SQL Server Migration Assistant (SSMA) for Oracle can help you migrate your legacy on-premises data warehouse platform to Azure Synapse.

Microsoft partners for Oracle migrations

Microsoft partners offer tools, services, and expertise to help you migrate your legacy on-premises data warehouse platform to Azure Synapse.

Next steps

To learn more about implementing modern data warehouses, see the next article in this series: Beyond Oracle migration, implement a modern data warehouse in Microsoft Azure.

Beyond Oracle migration, implement a modern data warehouse in Microsoft Azure

8/19/2022 • 23 minutes to read • Edit Online

This article is part seven of a seven-part series that provides guidance on how to migrate from Oracle to Azure Synapse Analytics. The focus of this article is best practices for implementing modern data warehouses.

Beyond data warehouse migration to Azure

A key reason to migrate your existing data warehouse to Azure Synapse Analytics is to utilize a globally secure, scalable, low-cost, cloud-native, pay-as-you-use analytical database. With Azure Synapse, you can integrate your migrated data warehouse with the complete Microsoft Azure analytical ecosystem to take advantage of other Microsoft technologies and modernize your migrated data warehouse. Those technologies include:

- Azure Data Lake Storage for cost effective data ingestion, staging, cleansing, and transformation. Data Lake Storage can free up the data warehouse capacity occupied by fast-growing staging tables.
- Azure Data Factory for collaborative IT and self-service data integration with connectors to cloud and on-premises data sources and streaming data.
- Common Data Model to share consistent trusted data across multiple technologies, including:
  - Azure Synapse
  - Azure Synapse Spark
  - Azure HDInsight
  - Power BI
  - Adobe Customer Experience Platform
  - Azure IoT
  - Microsoft ISV partners
- Microsoft data science technologies, including:
  - Azure Machine Learning studio
  - Azure Machine Learning
  - Azure Synapse Spark (Spark as a service)
  - Jupyter Notebooks
  - RStudio
  - ML.NET
  - .NET for Apache Spark, which lets data scientists use Azure Synapse data to train machine learning models at scale.
- Azure HDInsight to process large amounts of data, and to join big data with Azure Synapse data by creating a logical data warehouse using PolyBase.
- Azure Event Hubs, Azure Stream Analytics, and Apache Kafka to integrate live streaming data from Azure Synapse.

The growth of big data has led to an acute demand for machine learning to enable custom-built, trained machine learning models for use in Azure Synapse. Machine learning models enable in-database analytics to run at scale in batch, on an event-driven basis and on-demand. The ability to take advantage of in-database analytics in Azure Synapse from multiple BI tools and applications also guarantees consistent predictions and

recommendations.

In addition, you can integrate Azure Synapse with Microsoft partner tools on Azure to shorten time to value.

Let's take a closer look at how you can take advantage of technologies in the Microsoft analytical ecosystem to modernize your data warehouse after you've migrated to Azure Synapse.

Offload data staging and ETL processing to Data Lake Storage and Data Factory

Digital transformation has created a key challenge for enterprises by generating a torrent of new data for capture and analysis. A good example is transaction data created by opening online transactional processing (OLTP) systems to service access from mobile devices. Much of this data finds its way into data warehouses, and OLTP systems are the main source. With customers now driving the transaction rate rather than employees, the volume of data in data warehouse staging tables has been growing rapidly.

With the rapid influx of data into the enterprise, along with new sources of data like Internet of Things (IoT), companies must find ways to scale up data integration ETL processing. One method is to offload ingestion, data cleansing, transformation, and integration to a data lake and process data at scale there, as part of a data warehouse modernization program.

Once you've migrated your data warehouse to Azure Synapse, Microsoft can modernize your ETL processing by ingesting and staging data in Data Lake Storage. You can then clean, transform, and integrate your data at scale using Data Factory before loading it into Azure Synapse in parallel using PolyBase.

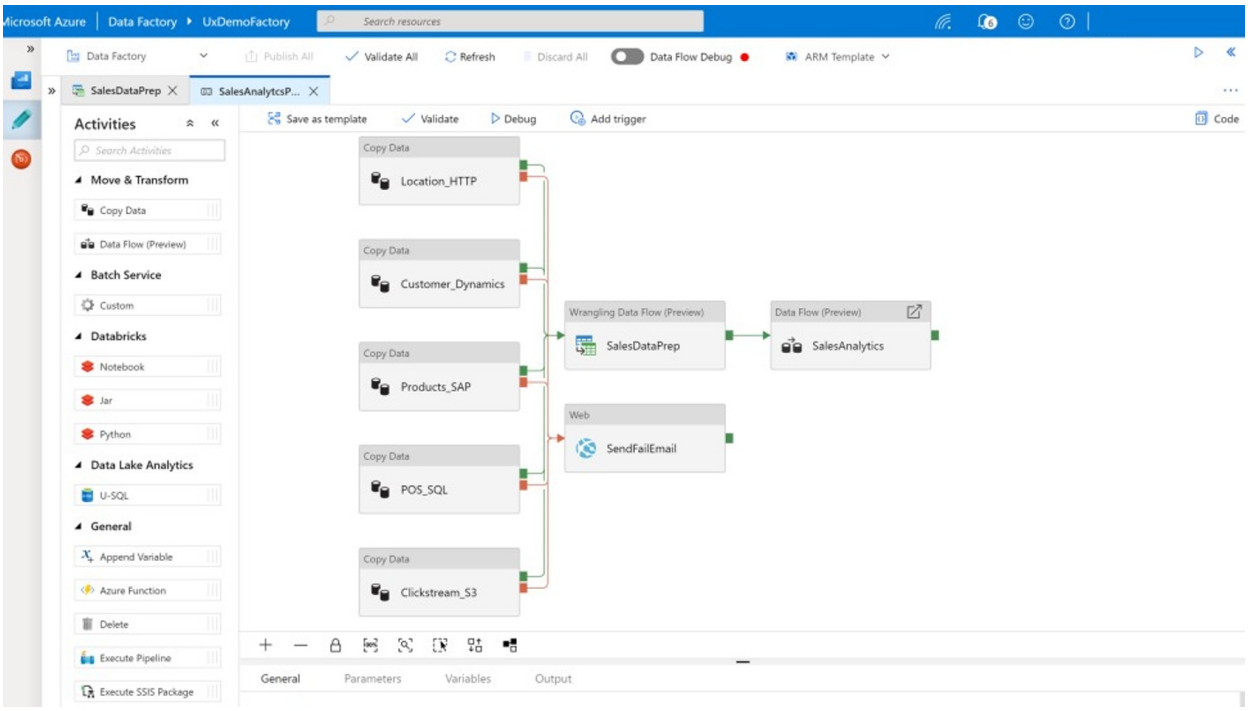
For ELT strategies, consider offloading ELT processing to Data Lake Storage to easily scale as your data volume or frequency grows.

Microsoft Azure Data Factory

Azure Data Factory is a pay-as-you-use, hybrid data integration service for highly scalable ETL and ELT processing. Data Factory provides a web-based UI to build data integration pipelines with no code. With Data Factory, you can:

- Build scalable data integration pipelines code-free.
- Easily acquire data at scale.
- Pay only for what you use.
- Connect to on-premises, cloud, and SaaS-based data sources.
- Ingest, move, clean, transform, integrate, and analyze cloud and on-premises data at scale.
- Seamlessly author, monitor, and manage pipelines that span data stores both on-premises and in the cloud.
- Enable pay-as-you-go scale-out in alignment with customer growth.

You can use these features without writing any code, or you can add custom code to Data Factory pipelines. The following screenshot shows an example Data Factory pipeline.



TIP

Data Factory lets you to build scalable data integration pipelines without code.

Implement Data Factory pipeline development from any of several places, including:

- Microsoft Azure portal.
- Microsoft Azure PowerShell.
- Programmatically from .NET and Python using a multi-language SDK.
- Azure Resource Manager (ARM) templates.
- REST APIs.

TIP

Data Factory can connect to on-premises, cloud, and SaaS data.

Developers and data scientists who prefer to write code can easily author Data Factory pipelines in Java, Python, and .NET using the software development kits (SDKs) available for those programming languages. Data Factory pipelines can be hybrid data pipelines because they can connect, ingest, clean, transform, and analyze data in on-premises data centers, Microsoft Azure, other clouds, and SaaS offerings.

After you develop Data Factory pipelines to integrate and analyze data, you can deploy those pipelines globally and schedule them to run in batch, invoke them on demand as a service, or run them in real-time on an event-driven basis. A Data Factory pipeline can also run on one or more execution engines and monitor execution to ensure performance and to track errors.

TIP

In Azure Data Factory, pipelines control the integration and analysis of data. Data Factory is enterprise-class data integration software aimed at IT professionals and has data wrangling capability for business users.

Use cases

Data Factory supports multiple use cases, such as:

- Prepare, integrate, and enrich data from cloud and on-premises data sources to populate your migrated data warehouse and data marts on Microsoft Azure Synapse.
- Prepare, integrate, and enrich data from cloud and on-premises data sources to produce training data for use in machine learning model development and in retraining analytical models.
- Orchestrate data preparation and analytics to create predictive and prescriptive analytical pipelines for processing and analyzing data in batch, such as sentiment analytics. Either act on the results of the analysis or populate your data warehouse with the results.
- Prepare, integrate, and enrich data for data-driven business applications running on the Azure cloud on top of operational data stores such as Azure Cosmos DB.

> **TIP**
> Build training data sets in data science to develop machine learning models.

Data sources

Data Factory lets you use connectors from both cloud and on-premises data sources. Agent software, known as a *self-hosted integration runtime*, securely accesses on-premises data sources and supports secure, scalable data transfer.

Transform data using Azure Data Factory

Within a Data Factory pipeline, you can ingest, clean, transform, integrate, and analyze any type of data from these sources. Data can be structured, semi-structured like JSON or Avro, or unstructured.

Without writing any code, professional ETL developers can use Data Factory mapping data flows to filter, split, join several types, lookup, pivot, unpivot, sort, union, and aggregate data. In addition, Data Factory supports surrogate keys, multiple write processing options like insert, upsert, update, table recreation, and table truncation, and several types of target data stores—also known as sinks. ETL developers can also create aggregations, including time-series aggregations that require a window to be placed on data columns.

> **TIP**
> Professional ETL developers can use Data Factory mapping data flows to clean, transform, and integrate data without the need to write code.

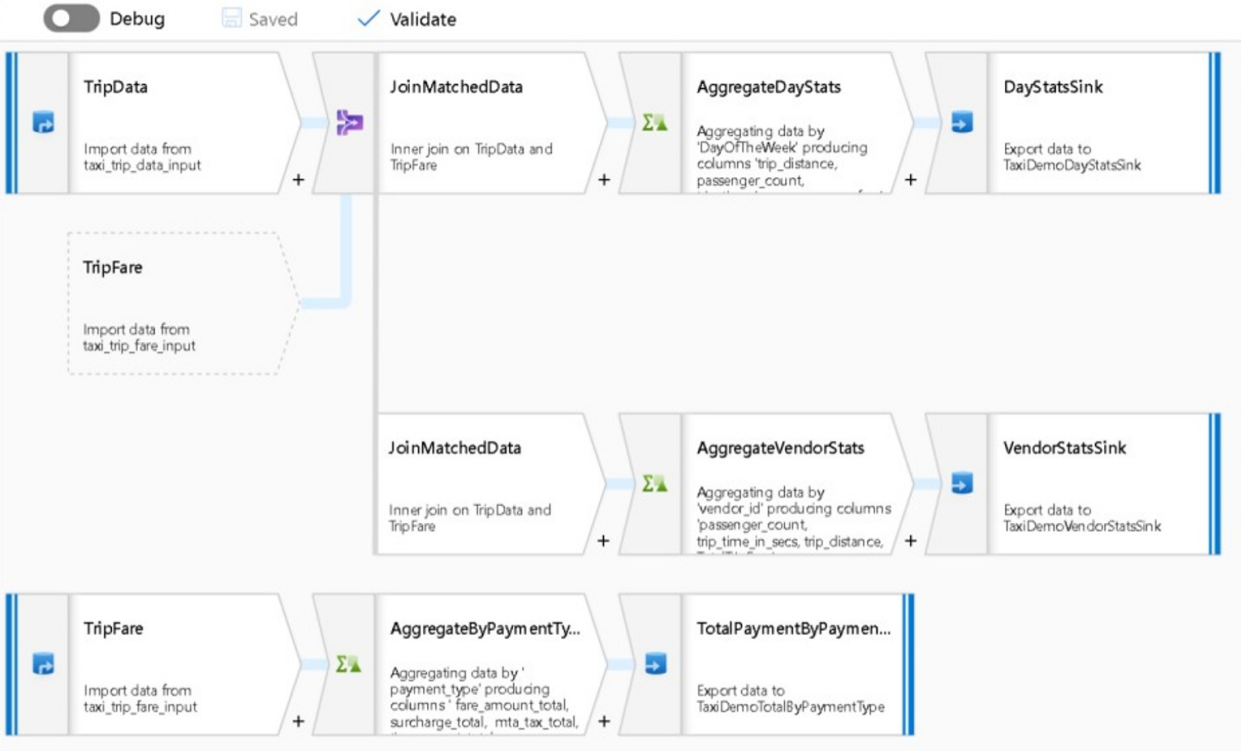
You can run mapping data flows that transform data as activities in a Data Factory pipeline, and if necessary, you can include multiple mapping data flows in a single pipeline. In this way, you can manage the complexity by breaking up challenging data transformation and integration tasks into smaller mapping dataflows that can be combined. And, you can add custom code when needed. In addition to this functionality, Data Factory mapping data flows include the ability to:

- Define expressions to clean and transform data, compute aggregations, and enrich data. For example, these expressions can perform feature engineering on a date field to break it into multiple fields to create training data during machine learning model development. You can construct expressions from a rich set of functions that include mathematical, temporal, split, merge, string concatenation, conditions, pattern match, replace, and many other functions.
- Automatically handle schema drift so that data transformation pipelines can avoid being impacted by schema changes in data sources. This ability is especially important for streaming IoT data, where schema changes can happen without notice if devices are upgraded or when readings are missed by gateway devices collecting IoT data.

- Partition data to enable transformations to run in parallel at scale.
- Inspect streaming data to view the metadata of a stream you're transforming.

> **TIP**
> Data Factory supports the ability to automatically detect and manage schema changes in inbound data, such as in streaming data.

The following screenshot shows an example Data Factory mapping data flow.



Data engineers can profile data quality and view the results of individual data transforms by enabling debug capability during development.

> **TIP**
> Data Factory can also partition data to enable ETL processing to run at scale.

If necessary, you can extend Data Factory transformational and analytical functionality by adding a linked service that contains your code into a pipeline. For example, an Azure Synapse Spark pool notebook might contain Python code that uses a trained model to score the data integrated by a mapping data flow.

You can store integrated data and any results from analytics within a Data Factory pipeline in one or more data stores, such as Data Lake Storage, Azure Synapse, or Hive tables in HDInsight. You can also invoke other activities to act on insights produced by a Data Factory analytical pipeline.

> **TIP**
> Data Factory pipelines are extensible because Data Factory lets you write your own code and run it as part of a pipeline.

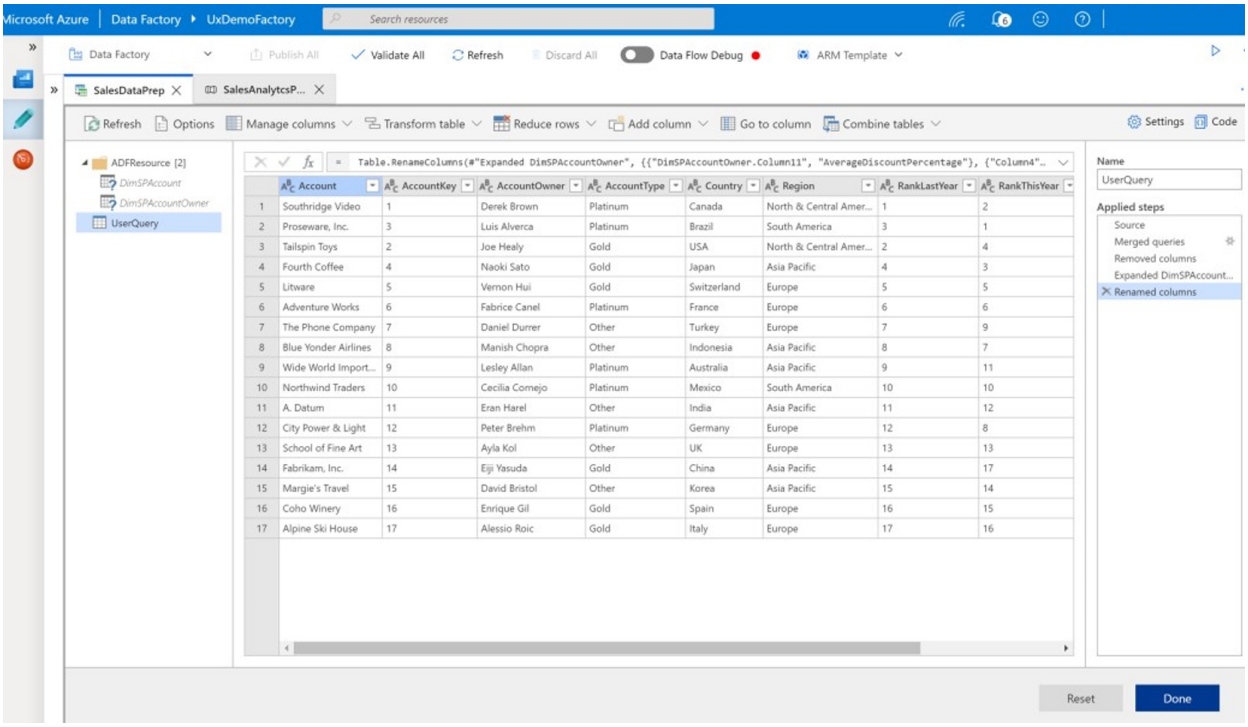
Utilize Spark to scale data integration

At run time, Data Factory internally uses Azure Synapse Spark pools, which are Microsoft's Spark as a service offering, to clean and integrate data in the Azure cloud. You can clean, integrate, and analyze high-volume, high-

velocity data, such as click-stream data, at scale. Microsoft's intention is to also run Data Factory pipelines on other Spark distributions. In addition to running ETL jobs on Spark, Data Factory can invoke Pig scripts and Hive queries to access and transform data stored in HDInsight.

Link self-service data prep and Data Factory ETL processing using wrangling data flows

Data wrangling lets business users, also known as citizen data integrators and data engineers, make use of the platform to visually discover, explore, and prepare data at scale without writing code. This Data Factory capability is easy to use and is similar to Microsoft Excel Power Query or Microsoft Power BI dataflows, where self-service business users use a spreadsheet-style UI with drop-down transforms to prepare and integrate data. The following screenshot shows an example Data Factory wrangling data flow.



Unlike Excel and Power BI, Data Factory wrangling data flows use Power Query to generate M code and then translate it into a massively parallel in-memory Spark job for cloud-scale execution. The combination of mapping data flows and wrangling data flows in Data Factory lets professional ETL developers and business users collaborate to prepare, integrate, and analyze data for a common business purpose. The preceding Data Factory mapping data flows diagram shows how both Data Factory and Azure Synapse Spark pool notebooks can be combined in the same Data Factory pipeline. The combination of mapping and wrangling data flows in Data Factory helps IT and business users stay aware of what data flows each has created, and supports data flow reuse to minimize reinvention and maximize productivity and consistency.

> **TIP**
>
> Data Factory supports both wrangling data flows and mapping data flows, so business users and IT users can integrate data collaboratively on a common platform.

Link data and analytics in analytical pipelines

In addition to cleaning and transforming data, Data Factory can combine data integration and analytics in the same pipeline. You can use Data Factory to create both data integration and analytical pipelines, the latter being an extension of the former. You can drop an analytical model into a pipeline to create an analytical pipeline that generates clean, integrated data for predictions or recommendations. Then, you can act on the predictions or recommendations immediately, or store them in your data warehouse to provide new insights and recommendations that can be viewed in BI tools.

To batch score your data, you can develop an analytical model that you invoke as a service within a Data Factory pipeline. You can develop analytical models code-free with Azure Machine Learning studio, or with the Azure

Machine Learning SDK using Azure Synapse Spark pool notebooks or R in RStudio. When you run Spark machine learning pipelines on Azure Synapse Spark pool notebooks, analysis happens at scale.

You can store integrated data and any Data Factory analytical pipeline results in one or more data stores, such as Data Lake Storage, Azure Synapse, or Hive tables in HDInsight. You can also invoke other activities to act on insights produced by a Data Factory analytical pipeline.

Use a lake database to share consistent trusted data

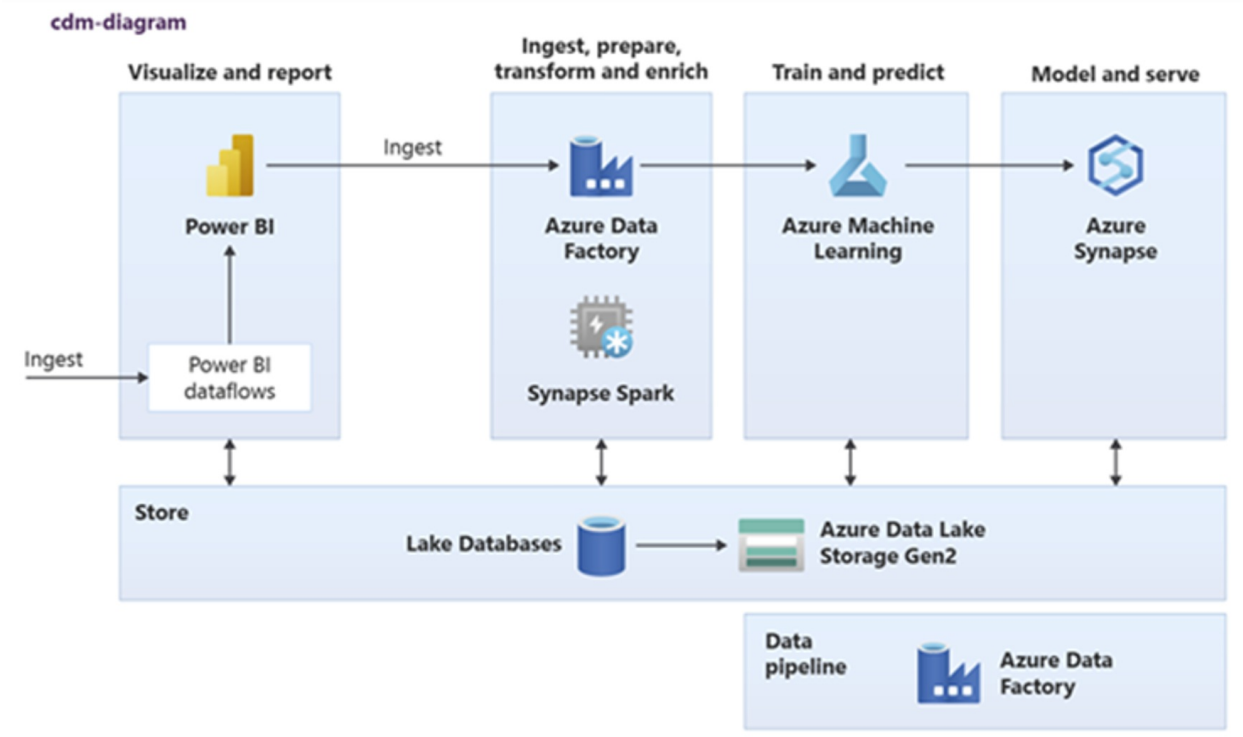
A key objective of any data integration setup is the ability to integrate data once and reuse it everywhere, not just in a data warehouse. For example, you might want to use integrated data in data science. Reuse avoids reinvention and ensures consistent, commonly understood data that everyone can trust.

Common Data Model describes core data entities that can be shared and reused across the enterprise. To achieve reuse, Common Data Model establishes a set of common data names and definitions that describe logical data entities. Examples of common data names include Customer, Account, Product, Supplier, Orders, Payments, and Returns. IT and business professionals can use data integration software to create and store common data assets to maximize their reuse and drive consistency everywhere.

Azure Synapse provides industry-specific database templates to help standardize data in the lake. Lake database templates provide schemas for predefined business areas, enabling data to be loaded into a lake database in a structured way. The power comes when you use data integration software to create lake database common data assets, resulting in self-describing trusted data that can be consumed by applications and analytical systems. You can create common data assets in Data Lake Storage by using Data Factory.

> **TIP**
> Data Lake Storage is shared storage that underpins Microsoft Azure Synapse, Azure Machine Learning, Azure Synapse Spark, and HDInsight.

Power BI, Azure Synapse Spark, Azure Synapse, and Azure Machine Learning can consume common data assets. The following diagram shows how a lake database can be used in Azure Synapse.



> **TIP**
> Integrate data to create lake database logical entities in shared storage to maximize the reuse of common data assets.

Integration with Microsoft data science technologies on Azure

Another key objective when modernizing a data warehouse is to produce insights for competitive advantage. You can produce insights by integrating your migrated data warehouse with Microsoft and third-party data science technologies in Azure. The following sections describe machine learning and data science technologies offered by Microsoft to see how they can be used with Azure Synapse in a modern data warehouse environment.

Microsoft technologies for data science on Azure

Microsoft offers a range of technologies that support advance analysis. With these technologies, you can build predictive analytical models using machine learning or analyze unstructured data using deep learning. The technologies include:

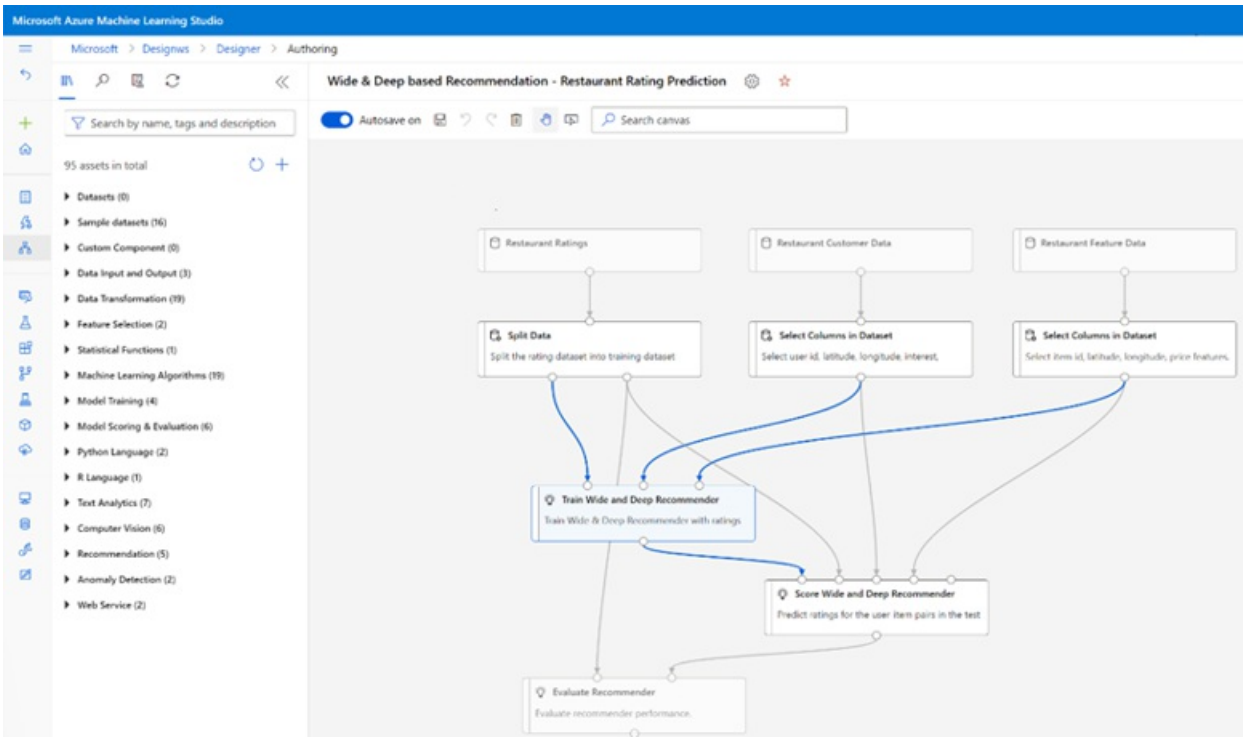
- Azure Machine Learning studio
- Azure Machine Learning
- Azure Synapse Spark pool notebooks
- ML.NET (API, CLI, or ML.NET Model Builder for Visual Studio)
- .NET for Apache Spark

Data scientists can use RStudio (R) and Jupyter Notebooks (Python) to develop analytical models, or they can use frameworks such as Keras or TensorFlow.

> **TIP**
> Develop machine learning models using a no/low-code approach or by using programming languages like Python, R, and .NET.

Azure Machine Learning studio

Azure Machine Learning studio is a fully managed cloud service that lets you build, deploy, and share predictive analytics using a drag-and-drop, web-based UI. The following screenshot shows the Azure Machine Learning studio UI.



Azure Machine Learning

Azure Machine Learning provides an SDK and services for Python that supports can help you quickly prepare data and also train and deploy machine learning models. You can use Azure Machine Learning in Azure notebooks using Jupyter Notebook, with open-source frameworks, such as PyTorch, TensorFlow, scikit-learn, or Spark MLlib—the machine learning library for Spark. Azure Machine Learning provides an AutoML capability that automatically tests multiple algorithms to identify the most accurate algorithms to expedite model development.

> **TIP**
> Azure Machine Learning provides an SDK for developing machine learning models using several open-source frameworks.

You can also use Azure Machine Learning to build machine learning pipelines that manage end-to-end workflow, programmatically scale in the cloud, and deploy models both to the cloud and the edge. Azure Machine Learning contains workspaces, which are logical spaces that you can programmatically or manually create in the Azure portal. These workspaces keep compute targets, experiments, data stores, trained machine learning models, Docker images, and deployed services all in one place to enable teams to work together. You can use Azure Machine Learning in Visual Studio with the Visual Studio for AI extension.

> **TIP**
> Organize and manage related data stores, experiments, trained models, Docker images, and deployed services in workspaces.

Azure Synapse Spark pool notebooks

An Azure Synapse Spark pool notebook is an Azure-optimized Apache Spark service. With Azure Synapse Spark pool notebooks:

- Data engineers can build and run scalable data preparation jobs using Data Factory.
- Data scientists can build and run machine learning models at scale using notebooks written in languages such as Scala, R, Python, Java, and SQL to visualize results.

> **TIP**
> Azure Synapse Spark is a dynamically scalable Spark as a service offering from Microsoft, Spark offers scalable execution of data preparation, model development, and deployed model execution.

Jobs running in Azure Synapse Spark pool notebooks can retrieve, process, and analyze data at scale from Azure Blob Storage, Data Lake Storage, Azure Synapse, HDInsight, and streaming data services such as Apache Kafka.

> **TIP**
> Azure Synapse Spark can access data in a range of Microsoft analytical ecosystem data stores on Azure.

Azure Synapse Spark pool notebooks support autoscaling and auto-termination to reduce total cost of ownership (TCO). Data scientists can use the MLflow open-source framework to manage the machine learning lifecycle.

ML.NET

ML.NET is an open-source, cross-platform machine learning framework for Windows, Linux, macOS. Microsoft created ML.NET so that .NET developers can use existing tools, such as ML.NET Model Builder for Visual Studio, to develop custom machine learning models and integrate them into their .NET applications.

> **TIP**
> Microsoft has extended its machine learning capability to .NET developers.

.NET for Apache Spark

.NET for Apache Spark extends Spark support beyond R, Scala, Python, and Java to .NET and aims to make Spark accessible to .NET developers across all Spark APIs. While .NET for Apache Spark is currently only available on Apache Spark in HDInsight, Microsoft intends to make .NET for Apache Spark available on Azure Synapse Spark pool notebooks.

Use Azure Synapse Analytics with your data warehouse

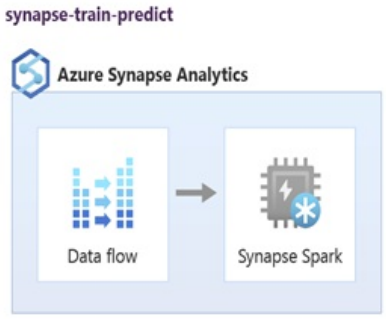
To combine machine learning models with Azure Synapse, you can:

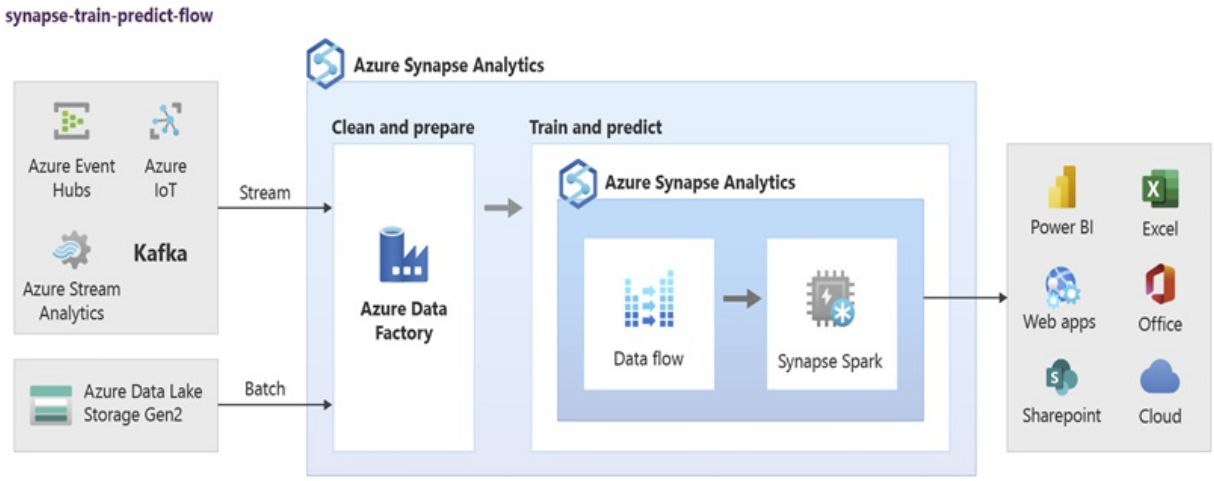
- Use machine learning models in batch or in real-time on streaming data to produce new insights, and add those insights to what you already know in Azure Synapse.
- Use the data in Azure Synapse to develop and train new predictive models for deployment elsewhere, such as in other applications.
- Deploy machine learning models, including models trained elsewhere, in Azure Synapse to analyze data in your data warehouse and drive new business value.

> **TIP**
> Train, test, evaluate, and run machine learning models at scale on Azure Synapse Spark pool notebooks by using data in Azure Synapse.

Data scientists can use RStudio, Jupyter Notebooks, and Azure Synapse Spark pool notebooks together with Azure Machine Learning to develop machine learning models that run at scale on Azure Synapse Spark pool notebooks using data in Azure Synapse. For example, data scientists could create an unsupervised model to segment customers to drive different marketing campaigns. Use supervised machine learning to train a model to predict a specific outcome, such as to predict a customer's propensity to churn, or to recommend the next

best offer for a customer to try to increase their value. The following diagram shows how Azure Synapse can be used for Azure Machine Learning.





In another scenario, you can ingest social network or review website data into Data Lake Storage, then prepare and analyze the data at scale on an Azure Synapse Spark pool notebook using natural language processing to score customer sentiment about your products or brand. You can then add those scores to your data warehouse. By using big data analytics to understand the effect of negative sentiment on product sales, you add to what you already know in your data warehouse.

> **TIP**
> Produce new insights using machine learning on Azure in batch or in real-time and add to what you know in your data warehouse.

Integrate live streaming data into Azure Synapse Analytics

When analyzing data in a modern data warehouse, you must be able to analyze streaming data in real-time and join it with historical data in your data warehouse. An example is combining IoT data with product or asset data.

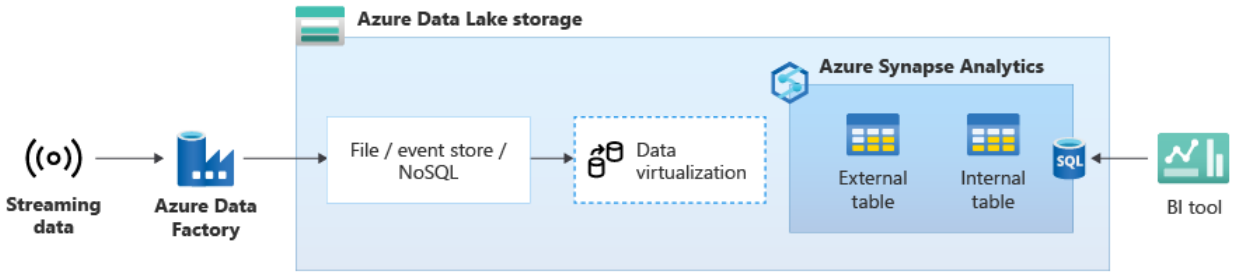
> **TIP**
> Integrate your data warehouse with streaming data from IoT devices or clickstreams.

Once you've successfully migrated your data warehouse to Azure Synapse, you can introduce live streaming data integration as part of a data warehouse modernization exercise by taking advantage of the extra functionality in Azure Synapse. To do so, ingest streaming data via Event Hubs, other technologies like Apache Kafka, or potentially your existing ETL tool if it supports the streaming data sources. Store the data in Data Lake Storage. Then, create an external table in Azure Synapse using PolyBase and point it at the data being streamed into Data Lake Storage so that your data warehouse now contains new tables that provide access to the real-time streaming data. Query the external table as if the data was in the data warehouse by using standard T-SQL

from any BI tool that has access to Azure Synapse. You can also join the streaming data to other tables with historical data to create views that join live streaming data to historical data to make it easier for business users to access the data.

> **TIP**
> Ingest streaming data into Data Lake Storage from Event Hubs or Apache Kafka, and access the data from Azure Synapse using PolyBase external tables.

In the following diagram, a real-time data warehouse on Azure Synapse is integrated with streaming data in Data Lake Storage.



Create a logical data warehouse using PolyBase

With PolyBase, you can create a logical data warehouse to simplify user access to multiple analytical data stores. Many companies have adopted "workload optimized" analytical data stores over the last several years in addition to their data warehouses. The analytical platforms on Azure include:

- Data Lake Storage with Azure Synapse Spark pool notebook (Spark as a service), for big data analytics.
- HDInsight (Hadoop as a service), also for big data analytics.
- NoSQL Graph databases for graph analysis, which could be done in Azure Cosmos DB.
- Event Hubs and Stream Analytics, for real-time analysis of data in motion.

You might have non-Microsoft equivalents of these platforms, or a master data management (MDM) system that needs to be accessed for consistent trusted data on customers, suppliers, products, assets, and more.

> **TIP**
> PolyBase simplifies access to multiple underlying analytical data stores on Azure for ease of access by business users.

Those analytical platforms emerged because of the explosion of new data sources inside and outside the enterprise and the demand by business users to capture and analyze the new data. The new data sources include:

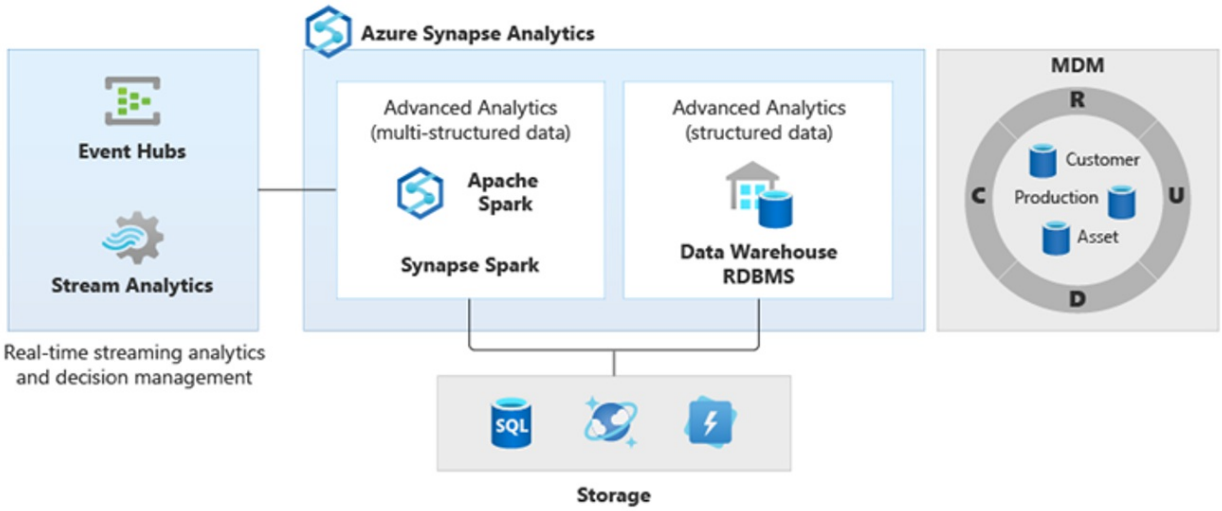
- Machine generated data, such as IoT sensor data and clickstream data.
- Human generated data, such as social network data, review web site data, customer inbound email, images, and video.
- Other external data, such as open government data and weather data.

This new data goes beyond the structured transaction data and main data sources that typically feed data warehouses and often includes:

- Semi-structured data like JSON, XML, or Avro.

- Unstructured data like text, voice, image, or video, which is more complex to process and analyze.
- High volume data, high velocity data, or both.

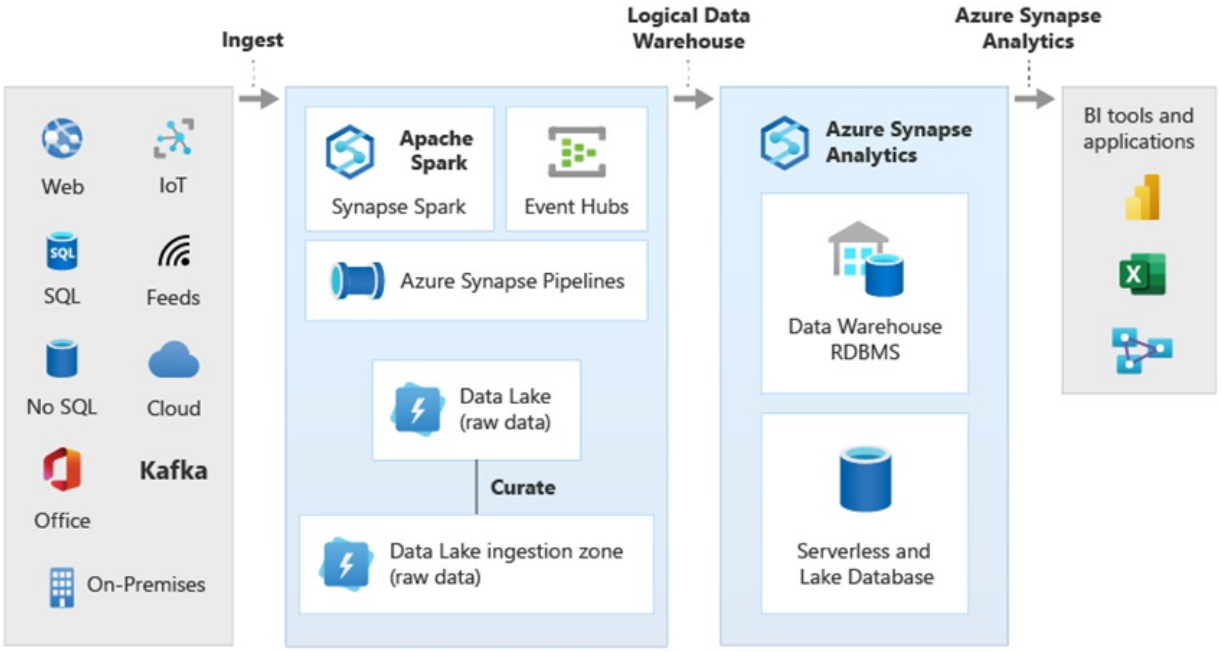
As a result, new more complex kinds of analysis have emerged, such as natural language processing, graph analysis, deep learning, streaming analytics, or complex analysis of large volumes of structured data. These kinds of analysis typically don't happen in a data warehouse, so it's not surprising to see different analytical platforms for different types of analytical workloads, as shown in the following diagram.



> **TIP**
> The ability to make data in multiple analytical data stores look like it's all in one system and join it to Azure Synapse is known as a logical data warehouse architecture.

Because these platforms produce new insights, it's normal to see a requirement to combine the new insights with what you already know in Azure Synapse, which is what PolyBase makes possible.

By using PolyBase data virtualization inside Azure Synapse, you can implement a logical data warehouse where data in Azure Synapse is joined to data in other Azure and on-premises analytical data stores like HDInsight, Azure Cosmos DB, or streaming data flowing into Data Lake Storage from Stream Analytics or Event Hubs. This approach lowers the complexity for users, who access external tables in Azure Synapse and don't need to know that the data they're accessing is stored in multiple underlying analytical systems. The following diagram shows a complex data warehouse structure accessed through comparatively simpler yet still powerful UI methods.



The diagram shows how other technologies in the Microsoft analytical ecosystem can be combined with the capability of the logical data warehouse architecture in Azure Synapse. For example, you can ingest data into Data Lake Storage and curate the data using Data Factory to create trusted data products that represent Microsoft lake database logical data entities. This trusted, commonly understood data can then be consumed and reused in different analytical environments such as Azure Synapse, Azure Synapse Spark pool notebooks, or Azure Cosmos DB. All insights produced in these environments are accessible via a logical data warehouse data virtualization layer made possible by PolyBase.

> **TIP**
> A logical data warehouse architecture simplifies business user access to data and adds new value to what you already know in your data warehouse.

Conclusions

After you migrate your data warehouse to Azure Synapse, you can take advantage of other technologies in the Microsoft analytical ecosystem. By doing so, you not only modernize your data warehouse, but bring insights produced in other Azure analytical data stores into an integrated analytical architecture.

You can broaden your ETL processing to ingest data of any type into Data Lake Storage, and then prepare and integrate the data at scale using Data Factory to produce trusted, commonly understood data assets. Those assets can be consumed by your data warehouse and accessed by data scientists and other applications. You can build real-time and batch oriented analytical pipelines and create machine learning models to run in batch, in real-time on streaming data, and on-demand as a service.

You can use PolyBase or `COPY INTO` to go beyond your data warehouse to simplify access to insights from multiple underlying analytical platforms on Azure. To do so, create holistic integrated views in a logical data warehouse that support access to streaming, big data, and traditional data warehouse insights from BI tools and applications.

By migrating your data warehouse to Azure Synapse, you can take advantage of the rich Microsoft analytical ecosystem running on Azure to drive new value in your business.

Next steps

To learn about migrating to a dedicated SQL pool, see Migrate a data warehouse to a dedicated SQL pool in

Azure Synapse Analytics.

Azure Synapse Analytics security white paper: Introduction

8/19/2022 • 5 minutes to read • Edit Online

Summary: Azure Synapse Analytics is a Microsoft limitless analytics platform that integrates enterprise data warehousing and big data processing into a single managed environment with no system integration required. Azure Synapse provides the end-to-end tools for your analytic life cycle with:

- Pipelines for data integration.
- Apache Spark pool for big data processing.
- Data Explorer for log and time series analytics.
- Serverless SQL pool for data exploration over Azure Data Lake.
- Dedicated SQL pool (formerly SQL DW) for enterprise data warehousing.
- Deep integration with Power BI, Azure Cosmos DB, and Azure Machine Learning.

Azure Synapse data security and privacy are non-negotiable. The purpose of this white paper is to provide a comprehensive overview of Azure Synapse security features, which are enterprise-grade and industry-leading. The white paper comprises a series of articles that cover the following five layers of security:

- Data protection
- Access control
- Authentication
- Network security
- Threat protection

This white paper targets all enterprise security stakeholders. They include security administrators, network administrations, Azure administrators, workspace administrators, and database administrators.

Writers: Vengatesh Parasuraman, Fretz Nuson, Ron Dunn, Khendr'a Reid, John Hoang, Nithesh Krishnappa, Mykola Kovalenko, Brad Schacht, Pedro Martinez, Mark Pryce-Maher, and Arshad Ali.

Technical Reviewers: Nandita Valsan, Rony Thomas, Abhishek Narain, Daniel Crawford, and Tammy Richter Jones.

Applies to: Azure Synapse Analytics, dedicated SQL pool (formerly SQL DW), serverless SQL pool, and Apache Spark pool.

> **IMPORTANT**
> This white paper does not apply to Azure SQL Database, Azure SQL Managed Instance, Azure Machine Learning, or Azure Databricks.

Introduction

Frequent headlines of data breaches, malware infections, and malicious code injection are among an extensive list of security concerns for companies looking to cloud modernization. The enterprise customer requires a cloud provider or service solution that can address their concerns as they can't afford to get it wrong.

Some common security questions include:

- How can I control who can see what data?
- What are the options for verifying a user's identity?
- How is my data protected?
- What network security technology can I use to protect the integrity, confidentiality, and access of my networks and data?
- What are the tools that detect and notify me of threats?

The purpose of this white paper is to provide answers to these common security questions, and many others.

Component architecture

Azure Synapse is a Platform-as-a-service (PaaS) analytics service that brings together multiple independent components such as dedicated SQL pools, serverless SQL pools, Apache Spark pools, and data integration pipelines. These components are designed to work together to provide a seamless analytical platform experience.

Dedicated SQL pools are provisioned clusters that provide enterprise data warehousing capabilities for SQL workloads. Data is ingested into managed storage powered by Azure Storage, which is also a PaaS service. Compute is isolated from storage enabling customers to scale compute independently of their data. Dedicated SQL pools also provide the ability to query data files directly over customer-managed Azure Storage accounts by using external tables.

Serverless SQL pools are on-demand clusters that provide a SQL interface to query and analyze data directly over customer-managed Azure Storage accounts. Since they're serverless, there's no managed storage, and the compute nodes scale automatically in response to the query workload.

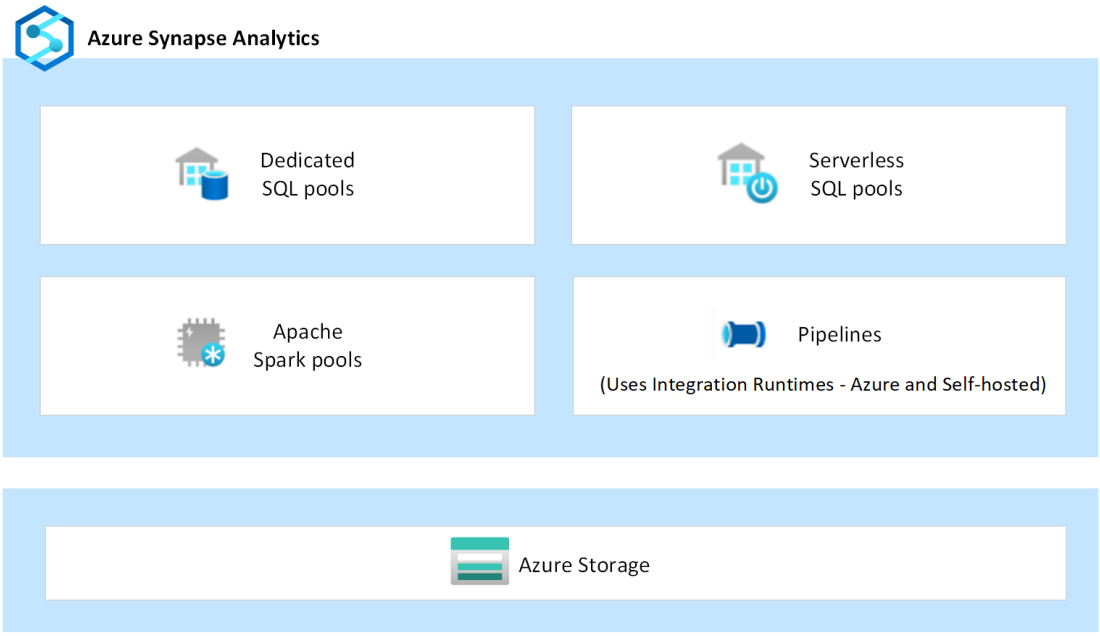
Apache Spark in Azure Synapse is one of Microsoft's implementations of open-source Apache Spark in the cloud. Spark instances are provisioned on-demand based on the metadata configurations defined in the Spark pools. Each user gets their own dedicated Spark instance for running their jobs. The data files processed by the Spark instances are managed by the customer in their own Azure Storage accounts.

Pipelines are a logical grouping of activities that perform data movement and data transformation at scale. Data flow is a transformation activity in a pipeline that's developed by using a low-code user interface. It can execute data transformations at scale. Behind the scenes, data flows use Apache Spark clusters of Azure Synapse to execute automatically generated code. Pipelines and data flows are compute-only services, and they don't have any managed storage associated with them.

Pipelines use the Integration Runtime (IR) as the scalable compute infrastructure for performing data movement and dispatch activities. Data movement activities run on the IR whereas the dispatch activities run on variety of other compute engines, including Azure SQL Database, Azure HDInsight, Azure Databricks, Apache Spark clusters of Azure Synapse, and others. Azure Synapse supports two types of IR: Azure Integration Runtime and Self-hosted Integration Runtime. The Azure IR provides a fully managed, scalable, and on-demand compute infrastructure. The Self-hosted IR is installed and configured by the customer in their own network, either in on-premises machines or in Azure cloud virtual machines.

Customers can choose to associate their Synapse workspace with a managed workspace virtual network. When associated with a managed workspace virtual network, Azure IRs and Apache Spark clusters that are used by pipelines, data flows, and the Apache Spark pools are deployed inside the managed workspace virtual network. This setup ensures network isolation between the workspaces for pipelines and Apache Spark workloads.

The following diagram depicts the various components of Azure Synapse.



Component isolation

Each individual component of Azure Synapse depicted in the diagram provides its own security features. Security features provide data protection, access control, authentication, network security, and threat protection for securing the compute and the associated data that's processed. Additionally, Azure Storage, being a PaaS service, provides additional security of its own, that's set up and managed by the customer in their own storage accounts. This level of component isolation limits and minimizes the exposure if there were a security vulnerability in any one of its components.

Security layers

Azure Synapse implements a multi-layered security architecture for end-to-end protection of your data. There are five layers:

- **Data protection** to identify and classify sensitive data, and encrypt data at rest and in motion.
- **Access control** to determine a user's right to interact with data.
- **Authentication** to prove the identity of users and applications.
- **Network security** to isolate network traffic with private endpoints and virtual private networks.
- **Threat protection** to identify potential security threats, such as unusual access locations, SQL injection attacks, authentication attacks, and more.

Next steps

In the next article in this white paper series, learn about data protection.

Azure Synapse Analytics security white paper: Data protection

8/19/2022 • 4 minutes to read • Edit Online

NOTE

This article forms part of the *Azure Synapse Analytics security white paper* series of articles. For an overview of the series, see Azure Synapse Analytics security white paper.

Data discovery and classification

Organizations need to protect their data to comply with federal, local, and company guidelines to mitigate risks of data breach. One challenge organizations face is: *How do you protect the data if you don't know where it is?* Another is: *What level of protection is needed?*—because some datasets require more protection than others.

Imagine an organization with hundreds or thousands of files stored in their data lake, and hundreds or thousands of tables in their databases. It would benefit from a process that automatically scans every row and column of the file system or table and classifies columns as *potentially* sensitive data. This process is known as *data discovery*.

Once the data discovery process is complete, it provides classification recommendations based on a predefined set of patterns, keywords, and rules. Someone can then review the recommendations and apply sensitivity-classification labels to appropriate columns. This process is known as *classification*.

Azure Synapse provides two options for data discovery and classification:

- Data Discovery & Classification, which is built into Azure Synapse and dedicated SQL pool (formerly SQL DW).
- Microsoft Purview, which is a unified data governance solution that helps manage and govern on-premises, multicloud, and software-as-a-service (SaaS) data. It can automate data discovery, lineage identification, and data classification. By producing a unified map of data assets and their relationships, it makes data easily discoverable.

NOTE

Microsoft Purview data discovery and classification is in public preview for Azure Synapse, dedicated SQL pool (formerly SQL DW), and serverless SQL pool. However, data lineage is currently not supported for Azure Synapse, dedicated SQL pool (formerly SQL DW), and serverless SQL pool. Apache Spark pool only supports lineage tracking.

Data encryption

Data is encrypted at rest and in transit.

Data at rest

By default, Azure Storage automatically encrypts all data using 256-bit Advanced Encryption Standard encryption (AES 256). It's one of the strongest block ciphers available and is FIPS 140-2 compliant. The platform manages the encryption key, and it forms the *first layer* of data encryption. This encryption applies to both user and system databases, including the **master** database.

Enabling Transparent Data Encryption (TDE) can add a *second layer* of data encryption for dedicated SQL pools. It performs real-time I/O encryption and decryption of database files, transaction logs files, and backups at rest without requiring any changes to the application. By default, it uses AES 256.

By default, TDE protects the database encryption key (DEK) with a built-in server certificate (service managed). There's an option to bring your own key (BYOK) that can be securely stored in Azure Key Vault.

Azure Synapse SQL serverless pool and Apache Spark pool are analytic engines that work directly on Azure Data Lake Gen2 (ALDS Gen2) or Azure Blob Storage. These analytic runtimes don't have any permanent storage and rely on Azure Storage encryption technologies for data protection. By default, Azure Storage encrypts all data using server-side encryption (SSE). It's enabled for all storage types (including ADLS Gen2) and cannot be disabled. SSE encrypts and decrypts data transparently using AES 256.

There are two SSE encryption options:

- **Microsoft-managed keys:** Microsoft manages every aspect of the encryption key, including key storage, ownership, and rotations. It's entirely transparent to customers.
- **Customer-managed keys:** In this case, the symmetric key used to encrypt data in Azure Storage is encrypted using a customer-provided key. It supports RSA and RSA-HSM (Hardware Security Modules) keys of sizes 2048, 3072, and 4096. Keys can be securely stored in Azure Key Vault or Azure Key Vault Managed HSM. It provides fine grain access control of the key and its management, including storage, backup, and rotations. For more information, see Customer-managed keys for Azure Storage encryption.

While SSE forms the first layer of encryption, cautious customers can double encrypt by enabling a second layer of 256-bit AES encryption at the Azure Storage infrastructure layer. Known as *infrastructure encryption*, it uses a platform-managed key together with a separate key from SSE. So, data in the storage account is encrypted twice; once at the service level and once at the infrastructure level with two different encryption algorithms and different keys.

Data in transit

Azure Synapse, dedicated SQL pool (formerly SQL DW), and serverless SQL pool use the Tabular Data Stream (TDS) protocol to communicate between the SQL pool endpoint and a client machine. TDS depends on Transport Layer Security (TLS) for channel encryption, ensuring all data packets are secured and encrypted between endpoint and client machine. It uses a signed server certificate from the Certificate Authority (CA) used for TLS encryption, managed by Microsoft. Azure Synapse supports data encryption in transit with TLS v1.2, using AES 256 encryption.

Azure Synapse leverages TLS to ensure data is encrypted in motion. SQL dedicated pools support TLS 1.0, TLS 1.1, and TLS 1.2 versions for encryption wherein Microsoft-provided drivers use TLS 1.2 by default. Serverless SQL pool and Apache Spark pool use TLS 1.2 for all outbound connections.

Next steps

In the next article in this white paper series, learn about access control.

Azure Synapse Analytics security white paper: Access control

8/19/2022 • 3 minutes to read • Edit Online

NOTE

This article forms part of the *Azure Synapse Analytics security white paper* series of articles. For an overview of the series, see Azure Synapse Analytics security white paper.

Depending on how the data has been modeled and stored, data governance and access control might require that developers and security administrators use different approaches, or combination of techniques, to implement a robust security foundation.

Azure Synapse supports a wide range of capabilities to control who can access what data. These capabilities are built upon a set of advanced access control features, including:

- Object-level security
- Row-level security
- Column-level security
- Dynamic data masking
- Synapse role-based access control

Object-level security

Every object in a dedicated SQL pool has associated permissions that can be granted to a principal. In the context of users and service accounts, that's how individual tables, views, stored procedures, and functions are secured. Object permissions, like SELECT, can be granted to user accounts (SQL logins, Azure Active Directory users or groups) and database roles, which provides flexibility for database administrators. Further, permissions granted on tables and views can be combined with other access control mechanisms (described below), such as column-level security, row-level security, and dynamic data masking.

In Azure Synapse, all permissions are granted to database-level users and roles. Additionally, any user granted the built-in Synapse Administrator RBAC role at the workspace level is automatically granted full access to all dedicated SQL pools.

In addition to securing SQL tables in Azure Synapse, dedicated SQL pool (formerly SQL DW), serverless SQL pool, and Spark tables can be secured too. By default, users assigned to the **Storage Blob Data Contributor** role of data lakes connected to the workspace have READ, WRITE, and EXECUTE permissions on all Spark-created tables *when users interactively execute code in notebook*. It's called *Azure Active Directory (Azure AD) pass-through*, and it applies to all data lakes connected to the workspace. However, if the same user executes the same notebook *through a pipeline*, the workspace Managed Service Identity (MSI) is used for authentication. So, for the pipeline to successfully execute workspace MSI, it must also belong to the **Storage Blob Data Contributor** role of the data lake that's accessed.

Row-level security

Row-level security allows security administrators to establish and control fine grained access to specific table rows based on the profile of a user (or a process) running a query. Profile or user characteristics may refer to group membership or execution context. Row-level security helps prevent unauthorized access when users

query data from the same tables but must see different subsets of data.

> **NOTE**
> Row-level security is supported in Azure Synapse and dedicated SQL pool (formerly SQL DW), but it's not supported for Apache Spark pool and serverless SQL pool.

Column-level security

Column-level security allows security administrators to set permissions that limit who can access sensitive columns in tables. It's set at the database level and can be implemented without the need to change the design of the data model or application tier.

> **NOTE**
> Column-level security is supported in Azure Synapse and dedicated SQL pool (formerly SQL DW), but it's not supported for Apache Spark pool and serverless SQL pool.

Dynamic data masking

Dynamic data masking allows security administrators to restrict sensitive data exposure by masking it on read to non-privileged users. It helps prevent unauthorized access to sensitive data by enabling administrators to determine how the data is displayed at query time. Based on the identity of the authenticated user and their group assignment in the SQL pool, a query returns either masked or unmasked data. Masking is always applied regardless of whether data is accessed directly from a table or by using a view or stored procedure.

> **NOTE**
> Dynamic data masking is supported in Azure Synapse and dedicated SQL pool (formerly SQL DW), but it's not supported for Apache Spark pool and serverless SQL pool.

Synapse role-based access control

Azure Synapse also includes Synapse role-based access control (RBAC) roles to manage different aspects of Synapse Studio. Leverage these built-in roles to assign permissions to users, groups, or other security principals to manage who can:

- Publish code artifacts and list or access published code artifacts.
- Execute code on Apache Spark pools and integration runtimes.
- Access linked (data) services that are protected by credentials.
- Monitor or cancel job executions, review job output and execution logs.

Next steps

In the next article in this white paper series, learn about authentication.

Azure Synapse Analytics security white paper: Authentication

8/19/2022 • 2 minutes to read • Edit Online

> **NOTE**
> This article forms part of the *Azure Synapse Analytics security white paper* series of articles. For an overview of the series, see Azure Synapse Analytics security white paper.

Authentication is the process of proving the user is who they claim to be. Authentication activities can be logged with Azure SQL Auditing, and an IT administrator can configure reports and alerts whenever a login from a suspicious location is attempted.

Benefits

Some of the benefits of these robust authentication mechanisms include:

- Strong password policies to deter brute force attacks.
- User password encryption.
- Firewall rules.
- SQL endpoints with Multi-factor authentication.
- Elimination of the need to manage credentials with managed identity.

Azure Synapse, dedicated SQL pool (formerly SQL DW), and serverless SQL pool currently support Azure Active Directory (Azure AD) authentication and SQL authentication, while Apache Spark pool supports only Azure AD authentication. Multi-factor authentication and managed identity are fully supported for Azure Synapse, dedicated SQL pool (formerly SQL DW), serverless SQL pool, and Apache Spark pool.

Next steps

In the next article in this white paper series, learn about network security.

Azure Synapse Analytics security white paper: Network security

8/19/2022 • 12 minutes to read • Edit Online

> **NOTE**
> This article forms part of the *Azure Synapse Analytics security white paper* series of articles. For an overview of the series, see Azure Synapse Analytics security white paper.

To secure Azure Synapse, there are a range of network security options to consider.

Network security terminology

This opening section provides an overview and definitions of some of key Azure Synapse terms related to network security. Keep these definitions in mind while reading this article.

Synapse workspace

A *Synapse workspace* is a securable logical collection of all services offered by Azure Synapse. It includes dedicated SQL pools (formerly SQL DW), serverless SQL pools, Apache Spark pools, pipelines, and other services. Certain network configuration settings, such as IP firewall rules, managed virtual network, and approved tenants for exfiltration protection, are configured and secured at the workspace level.

Synapse workspace endpoints

An endpoint is a point of an incoming connection to access a service. Each Synapse workspace has three distinct endpoints:

- **Dedicated SQL endpoint** for accessing dedicated SQL pools.
- **Serverless SQL endpoint** for accessing serverless SQL pools.
- **Development endpoint** for accessing Apache Spark pools and pipeline resources in the workspace.

These endpoints are automatically created when the Synapse workspace is created.

Synapse Studio

Synapse Studio is a secure web front-end development environment for Azure Synapse. It supports various roles, including the data engineer, data scientist, data developer, data analyst, and Synapse administrator.

Use Synapse Studio to perform various data and management operations in Azure Synapse, such as:

- Connecting to dedicated SQL pools, serverless SQL pools, and running SQL scripts.
- Developing and running notebooks on Apache Spark pools.
- Developing and running pipelines.
- Monitoring dedicated SQL pools, serverless SQL pools, Apache Spark pools, and pipeline jobs.
- Managing Synapse RBAC permissions of workspace items.
- Creating managed private endpoint connections to data sources and sinks.

Connections to workspace endpoints can be made using Synapse Studio. Also, it's possible to create private endpoints to ensure that communication to the workspace endpoints is private.

Public network access and firewall rules

By default, the workspace endpoints are *public endpoints* when they're provisioned. Access to these workspace endpoints from any public network is enabled, including networks that are outside the customer's organization, without requiring a VPN connection or an ExpressRoute connection to Azure.

All Azure services, including PaaS services like Azure Synapse, are protected by DDoS basic protection to mitigate malicious attacks (active traffic monitoring, always on detection, and automatic attack mitigations).

All traffic to workspace endpoints—even via public networks—is encrypted and secured in transit by Transport Level Security (TLS) protocol.

To protect any sensitive data, it's recommended to disable public access to the workspace endpoints entirely. By doing so, it ensures all workspace endpoints can only be accessed using private endpoints.

Disabling public access for all the Synapse workspaces in a subscription or a resource group is enforced by assigning an Azure Policy. It's also possible to disable public network access on per-workspace basis based on the sensitivity of data processed by the workspace.

However, if public access needs to be enabled, it's highly recommended to configure the IP firewall rules to allow inbound connections only from the specified list of public IP addresses.

Consider enabling public access when the on-premises environment doesn't have VPN access or ExpressRoute to Azure, and it requires access to the workspace endpoints. In this case, specify a list of public IP addresses of the on-premises data centers and gateways in the IP firewall rules.

Private endpoints

An Azure private endpoint is a virtual network interface with a private IP address that's created in the customer's own Azure Virtual Network (VNet) subnet. A private endpoint can be created for any Azure service that supports private endpoints, such as Azure Synapse, dedicated SQL pools (formerly SQL DW), Azure SQL Databases, Azure Storage, or any service in Azure powered by Azure Private Link service.

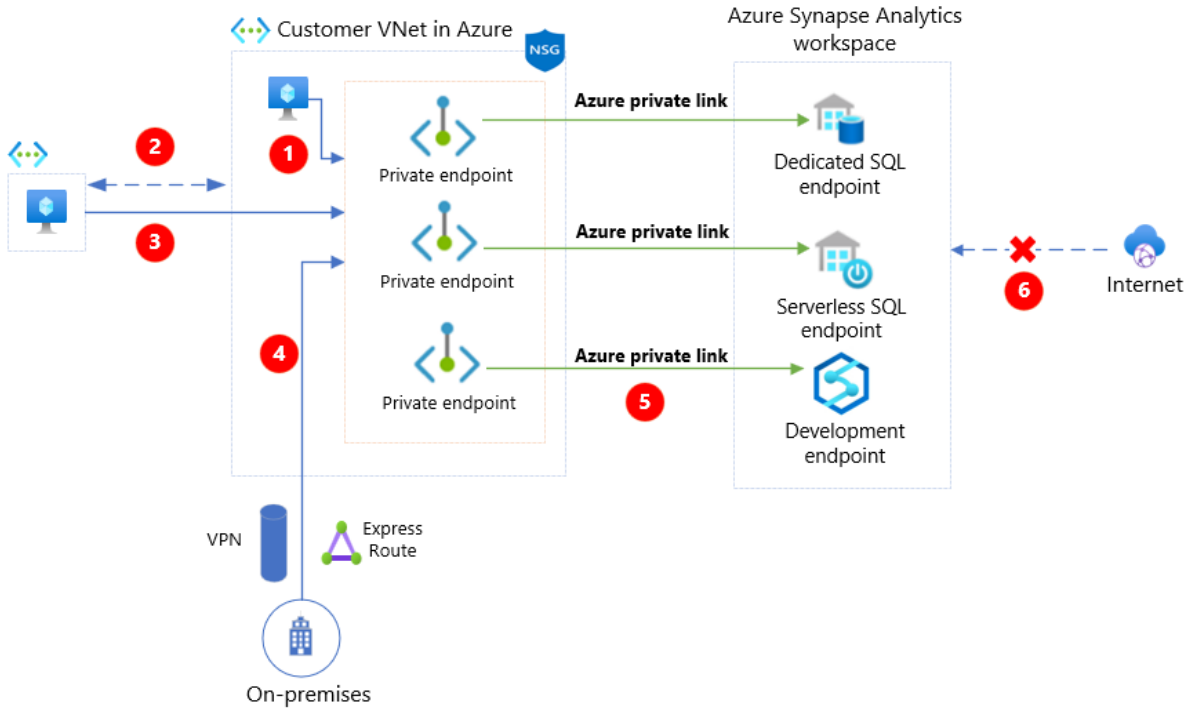
It's possible to create private endpoints in the VNet for all three Synapse workspace endpoints, individually. This way, there could be three private endpoints created for three endpoints of a Synapse workspace: one for dedicated SQL pool, one for serverless SQL pool, and one for the development endpoint.

Private endpoints have many security benefits compared to the public endpoints. Private endpoints in an Azure VNet can be accessed only from within:

- The same VNet that contains this private endpoint.
- Regionally or globally peered Azure VNets.
- On-premises networks connected to Azure via VPN Gateway or ExpressRoute.

The main benefit of private endpoints is that it's no longer necessary to expose workspace endpoints to the public internet. *The less exposure, the better.*

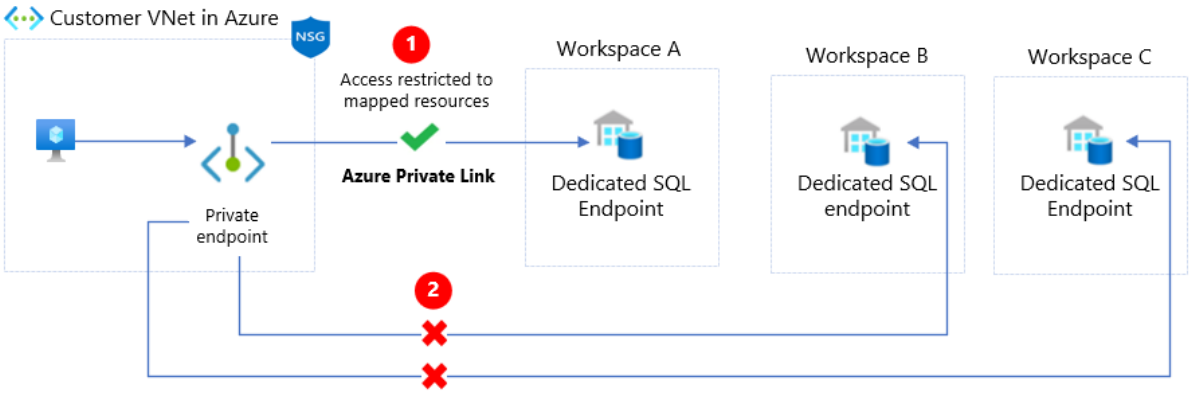
The following diagram depicts private endpoints.



The above diagram depicts the following key points:

ITEM	DESCRIPTION
1	Workstations from within the customer VNet access the Azure Synapse private endpoints.
2	Peering between customer VNet and another VNet.
3	Workstation from peered VNet access the Azure Synapse private endpoints.
4	On-premises network access the Azure Synapse private endpoints through VPN or ExpressRoute.
5	Workspace endpoints are mapped into customer's VNet through private endpoints using Azure Private Link service.
6	Public access is disabled on the Synapse workspace.

In the following diagram, a private endpoint is mapped to an instance of a PaaS resource instead of the entire service. In the event of a security incident within the network, only the mapped resource instance is exposed, minimizing the exposure and threat of data leakage and exfiltration.

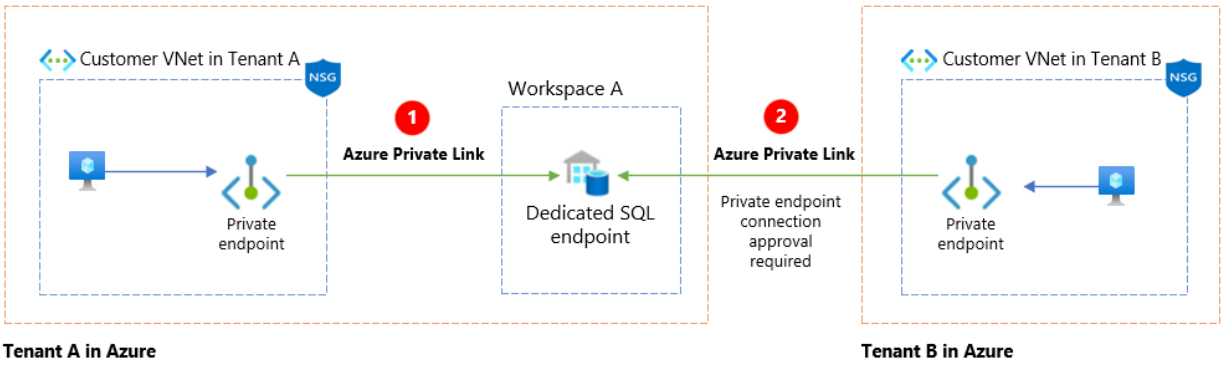


The above diagram depicts the following key points:

ITEM	DESCRIPTION
1	The private endpoint in the customer VNet is mapped to a single dedicated SQL pool (formerly SQL DW) endpoint in Workspace A.
2	Other SQL pool endpoints in the other workspaces (B and C) aren't accessible through this private endpoint, minimizing exposure.

Private endpoint works across Azure Active Directory (Azure AD) tenants and regions, so it's possible to create private endpoint connections to Synapse workspaces across tenants and regions. In this case, it goes through the private endpoint connection approval workflow. The resource owner controls which private endpoint connections are approved or denied. The resource owner is in full control of who can connect to their workspaces.

The following diagram depicts a private endpoint connection approval workflow.



The above diagram depicts the following key points:

ITEM	DESCRIPTION
1	Dedicated SQL pool (formerly SQL DW) in Workspace A in Tenant A is accessed by a private endpoint in the customer VNet in Tenant A.
2	The same dedicated SQL pool (formerly SQL DW) in Workspace A in Tenant A is accessed by a private endpoint in the customer VNet in Tenant B through a connection approval workflow.

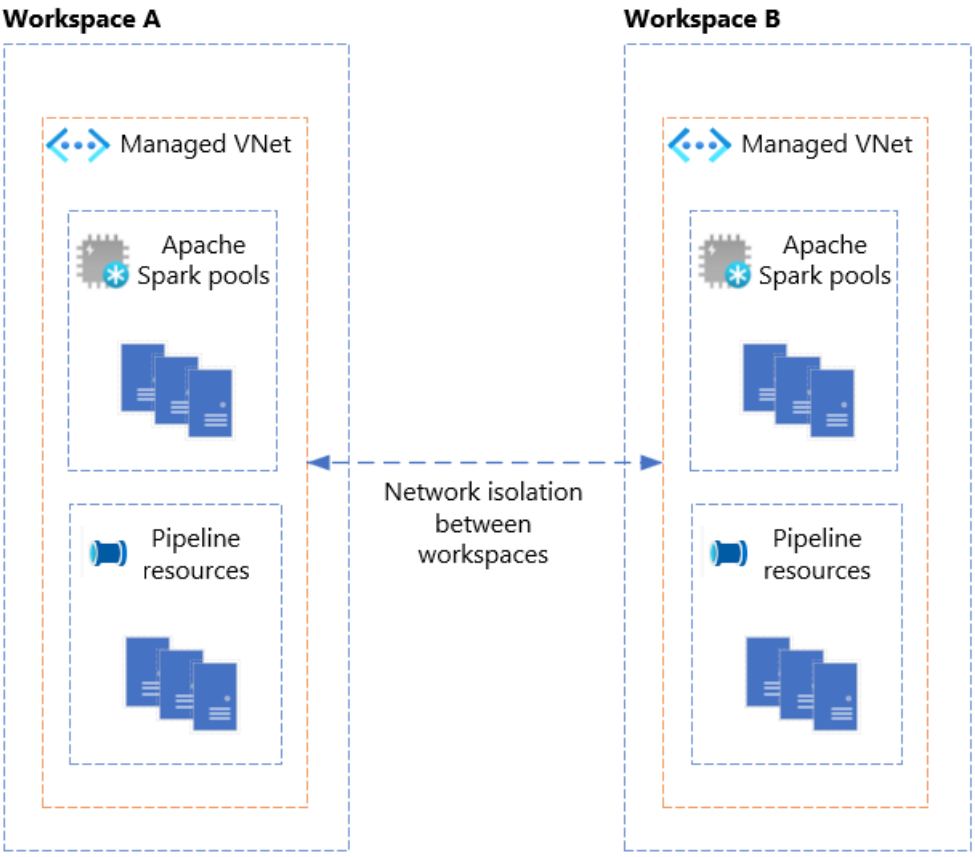
Managed VNet

The Synapse Managed VNet feature provides a fully managed network isolation for the Apache Spark pool and pipeline compute resources between Synapse workspaces. It can be configured at workspace creation time. In addition, it also provides network isolation for Spark clusters within the same workspace. Each workspace has its own virtual network, which is fully managed by Synapse. The Managed VNet isn't visible to the users to make any modifications. Any pipeline or Apache Spark pool compute resources that are spun up by Azure Synapse in a Managed VNet gets provisioned inside its own VNet. This way, there's full network isolation from other workspaces.

This configuration eliminates the need to create and manage VNets and network security groups for the Apache Spark pool and pipeline resources, as is typically done by VNet Injection.

As such, multi-tenant services in a Synapse workspace, such as dedicated SQL pools and serverless SQL pools, are **not** provisioned inside the Managed VNet.

The following diagram depicts network isolation between two Managed VNets of Workspaces A and B with their Apache Spark pools and pipeline resources inside the Managed VNets.



Managed private endpoint connection

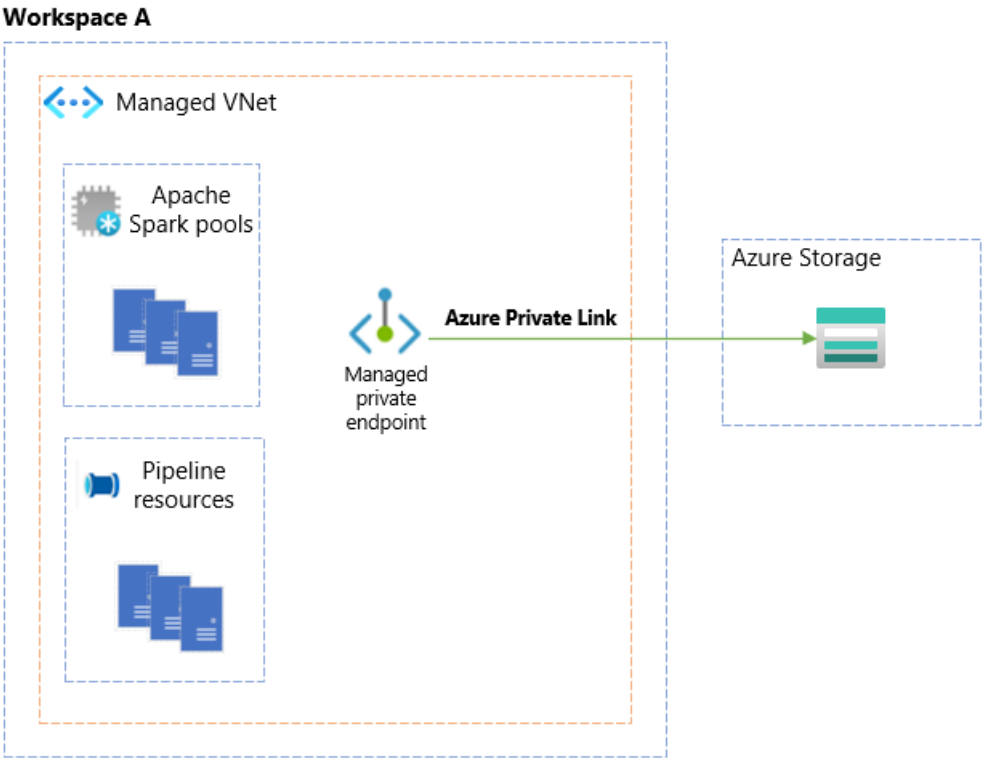
A managed private endpoint connection enables connections to any Azure PaaS service (that supports Private Link), securely and seamlessly, without the need to create a private endpoint for that service from the customer's VNet. Synapse automatically creates and manages the private endpoint. These connections are used by the compute resources that are provisioned inside the Synapse Managed VNet, such as Apache Spark pools and pipeline resources, to connect to the Azure PaaS services *privately*.

For example, if you want to connect to your Azure storage account *privately* from your pipeline, the usual approach is to create a private endpoint for the storage account and use a self-hosted integration runtime to connect to your storage private endpoint. With Synapse Managed VNets, you can privately connect to your

storage account using Azure integration runtime simply by creating a managed private endpoint connection directly to that storage account. This approach eliminates the need to have a self-hosted integration runtime to connect to your Azure PaaS services privately.

As such, multi-tenant services in a Synapse workspace, such as dedicated SQL pools and serverless SQL pools, are **not** provisioned inside the Managed VNet. So, they don't use the managed private endpoint connections created in the workspace for their outbound connectivity.

The following diagram depicts a managed private endpoint connecting to an Azure storage account from a Managed VNet in Workspace A.



Advanced Spark security

A Managed VNet also provides some added advantages for Apache Spark pool users. There's no need to worry about configuring a *fixed* subnet address space as would be done in VNet Injection. Azure Synapse automatically takes care of allocating these address spaces dynamically for workloads.

In addition, Spark pools operate as a job cluster. It means each user gets their own Spark cluster when interacting with the workspace. Creating an Spark pool within the workspace is metadata information for what will be assigned to the user when executing Spark workloads. It means each user will get their own Spark cluster *in a dedicated subnet inside the Managed VNet* to execute workloads. Spark pool sessions from the same user execute on the same compute resources. By providing this functionality, there are three main benefits:

- Greater security due to workload isolation based on the user.
- Reduction of noisy neighbors.
- Greater performance.

Data exfiltration protection

Synapse workspaces with Managed VNet have an additional security feature called *data exfiltration protection*. It protects all egress traffic going out from Azure Synapse from all services, including dedicated SQL pools, serverless SQL pools, Apache spark pools, and pipelines. It's configured by enabling data exfiltration protection at the workspace level (at workspace creation time) to restrict the outbound connections to an allowed list of

Azure Active Directory (Azure AD) tenants. By default, only the home tenant of the workspace is added to the list, but it's possible to add or modify the list of Azure AD tenants anytime after the workspace is created. Adding additional tenants is a highly privileged operation that requires the elevated role of Synapse Administrator. It effectively controls exfiltration of data from Azure Synapse to other organizations and tenants, without the need to have complicated network security policies in place.

For workspaces with data exfiltration protection enabled, Synapse pipelines and Apache Spark pools must use managed private endpoint connections for all their outbound connections.

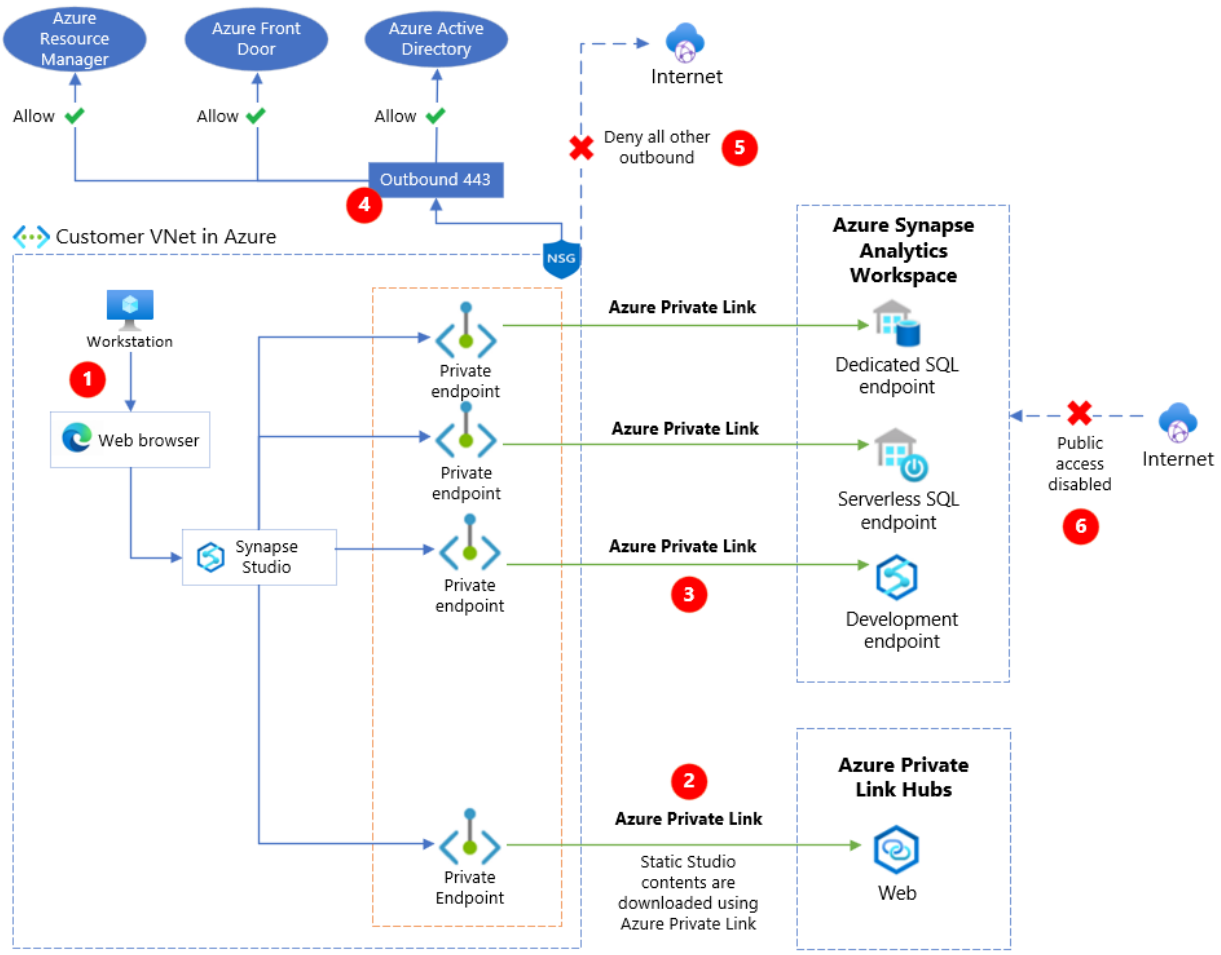
Dedicated SQL pool and serverless SQL pool don't use managed private endpoints for their outbound connectivity; however, any outbound connectivity from SQL pools can only be made to the *approved targets*, which are the targets of managed private endpoint connections.

Private link hubs for Synapse Studio

Synapse Private Link Hubs allows securely connecting to Synapse Studio from the customer's VNet using Azure Private Link. This feature is useful for customers who want to access the Synapse workspace using the Synapse Studio from a controlled and restricted environment, where the outbound internet traffic is restricted to a limited set of Azure services.

It's achieved by creating a private link hub resource and a private endpoint to this hub from the VNet. This private endpoint is then used to access the studio using its fully qualified domain name (FQDN), *web.azuresynapse.net*, with a private IP address from the VNet. The private link hub resource downloads the static contents of Synapse Studio over Azure Private Link to the user's workstation. In addition, separate private endpoints must be created for the individual workspace endpoints to ensure that communication to the workspace endpoints is private.

The following diagram depicts private link hubs for Synapse Studio.



The above diagram depicts the following key points:

ITEM	DESCRIPTION
1	The workstation in a restricted customer VNet accesses the Synapse Studio using a web browser.
2	A private endpoint created for private link hubs resource is used to download the static studio contents using Azure Private Link.
3	Private endpoints created for Synapse workspace endpoints access the workspace resources securely using Azure Private Links.
4	Network security group rules in the restricted customer VNet allow outbound traffic over port 443 to a limited set of Azure services, such as Azure Resource Manager, Azure Front Door, and Azure Active Directory.
5	Network security group rules in the restricted customer VNet deny all other outbound traffic from the VNet.
6	Public access is disabled on the Synapse workspace.

Dedicated SQL pool (formerly SQL DW)

Prior to the Azure Synapse offering, an Azure SQL data warehouse product named SQL DW was offered. It's now renamed as dedicated SQL pool (formerly SQL DW).

Dedicated SQL pool (formerly SQL DW) is created inside a logical Azure SQL server. It's a securable logical construct that acts as a central administrative point for a collection of databases including SQL DW and other Azure SQL databases.

Most of the core network security features discussed in the previous sections of this article for Azure Synapse are also applicable to dedicated SQL pool (formerly SQL DW). They include:

- IP firewall rules
- Disabling public network access
- Private endpoints
- Data exfiltration protection through outbound firewall rules

Since dedicated SQL pool (formerly SQL DW) is a multi-tenant service, it's not provisioned inside a Managed VNet. It means some of the features, such as Managed VNet and managed private endpoint connections, aren't applicable to it.

Network security feature matrix

The following comparison table provides a high-level overview of network security features supported across the Azure Synapse offerings:

FEATURE	AZURE SYNAPSE: APACHE SPARK POOL	AZURE SYNAPSE: DEDICATED SQL POOL	AZURE SYNAPSE: SERVERLESS SQL POOL	DEDICATED SQL POOL (FORMERLY SQL DW)
IP firewall rules	Yes	Yes	Yes	Yes
Disabling public access	Yes	Yes	Yes	Yes
Private endpoints	Yes	Yes	Yes	Yes
Data exfiltration protection	Yes	Yes	Yes	Yes
Secure access using Synapse Studio	Yes	Yes	Yes	No
Access from restricted network using Synapse private link hub	Yes	Yes	Yes	No
Managed VNet and workspace-level network isolation	Yes	N/A	N/A	N/A
Managed private endpoint connections for outbound connectivity	Yes	N/A	N/A	N/A
User-level network isolation	Yes	N/A	N/A	N/A

Next steps

In the next article in this white paper series, learn about threat protection.

Azure Synapse Analytics security white paper: Threat detection

8/19/2022 • 2 minutes to read • Edit Online

> **NOTE**
> This article forms part of the *Azure Synapse Analytics security white paper* series of articles. For an overview of the series, see Azure Synapse Analytics security white paper.

Azure Synapse provides SQL Auditing, SQL Threat Detection, and Vulnerability Assessment to audit, protect, and monitor databases.

Auditing

Auditing for Azure SQL Database and Azure Synapse tracks database events and writes them to an audit log in an Azure storage account, Log Analytics workspace, or Event Hubs. For any database, auditing is important. It produces an audit trail over time to help understand database activity and gain insight into discrepancies and anomalies that could indicate business concerns or suspected security violations. Used with Data discovery and classification, when any sensitive columns or tables are queried by users, entries will appear in a field named **data_sensitivity_information** of the **sql_audit_information** table.

> **NOTE**
> Azure SQL Auditing applies to Azure Synapse, dedicated SQL pool (formerly SQL DW), and serverless SQL pool, but it doesn't apply to Apache Spark pool.

Threat detection

Microsoft Defender for Cloud is a tool for security posture management and threat detection. It protects workloads running in Azure, including (but not exclusively) servers, app service, key vaults, Kubernetes services, storage accounts, and Azure SQL Databases.

As one of the options available with Microsoft Defender for Cloud, Microsoft Defender for SQL extends Defender for Cloud's data security package to secure databases. It can discover and mitigate potential database vulnerabilities by detecting anomalous activities that could be a potential threat to the database. Specifically, it continually monitors your database for:

- Potential SQL injection attacks
- Anomalous database access and queries
- Suspicious database activity

Alert notifications include details of the incident, and recommendations on how to investigate and remediate threats.

> **NOTE**
> Microsoft Defender for SQL applies to Azure Synapse and dedicated SQL pool (formerly SQL DW). It doesn't apply to serverless SQL pool or Apache Spark pool.

Vulnerability assessment

SQL vulnerability assessment is part of the Microsoft Defender for SQL offering. It continually monitors the data warehouse, ensuring that databases are always maintained at a high level of security and that organizational policies are met. It provides a comprehensive security report along with actionable remediation steps for each issue found, making it easy to proactively manage database security stature even if you're not a security expert.

> **NOTE**
> SQL vulnerability assessment applies to Azure Synapse and dedicated SQL pool (formerly SQL DW). It doesn't apply to serverless SQL pool or Apache Spark pool.

Compliance

For an overview of Azure compliance offerings, download the latest version of the Microsoft Azure Compliance Offerings document.

Next steps

For more information related to this white paper, check out the following resources:

- Azure Synapse Analytics Blog
- Azure security baseline for Azure Synapse dedicated SQL pool (formerly SQL DW)
- Overview of the Azure Security Benchmark (v3)
- Security baselines for Azure

Database collation support for Synapse SQL in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

Collations provide the locale, code page, sort order, and character sensitivity rules for character-based data types. Once chosen, all columns and expressions requiring collation information inherit the chosen collation from the database setting. The default inheritance can be overridden by explicitly stating a different collation for a character-based data type.

You can change the default database collation from the Azure portal when you create a new dedicated SQL pool database. This capability makes it even easier to create a new database using one of the 3800 supported database collations.

You can specify the default serverless SQL pool database collation at creation time using CREATE DATABASE statement.

> **NOTE**
> In Azure Synapse Analytics, query text (including variables, constants, etc.) is always handled using the database-level collation, and not the server-level collation as in other SQL Server offerings.

Change collation

To change the default collation for dedicated SQL pool database, update to the Collation field in the provisioning experience. For example, if you wanted to change the default collation to case sensitive, you would rename the Collation from SQL_Latin1_General_CP1_CI_AS to SQL_Latin1_General_CP1_CS_AS.

To change the default collation for a serverless SQL pool database, you can use ALTER DATABASE statement.

List of unsupported collation types for Dedicated SQL pools

- Japanese_Bushu_Kakusu_140_BIN
- Japanese_Bushu_Kakusu_140_BIN2
- Japanese_Bushu_Kakusu_140_CI_AI_VSS
- Japanese_Bushu_Kakusu_140_CI_AI_WS_VSS
- Japanese_Bushu_Kakusu_140_CI_AI_KS_VSS
- Japanese_Bushu_Kakusu_140_CI_AI_KS_WS_VSS
- Japanese_Bushu_Kakusu_140_CI_AS_VSS
- Japanese_Bushu_Kakusu_140_CI_AS_WS_VSS
- Japanese_Bushu_Kakusu_140_CI_AS_KS_VSS
- Japanese_Bushu_Kakusu_140_CI_AS_KS_WS_VSS
- Japanese_Bushu_Kakusu_140_CS_AI_VSS
- Japanese_Bushu_Kakusu_140_CS_AI_WS_VSS
- Japanese_Bushu_Kakusu_140_CS_AI_KS_VSS
- Japanese_Bushu_Kakusu_140_CS_AI_KS_WS_VSS
- Japanese_Bushu_Kakusu_140_CS_AS_VSS
- Japanese_Bushu_Kakusu_140_CS_AS_WS_VSS

- Japanese_Bushu_Kakusu_140_CS_AS_KS_VSS
- Japanese_Bushu_Kakusu_140_CS_AS_KS_WS_VSS
- Japanese_Bushu_Kakusu_140_CI_AI
- Japanese_Bushu_Kakusu_140_CI_AI_WS
- Japanese_Bushu_Kakusu_140_CI_AI_KS
- Japanese_Bushu_Kakusu_140_CI_AI_KS_WS
- Japanese_Bushu_Kakusu_140_CI_AS
- Japanese_Bushu_Kakusu_140_CI_AS_WS
- Japanese_Bushu_Kakusu_140_CI_AS_KS
- Japanese_Bushu_Kakusu_140_CI_AS_KS_WS
- Japanese_Bushu_Kakusu_140_CS_AI
- Japanese_Bushu_Kakusu_140_CS_AI_WS
- Japanese_Bushu_Kakusu_140_CS_AI_KS
- Japanese_Bushu_Kakusu_140_CS_AI_KS_WS
- Japanese_Bushu_Kakusu_140_CS_AS
- Japanese_Bushu_Kakusu_140_CS_AS_WS
- Japanese_Bushu_Kakusu_140_CS_AS_KS
- Japanese_Bushu_Kakusu_140_CS_AS_KS_WS
- Japanese_XJIS_140_BIN
- Japanese_XJIS_140_BIN2
- Japanese_XJIS_140_CI_AI_VSS
- Japanese_XJIS_140_CI_AI_WS_VSS
- Japanese_XJIS_140_CI_AI_KS_VSS
- Japanese_XJIS_140_CI_AI_KS_WS_VSS
- Japanese_XJIS_140_CI_AS_VSS
- Japanese_XJIS_140_CI_AS_WS_VSS
- Japanese_XJIS_140_CI_AS_KS_VSS
- Japanese_XJIS_140_CI_AS_KS_WS_VSS
- Japanese_XJIS_140_CS_AI_VSS
- Japanese_XJIS_140_CS_AI_WS_VSS
- Japanese_XJIS_140_CS_AI_KS_VSS
- Japanese_XJIS_140_CS_AI_KS_WS_VSS
- Japanese_XJIS_140_CS_AS_VSS
- Japanese_XJIS_140_CS_AS_WS_VSS
- Japanese_XJIS_140_CS_AS_KS_VSS
- Japanese_XJIS_140_CS_AS_KS_WS_VSS
- Japanese_XJIS_140_CI_AI
- Japanese_XJIS_140_CI_AI_WS
- Japanese_XJIS_140_CI_AI_KS
- Japanese_XJIS_140_CI_AI_KS_WS
- Japanese_XJIS_140_CI_AS
- Japanese_XJIS_140_CI_AS_WS
- Japanese_XJIS_140_CI_AS_KS
- Japanese_XJIS_140_CI_AS_KS_WS
- Japanese_XJIS_140_CS_AI
- Japanese_XJIS_140_CS_AI_WS

- Japanese_XJIS_140_CS_AI_KS
- Japanese_XJIS_140_CS_AI_KS_WS
- Japanese_XJIS_140_CS_AS
- Japanese_XJIS_140_CS_AS_WS
- Japanese_XJIS_140_CS_AS_KS
- Japanese_XJIS_140_CS_AS_KS_WS
- SQL_EBCDIC1141_CP1_CS_AS
- SQL_EBCDIC277_2_CP1_CS_AS
- UTF-8

Check the current collation

To check the current collation for the database, you can run the following T-SQL snippet:

```
SELECT DATABASEPROPERTYEX(DB_NAME(), 'Collation') AS Collation;
```

When passed 'Collation' as the property parameter, the DatabasePropertyEx function returns the current collation for the database specified. You can learn more about the DatabasePropertyEx function on MSDN.

Next steps

Additional information on best practices for dedicated SQL pool and serverless SQL pool can be found in the following articles:

- Best Practices for dedicated SQL pool
- Best practices for serverless SQL pool

Introduction to Microsoft Spark Utilities

8/19/2022 • 16 minutes to read • Edit Online

Microsoft Spark Utilities (MSSparkUtils) is a builtin package to help you easily perform common tasks. You can use MSSparkUtils to work with file systems, to get environment variables, to chain notebooks together, and to work with secrets. MSSparkUtils are available in `PySpark (Python)`, `Scala`, and `.NET Spark (C#)` notebooks and Synapse pipelines.

Pre-requisites

Configure access to Azure Data Lake Storage Gen2

Synapse notebooks use Azure Active Directory (Azure AD) pass-through to access the ADLS Gen2 accounts. You need to be a **Storage Blob Data Contributor** to access the ADLS Gen2 account (or folder).

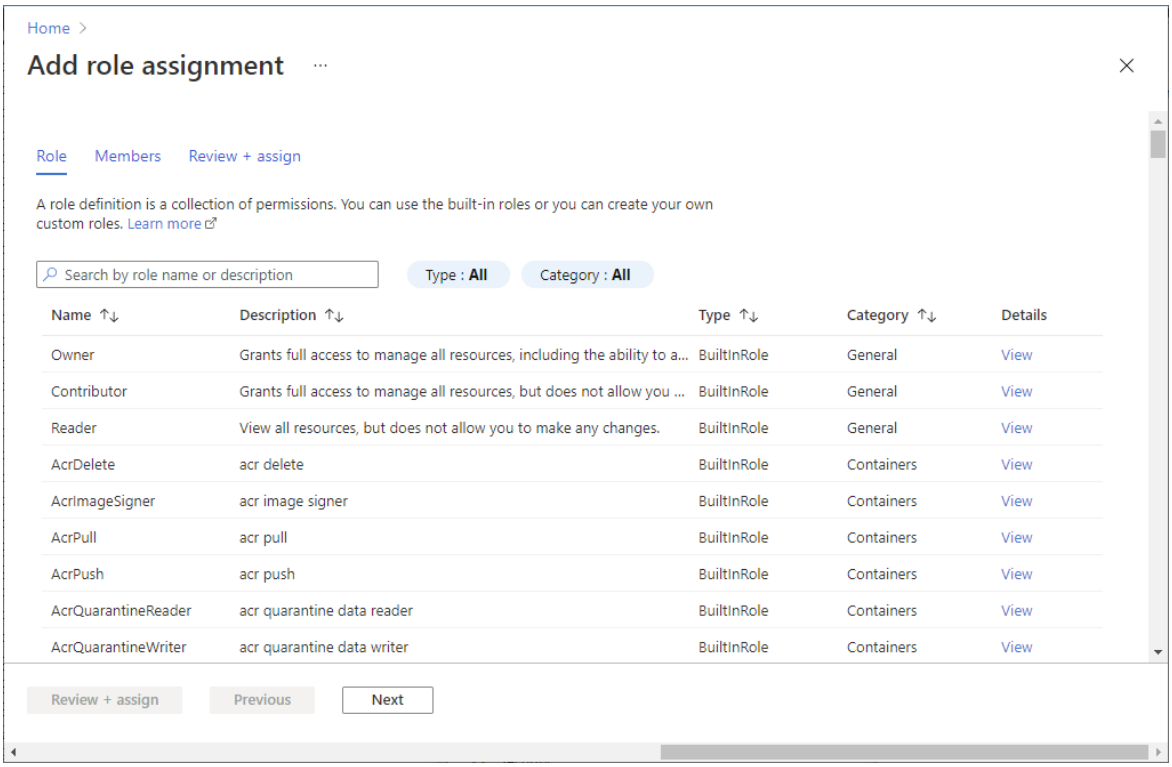
Synapse pipelines use workspace's Managed Service Identity (MSI) to access the storage accounts. To use MSSparkUtils in your pipeline activities, your workspace identity needs to be **Storage Blob Data Contributor** to access the ADLS Gen2 account (or folder).

Follow these steps to make sure your Azure AD and workspace MSI have access to the ADLS Gen2 account:

1. Open the Azure portal and the storage account you want to access. You can navigate to the specific container you want to access.
2. Select the **Access control (IAM)** from the left panel.
3. Select **Add** > **Add role assignment** to open the Add role assignment page.
4. Assign the following role. For detailed steps, see Assign Azure roles using the Azure portal.

SETTING	VALUE
Role	Storage Blob Data Contributor
Assign access to	USER and MANAGEDIDENTITY
Members	your Azure AD account and your workspace identity

> **NOTE**
> The managed identity name is also the workspace name.



5. Select **Save**.

You can access data on ADLS Gen2 with Synapse Spark via the following URL:

```
abfss://<container_name>@<storage_account_name>.dfs.core.windows.net/<path>
```

Configure access to Azure Blob Storage

Synapse use **Shared access signature (SAS)** to access Azure Blob Storage. To avoid exposing SAS keys in the code, we recommend creating a new linked service in Synapse workspace to the Azure Blob Storage account you want to access.

Follow these steps to add a new linked service for an Azure Blob Storage account:

1. Open the Azure Synapse Studio.
2. Select **Manage** from the left panel and select **Linked services** under the **External connections**.
3. Search **Azure Blob Storage** in the **New linked Service** panel on the right.
4. Select **Continue**.
5. Select the Azure Blob Storage Account to access and configure the linked service name. Suggest using **Account key** for the **Authentication method**.
6. Select **Test connection** to validate the settings are correct.
7. Select **Create** first and click **Publish all** to save your changes.

You can access data on Azure Blob Storage with Synapse Spark via following URL:

```
wasb[s]://<container_name>@<storage_account_name>.blob.core.windows.net/<path>
```

Here is a code example:

```
from pyspark.sql import SparkSession

# Azure storage access info
blob_account_name = 'Your account name' # replace with your blob name
blob_container_name = 'Your container name' # replace with your container name
blob_relative_path = 'Your path' # replace with your relative folder path
linked_service_name = 'Your linked service name' # replace with your linked service name

blob_sas_token = mssparkutils.credentials.getConnectionStringOrCreds(linked_service_name)

# Allow SPARK to access from Blob remotely

wasb_path = 'wasbs://%s@%s.blob.core.windows.net/%s' % (blob_container_name, blob_account_name,
blob_relative_path)

spark.conf.set('fs.azure.sas.%s.%s.blob.core.windows.net' % (blob_container_name, blob_account_name),
blob_sas_token)
print('Remote blob path: ' + wasb_path)
```

```
val blob_account_name = "" // replace with your blob name
val blob_container_name = "" //replace with your container name
val blob_relative_path = "/" //replace with your relative folder path
val linked_service_name = "" //replace with your linked service name


val blob_sas_token = mssparkutils.credentials.getConnectionStringOrCreds(linked_service_name)

val wasbs_path =
f"wasbs://$blob_container_name@$blob_account_name.blob.core.windows.net/$blob_relative_path"
spark.conf.set(f"fs.azure.sas.$blob_container_name.$blob_account_name.blob.core.windows.net",blob_sas_token)
```

```
var blob_account_name = "";  // replace with your blob name
var blob_container_name = "";     // replace with your container name
var blob_relative_path = "";  // replace with your relative folder path
var linked_service_name = "";    // replace with your linked service name
var blob_sas_token = Credentials.GetConnectionStringOrCreds(linked_service_name);

spark.SparkContext.GetConf().Set($"fs.azure.sas.{blob_container_name}.
{blob_account_name}.blob.core.windows.net", blob_sas_token);

var wasbs_path =
$"wasbs://{blob_container_name}@{blob_account_name}.blob.core.windows.net/{blob_relative_path}";

Console.WriteLine(wasbs_path);
```

Configure access to Azure Key Vault

You can add an Azure Key Vault as a linked service to manage your credentials in Synapse. Follow these steps to add an Azure Key Vault as a Synapse linked service:

1. Open the Azure Synapse Studio.
2. Select **Manage** from the left panel and select **Linked services** under the **External connections**.
3. Search **Azure Key Vault** in the **New linked Service** panel on the right.
4. Select the Azure Key Vault Account to access and configure the linked service name.
5. Select **Test connection** to validate the settings are correct.
6. Select **Create** first and click **Publish all** to save your change.

Synapse notebooks use Azure active directory(Azure AD) pass-through to access Azure Key Vault. Synapse

pipelines use workspace identity(MSI) to access Azure Key Vault. To make sure your code work both in notebook and in Synapse pipeline, we recommend granting secret access permission for both your Azure AD account and workspace identity.

Follow these steps to grant secret access to your workspace identity:

1. Open the Azure portal and the Azure Key Vault you want to access.
2. Select the **Access policies** from the left panel.
3. Select **Add Access Policy**:
   - Choose **Key, Secret, & Certificate Management** as config template.
   - Select **your Azure AD account** and **your workspace identity** (same as your workspace name) in the select principal or make sure it is already assigned.
4. Select **Select** and **Add**.
5. Select the **Save** button to commit changes.

File system utilities

`mssparkutils.fs` provides utilities for working with various file systems, including Azure Data Lake Storage Gen2 (ADLS Gen2) and Azure Blob Storage. Make sure you configure access to Azure Data Lake Storage Gen2 and Azure Blob Storage appropriately.

Run the following commands for an overview of the available methods:

```
from notebookutils import mssparkutils
mssparkutils.fs.help()
```

```
mssparkutils.fs.help()
```

```
using Microsoft.Spark.Extensions.Azure.Synapse.Analytics.Notebook.MSSparkUtils;
FS.Help()
```

Results in:

```
mssparkutils.fs provides utilities for working with various FileSystems.

Below is overview about the available methods:

cp(from: String, to: String, recurse: Boolean = false): Boolean -> Copies a file or directory, possibly
across FileSystems
mv(from: String, to: String, recurse: Boolean = false): Boolean -> Moves a file or directory, possibly
across FileSystems
ls(dir: String): Array -> Lists the contents of a directory
mkdirs(dir: String): Boolean -> Creates the given directory if it does not exist, also creating any
necessary parent directories
put(file: String, contents: String, overwrite: Boolean = false): Boolean -> Writes the given String out to a
file, encoded in UTF-8
head(file: String, maxBytes: int = 1024 * 100): String -> Returns up to the first 'maxBytes' bytes of the
given file as a String encoded in UTF-8
append(file: String, content: String, createFileIfNotExists: Boolean): Boolean -> Append the content to a
file
rm(dir: String, recurse: Boolean = false): Boolean -> Removes a file or directory

Use mssparkutils.fs.help("methodName") for more info about a method.
```

List files

List the content of a directory.

```
mssparkutils.fs.ls('Your directory path')
```

```
mssparkutils.fs.ls("Your directory path")
```

```
FS.Ls("Your directory path")
```

View file properties

Returns file properties including file name, file path, file size, and whether it is a directory and a file.

```
files = mssparkutils.fs.ls('Your directory path')
for file in files:
    print(file.name, file.isDir, file.isFile, file.path, file.size)
```

```
val files = mssparkutils.fs.ls("/")
files.foreach{
    file => println(file.name,file.isDir,file.isFile,file.size)
}
```

```
var Files = FS.Ls("/");
foreach(var File in Files) {
    Console.WriteLine(File.Name+" "+File.IsDir+" "+File.IsFile+" "+File.Size);
}
```

Create new directory

Creates the given directory if it does not exist and any necessary parent directories.

```
mssparkutils.fs.mkdirs('new directory name')
```

```
mssparkutils.fs.mkdirs("new directory name")
```

```
FS.Mkdirs("new directory name")
```

Copy file

Copies a file or directory. Supports copy across file systems.

```
mssparkutils.fs.cp('source file or directory', 'destination file or directory', True)# Set the third
parameter as True to copy all files and directories recursively
```

```
mssparkutils.fs.cp("source file or directory", "destination file or directory", true) // Set the third
parameter as True to copy all files and directories recursively
```

```
FS.Cp("source file or directory", "destination file or directory", true) // Set the third parameter as True to copy all files and directories recursively
```

Preview file content

Returns up to the first 'maxBytes' bytes of the given file as a String encoded in UTF-8.

```
mssparkutils.fs.head('file path', maxBytes to read)
```

```
mssparkutils.fs.head("file path", maxBytes to read)
```

```
FS.Head("file path", maxBytes to read)
```

Move file

Moves a file or directory. Supports move across file systems.

```
mssparkutils.fs.mv('source file or directory', 'destination directory', True) # Set the last parameter as True to firstly create the parent directory if it does not exist
```

```
mssparkutils.fs.mv("source file or directory", "destination directory", true) // Set the last parameter as True to firstly create the parent directory if it does not exist
```

```
FS.Mv("source file or directory", "destination directory", true)
```

Write file

Writes the given string out to a file, encoded in UTF-8.

```
mssparkutils.fs.put("file path", "content to write", True) # Set the last parameter as True to overwrite the file if it existed already
```

```
mssparkutils.fs.put("file path", "content to write", true) // Set the last parameter as True to overwrite the file if it existed already
```

```
FS.Put("file path", "content to write", true) // Set the last parameter as True to overwrite the file if it existed already
```

Append content to a file

Appends the given string to a file, encoded in UTF-8.

```
mssparkutils.fs.append("file path", "content to append", True) # Set the last parameter as True to create the file if it does not exist
```

```
mssparkutils.fs.append("file path","content to append",true) // Set the last parameter as True to create the file if it does not exist
```

```
FS.Append("file path", "content to append", true) // Set the last parameter as True to create the file if it does not exist
```

Delete file or directory

Removes a file or a directory.

```
mssparkutils.fs.rm('file path', True) # Set the last parameter as True to remove all files and directories recursively
```

```
mssparkutils.fs.rm("file path", true) // Set the last parameter as True to remove all files and directories recursively
```

```
FS.Rm("file path", true) // Set the last parameter as True to remove all files and directories recursively
```

Notebook utilities

Not supported.

You can use the MSSparkUtils Notebook Utilities to run a notebook or exit a notebook with a value. Run the following command to get an overview of the available methods:

```
mssparkutils.notebook.help()
```

Get results:

```
The notebook module.

exit(value: String): void -> This method lets you exit a notebook with a value.
run(path: String, timeoutSeconds: int, arguments: Map): String -> This method runs a notebook and returns its exit value.
```

Reference a notebook

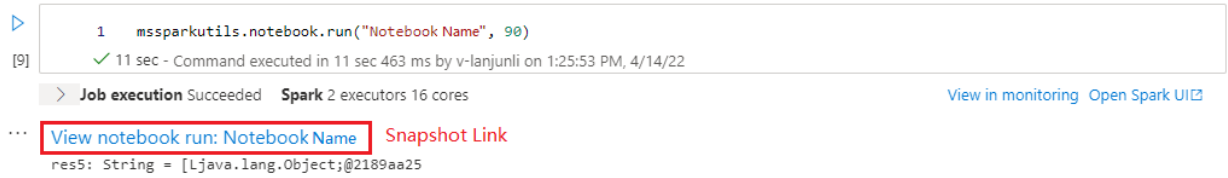
Reference a notebook and returns its exit value. You can run nesting function calls in a notebook interactively or in a pipeline. The notebook being referenced will run on the Spark pool of which notebook calls this function.

```
mssparkutils.notebook.run("notebook path", <timeoutSeconds>, <parameterMap>)
```

For example:

```
mssparkutils.notebook.run("folder/Sample1", 90, {"input": 20 })
```

After the run finished, you will see a snapshot link named '**View notebook run: *Notebook Name***' shown in the cell output, you can click the link to see the snapshot for this specific run.



Exit a notebook

Exits a notebook with a value. You can run nesting function calls in a notebook interactively or in a pipeline.

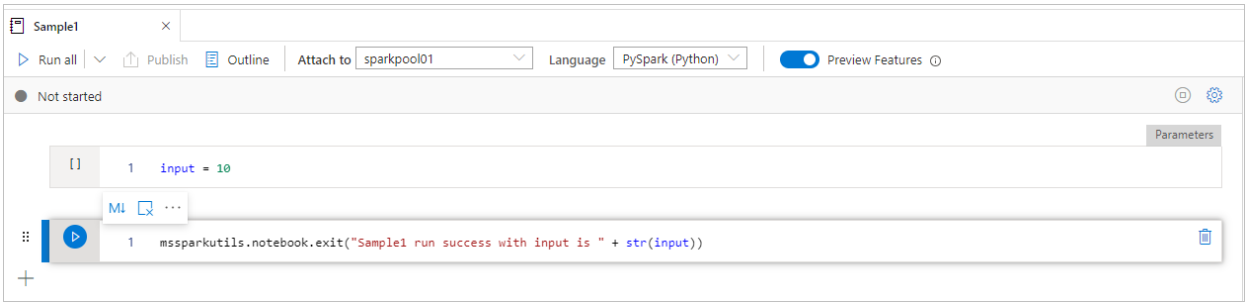
- When you call an `exit()` function a notebook interactively, Azure Synapse will throw an exception, skip running subsequence cells, and keep Spark session alive.
- When you orchestrate a notebook that calls an `exit()` function in a Synapse pipeline, Azure Synapse will return an exit value, complete the pipeline run, and stop the Spark session.
- When you call an `exit()` function in a notebook being referenced, Azure Synapse will stop the further execution in the notebook being referenced, and continue to run next cells in the notebook that call the `run()` function. For example: Notebook1 has three cells and calls an `exit()` function in the second cell. Notebook2 has five cells and calls `run(notebook1)` in the third cell. When you run Notebook2, Notebook1 will be stopped at the second cell when hitting the `exit()` function. Notebook2 will continue to run its fourth cell and fifth cell.

```
mssparkutils.notebook.exit("value string")
```

For example:

Sample1 notebook locates under **folder/** with following two cells:

- cell 1 defines an **input** parameter with default value set to 10.
- cell 2 exits the notebook with **input** as exit value.



You can run the **Sample1** in another notebook with default values:

```
exitVal = mssparkutils.notebook.run("folder/Sample1")
print (exitVal)
```

Results in:

```
Sample1 run success with input is 10
```

You can run the **Sample1** in another notebook and set the **input** value as 20:

```
exitVal = mssparkutils.notebook.run("mssparkutils/folder/Sample1", 90, {"input": 20 })
print (exitVal)
```

Results in:

```
Sample1 run success with input is 20
```

You can use the MSSparkUtils Notebook Utilities to run a notebook or exit a notebook with a value. Run the following command to get an overview of the available methods:

```
mssparkutils.notebook.help()
```

Get results:

```
The notebook module.

exit(value: String): void -> This method lets you exit a notebook with a value.
run(path: String, timeoutSeconds: int, arguments: Map): String -> This method runs a notebook and returns
its exit value.
```

Reference a notebook

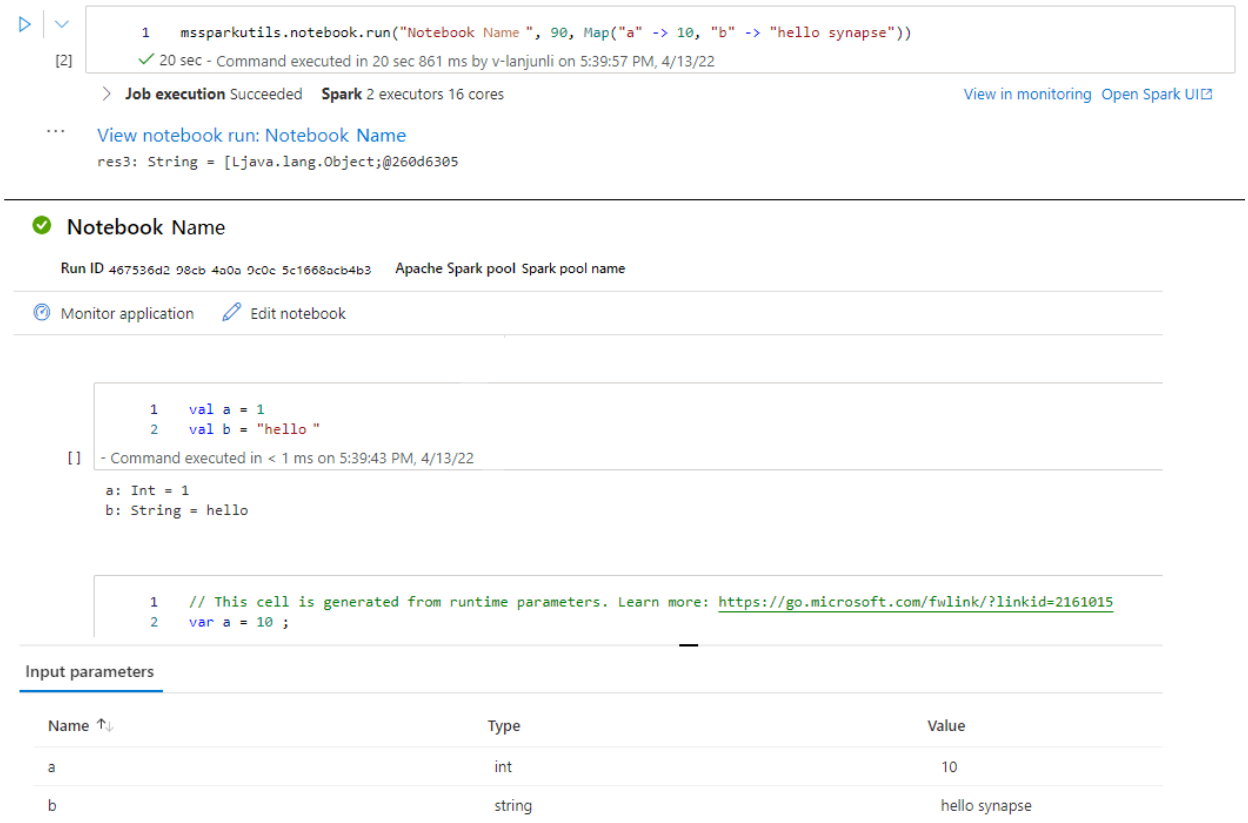
Reference a notebook and returns its exit value. You can run nesting function calls in a notebook interactively or in a pipeline. The notebook being referenced will run on the Spark pool of which notebook calls this function.

```
mssparkutils.notebook.run("notebook path", <timeoutSeconds>, <parameterMap>)
```

For example:

```
mssparkutils.notebook.run("folder/Sample1", 90, Map("input" -> 20))
```

After the run finished, you will see a snapshot link named '**View notebook run: *Notebook Name***' shown in the cell output, you can click the link to see the snapshot for this specific run.



Exit a notebook

Exits a notebook with a value. You can run nesting function calls in a notebook interactively or in a pipeline.

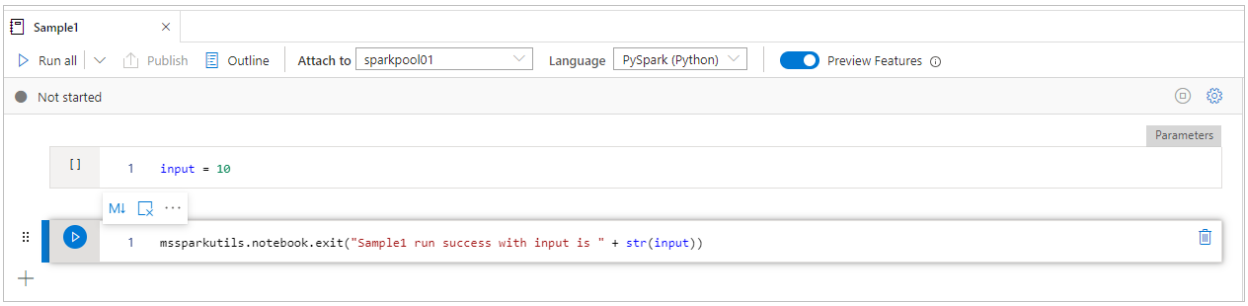
- When you call an `exit()` function a notebook interactively, Azure Synapse will throw an exception, skip running subsequence cells, and keep Spark session alive.
- When you orchestrate a notebook that calls an `exit()` function in a Synapse pipeline, Azure Synapse will return an exit value, complete the pipeline run, and stop the Spark session.
- When you call an `exit()` function in a notebook being referenced, Azure Synapse will stop the further execution in the notebook being referenced, and continue to run next cells in the notebook that call the `run()` function. For example: Notebook1 has three cells and calls an `exit()` function in the second cell. Notebook2 has five cells and calls `run(notebook1)` in the third cell. When you run Notebook2, Notebook1 will be stopped at the second cell when hitting the `exit()` function. Notebook2 will continue to run its fourth cell and fifth cell.

```
mssparkutils.notebook.exit("value string")
```

For example:

Sample1 notebook locates under **mssparkutils/folder/** with following two cells:

- cell 1 defines an **input** parameter with default value set to 10.
- cell 2 exits the notebook with **input** as exit value.



You can run the **Sample1** in another notebook with default values:

```
val exitVal = mssparkutils.notebook.run("mssparkutils/folder/Sample1")
print(exitVal)
```

Results in:

```
exitVal: String = Sample1 run success with input is 10
Sample1 run success with input is 10
```

You can run the **Sample1** in another notebook and set the **input** value as 20:

```
val exitVal = mssparkutils.notebook.run("mssparkutils/folder/Sample1", 90, {"input": 20 })
print(exitVal)
```

Results in:

```
exitVal: String = Sample1 run success with input is 20
Sample1 run success with input is 20
```

Credentials utilities

You can use the MSSparkUtils Credentials Utilities to get the access tokens of linked services and manage secrets in Azure Key Vault.

Run the following command to get an overview of the available methods:

```
mssparkutils.credentials.help()
```

```
mssparkutils.credentials.help()
```

```
Credentials.Help()
```

Get result:

```
getToken(audience, name): returns AAD token for a given audience, name (optional)
isValidToken(token): returns true if token hasn't expired
getConnectionStringOrCreds(linkedService): returns connection string or credentials for linked service
getSecret(akvName, secret, linkedService): returns AKV secret for a given AKV linked service, akvName,
secret key
getSecret(akvName, secret): returns AKV secret for a given akvName, secret key
putSecret(akvName, secretName, secretValue, linkedService): puts AKV secret for a given akvName, secretName
putSecret(akvName, secretName, secretValue): puts AKV secret for a given akvName, secretName
```

Get token

Returns Azure AD token for a given audience, name (optional). The table below list all the available audience types:

AUDIENCE TYPE	AUDIENCE KEY
Audience Resolve Type	'Audience'
Storage Audience Resource	'Storage'
Dedicated SQL pools (Data warehouse)	'DW'
Data Lake Audience Resource	'AzureManagement'
Vault Audience Resource	'DataLakeStore'
Azure OSSDB Audience Resource	'AzureOSSDB'
Azure Synapse Resource	'Synapse'
Azure Data Factory Resource	'ADF'

```
mssparkutils.credentials.getToken('audience Key')
```

```
mssparkutils.credentials.getToken("audience Key")
```

```
Credentials.GetToken("audience Key")
```

Validate token

Returns true if token hasn't expired.

```
mssparkutils.credentials.isValidToken('your token')
```

```
mssparkutils.credentials.isValidToken("your token")
```

```
Credentials.IsValidToken("your token")
```

Get connection string or credentials for linked service

Returns connection string or credentials for linked service.

```
mssparkutils.credentials.getConnectionStringOrCreds('linked service name')
```

```
mssparkutils.credentials.getConnectionStringOrCreds("linked service name")
```

```
Credentials.GetConnectionStringOrCreds("linked service name")
```

Get secret using workspace identity

Returns Azure Key Vault secret for a given Azure Key Vault name, secret name, and linked service name using workspace identity. Make sure you configure access to Azure Key Vault appropriately.

```
mssparkutils.credentials.getSecret('azure key vault name','secret name','linked service name')
```

```
mssparkutils.credentials.getSecret("azure key vault name","secret name","linked service name")
```

```
Credentials.GetSecret("azure key vault name","secret name","linked service name")
```

Get secret using user credentials

Returns Azure Key Vault secret for a given Azure Key Vault name, secret name, and linked service name using user credentials.

```
mssparkutils.credentials.getSecret('azure key vault name','secret name')
```

```
mssparkutils.credentials.getSecret("azure key vault name","secret name")
```

```
Credentials.GetSecret("azure key vault name","secret name")
```

Put secret using workspace identity

Puts Azure Key Vault secret for a given Azure Key Vault name, secret name, and linked service name using workspace identity. Make sure you configure the access to Azure Key Vault appropriately.

```
mssparkutils.credentials.putSecret('azure key vault name','secret name','secret value','linked service
name')
```

Put secret using workspace identity

Puts Azure Key Vault secret for a given Azure Key Vault name, secret name, and linked service name using workspace identity. Make sure you configure the access to Azure Key Vault appropriately.

```
mssparkutils.credentials.putSecret("azure key vault name","secret name","secret value","linked service
name")
```

Put secret using user credentials

Puts Azure Key Vault secret for a given Azure Key Vault name, secret name, and linked service name using user credentials.

```
mssparkutils.credentials.putSecret('azure key vault name','secret name','secret value')
```

Put secret using user credentials

Puts Azure Key Vault secret for a given Azure Key Vault name, secret name, and linked service name using user credentials.

```
mssparkutils.credentials.putSecret("azure key vault name","secret name","secret value")
```

Environment utilities

Run following commands to get an overview of the available methods:

```
mssparkutils.env.help()
```

```
mssparkutils.env.help()
```

```
Env.Help()
```

Get result:

```
GetUserName(): returns user name
GetUserId(): returns unique user id
GetJobId(): returns job id
GetWorkspaceName(): returns workspace name
GetPoolName(): returns Spark pool name
GetClusterId(): returns cluster id
```

Get user name

Returns current user name.

```
mssparkutils.env.getUserName()
```

```
mssparkutils.env.getUserName()
```

```
Env.GetUserName()
```

Get user ID

Returns current user ID.

```
mssparkutils.env.getUserId()
```

```
mssparkutils.env.getUserId()
```

```
Env.GetUserId()
```

Get job ID

Returns job ID.

```
mssparkutils.env.getJobId()
```

```
mssparkutils.env.getJobId()
```

```
Env.GetJobId()
```

Get workspace name

Returns workspace name.

```
mssparkutils.env.getWorkspaceName()
```

```
mssparkutils.env.getWorkspaceName()
```

```
Env.GetWorkspaceName()
```

Get pool name

Returns Spark pool name.

```
mssparkutils.env.getPoolName()
```

```
mssparkutils.env.getPoolName()
```

```
Env.GetPoolName()
```

Get cluster ID

Returns current cluster ID.

```
mssparkutils.env.getClusterId()
```

```
mssparkutils.env.getClusterId()
```

```
Env.GetClusterId()
```

Runtime Context

Mssparkutils runtime utils exposed 3 runtime properties, you can use the mssparkutils runtime context to get

the properties listed as below:

- **Notebookname** - The name of current notebook, will always return value for both interactive mode and pipeline mode.
- **Pipelinejobid** - The pipeline run ID, will return value in pipeline mode and return empty string in interactive mode.
- **Activityrunid** - The notebook activity run ID, will return value in pipeline mode and return empty string in interactive mode.

Currently runtime context support both Python and Scala.

```
mssparkutils.runtime.context
```

```
%%spark
mssparkutils.runtime.context
```

Session management

Stop an interactive session

Instead of manually click stop button, sometimes it's more convenient to stop an interactive session by calling an API in the code. For such cases, we provide an API `mssparkutils.session.stop()` to support stopping the interactive session via code, it's available for Scala and Python.

```
mssparkutils.session.stop()
```

```
mssparkutils.session.stop()
```

`mssparkutils.session.stop()` API will stop the current interactive session asynchronously in the background, it stops the Spark session and release resources occupied by the session so they are available to other sessions in the same pool.

> **NOTE**
> We don't recommend call language built-in APIs like `sys.exit` in Scala or `sys.exit()` in Python in your code, because such APIs just kill the interpreter process, leaving Spark session alive and resources not released.

Next steps

- Check out Synapse sample notebooks
- Quickstart: Create an Apache Spark pool in Azure Synapse Analytics using web tools
- What is Apache Spark in Azure Synapse Analytics
- Azure Synapse Analytics
- How to use file mount/unmount API in Synapse

Introduction to file mount/unmount APIs in Azure Synapse Analytics

8/19/2022 • 5 minutes to read • Edit Online

The Azure Synapse Studio team built two new mount/unmount APIs in the Microsoft Spark Utilities (`mssparkutils`) package. You can use these APIs to attach remote storage (Azure Blob Storage or Azure Data Lake Storage Gen2) to all working nodes (driver node and worker nodes). After the storage is in place, you can use the local file API to access data as if it's stored in the local file system. For more information, see Introduction to Microsoft Spark Utilities.

The article shows you how to use mount/unmount APIs in your workspace. You'll learn:

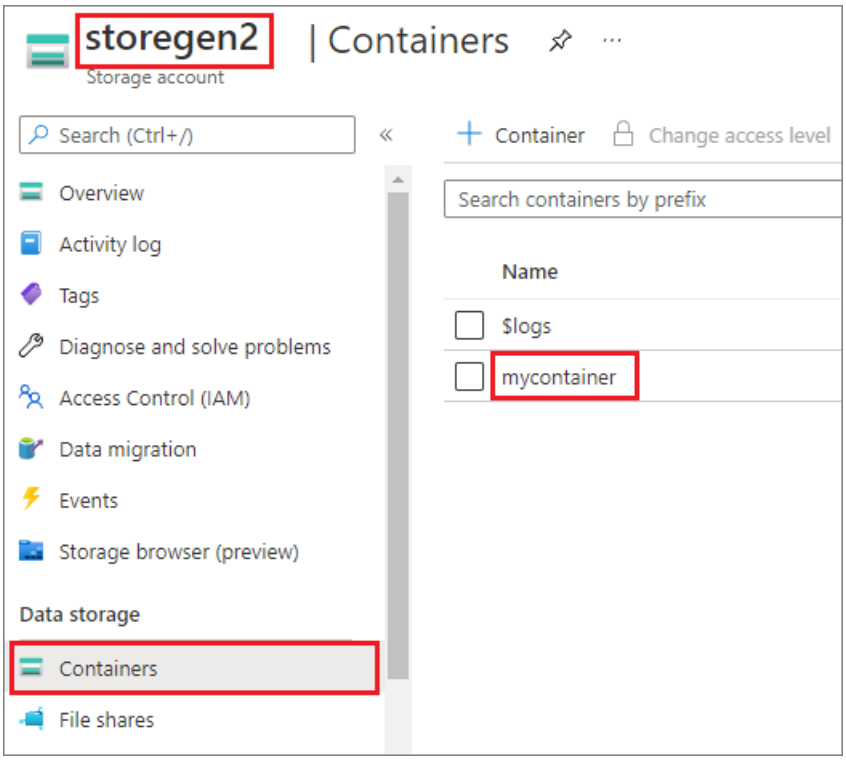
- How to mount Data Lake Storage Gen2 or Blob Storage.
- How to access files under the mount point via the local file system API.
- How to access files under the mount point by using the `mssparktuils fs` API.
- How to access files under the mount point by using the Spark read API.
- How to unmount the mount point.

> **WARNING**
>
> Azure file-share mounting is temporarily disabled. You can use Data Lake Storage Gen2 or Azure Blob Storage mounting instead, as described in the next section.
>
> Azure Data Lake Storage Gen1 storage is not supported. You can migrate to Data Lake Storage Gen2 by following the Azure Data Lake Storage Gen1 to Gen2 migration guidance before using the mount APIs.

Mount storage

This section illustrates how to mount Data Lake Storage Gen2 step by step as an example. Mounting Blob Storage works similarly.

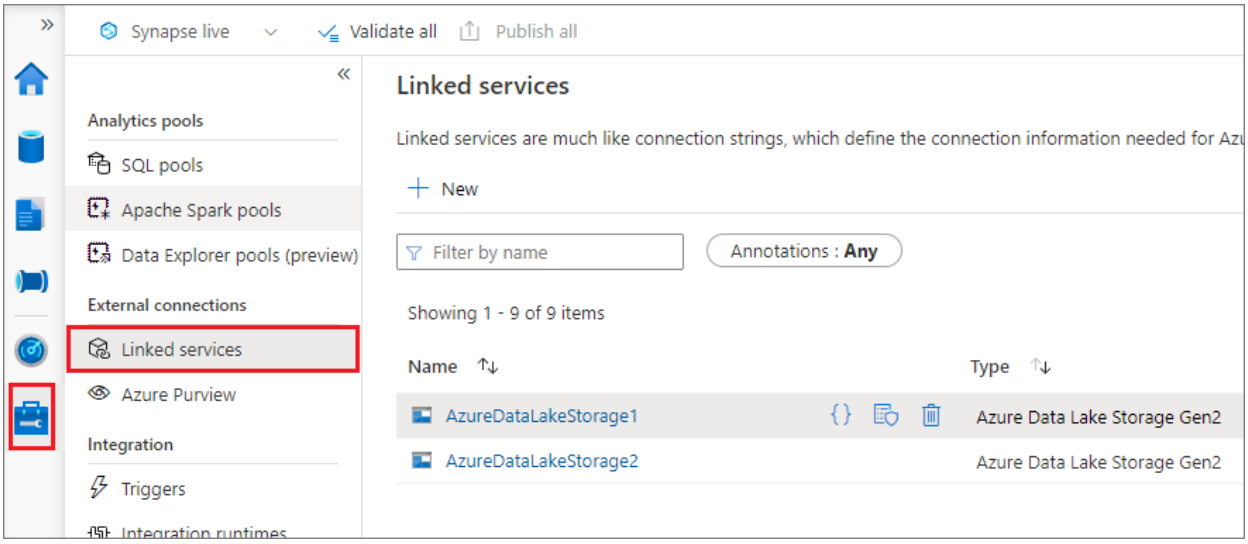
The example assumes that you have one Data Lake Storage Gen2 account named `storegen2`. The account has one container named `mycontainer` that you want to mount to `/test` in your Spark pool.



To mount the container called `mycontainer`, `mssparkutils` first needs to check whether you have the permission to access the container. Currently, Azure Synapse Analytics supports three authentication methods for the trigger mount operation: `linkedService`, `accountKey`, and `sastoken`.

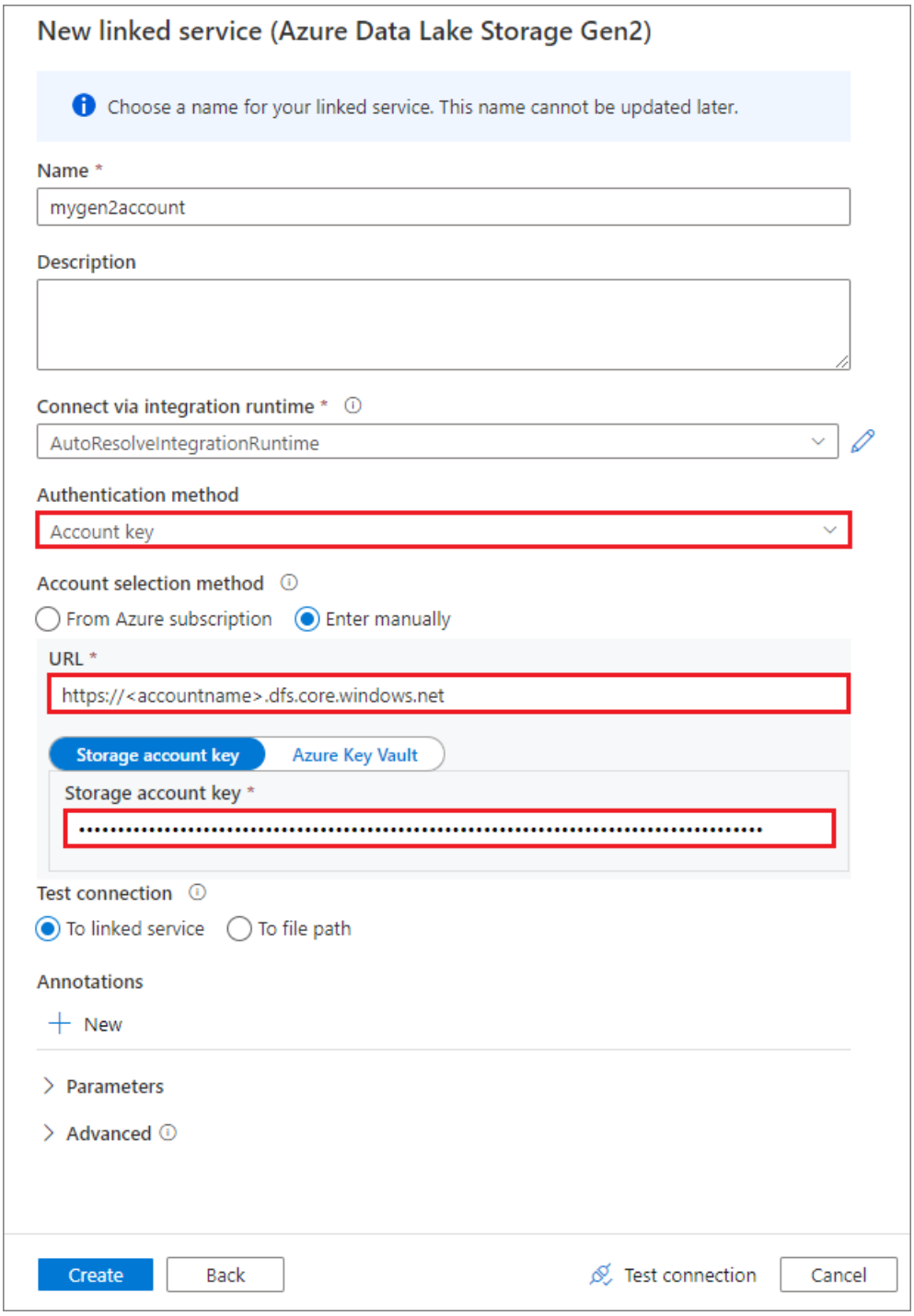
Mount by using a linked service (recommended)

We recommend a trigger mount via linked service. This method avoids security leaks, because `mssparkutils` doesn't store any secret or authentication values itself. Instead, `mssparkutils` always fetches authentication values from the linked service to request blob data from remote storage.

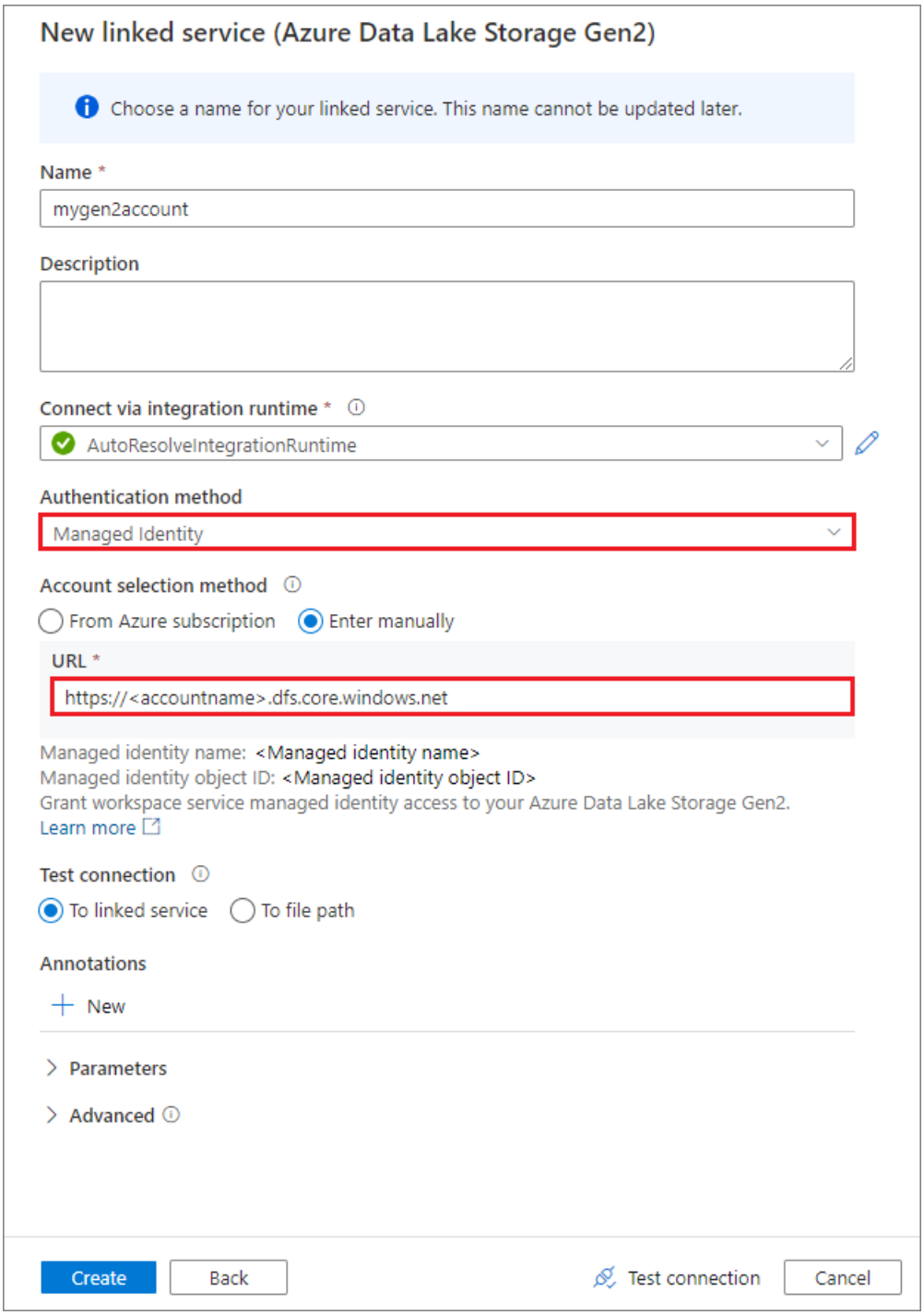


You can create a linked service for Data Lake Storage Gen2 or Blob Storage. Currently, Azure Synapse Analytics supports two authentication methods when you create a linked service:

- Create a linked service by using an account key



- Create a linked service by using a managed identity



NOTE

If you create a linked service by using a managed identity as the authentication method, make sure that the workspace MSI file has the Storage Blob Data Contributor role of the mounted container.

After you create linked service successfully, you can easily mount the container to your Spark pool by using the following Python code:

```
mssparkutils.fs.mount(
    "abfss://mycontainer@<accountname>.dfs.core.windows.net",
    "/test",
    {"linkedService":"mygen2account"}
)
```

NOTE

You might need to import `mssparkutils` if it's not available:

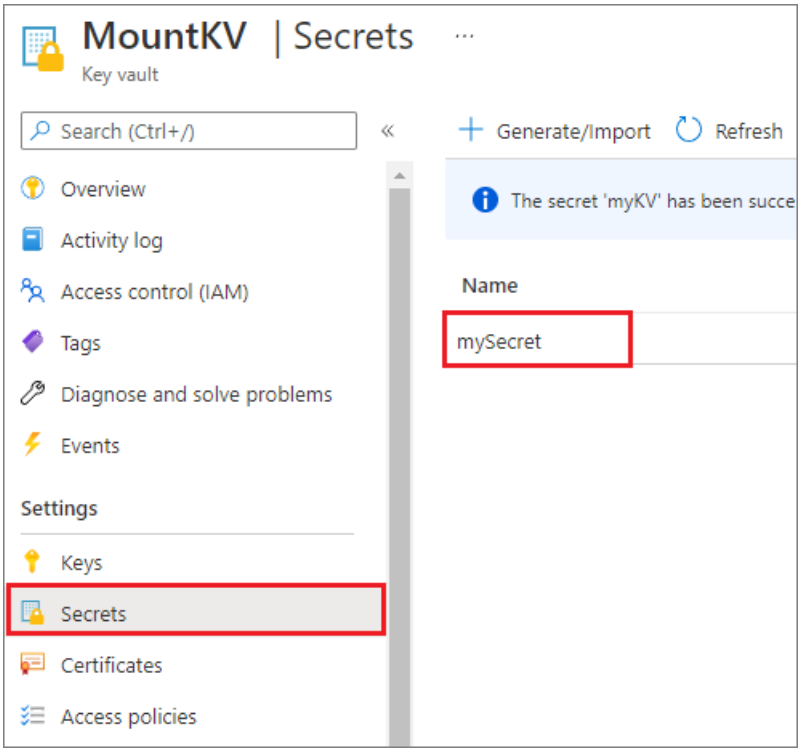
```
Fromnotebookutilsimportmssparkutils
```

We don't recommend that you mount a root folder, no matter which authentication method you use.

Mount via shared access signature token or account key

In addition to mounting through a linked service, `mssparkutils` supports explicitly passing an account key or shared access signature (SAS) token as a parameter to mount the target.

For security reasons, we recommend that you store account keys or SAS tokens in Azure Key Vault (as the following example screenshot shows). You can then retrieve them by using the `mssparkutil.credentials.getSecret` API. For more information, see Manage storage account keys with Key Vault and the Azure CLI (legacy).



Here's the sample code:

```
from notebookutils import mssparkutils

accountKey = mssparkutils.credentials.getSecret("MountKV","mySecret")
mssparkutils.fs.mount(
    "abfss://mycontainer@<accountname>.dfs.core.windows.net",
    "/test",
    {"accountKey":accountKey}
)
```

NOTE

For security reasons, don't store credentials in code.

Access files under the mount point by using the mssparktuils fs API

The main purpose of the mount operation is to let customers access the data stored in a remote storage account by using a local file system API. You can also access the data by using the `mssparkutils fs` API with a mounted path as a parameter. The path format used here is a little different.

Assume that you mounted the Data Lake Storage Gen2 container `mycontainer` to `/test` by using the mount API. When you access the data by using a local file system API, the path format is like this:

`/synfs/{jobId}/test/{filename}`

When you want to access the data by using the `mssparkutils fs` API, the path format is like this:

`synfs:/{jobId}/test/{filename}`

You can see that `synfs` is used as the schema in this case, instead of a part of the mounted path.

The following three examples show how to access a file with a mount point path by using `mssparkutils fs`. In the examples, `49` is a Spark job ID that we got from calling `mssparkutils.env.getJobId()`.

- List directories:

  ```
  mssparkutils.fs.ls("synfs:/49/test")
  ```

- Read file content:

  ```
  mssparkutils.fs.head("synfs:/49/test/myFile.txt")
  ```

- Create a directory:

  ```
  mssparkutils.fs.mkdirs("synfs:/49/test/newdir")
  ```

Access files under the mount point by using the Spark read API

You can provide a parameter to access the data through the Spark read API. The path format here is the same when you use the `mssparkutils fs` API:

`synfs:/{jobId}/test/{filename}`

Read a file from a mounted Data Lake Storage Gen2 storage account

The following example assumes that a Data Lake Storage Gen2 storage account was already mounted, and then you read the file by using a mount path:

```
%%pyspark

df = spark.read.load("synfs:/49/test/myFile.csv", format='csv')
df.show()
```

Read a file from a mounted Blob Storage account

If you mounted a Blob Storage account and want to access it by using `mssparkutils` or the Spark API, you need to explicitly configure the SAS token via Spark configuration before you try to mount the container by using the mount API:

1. To access a Blob Storage account by using `mssparkutils` or the Spark API after a trigger mount, update the Spark configuration as shown in the following code example. You can bypass this step if you want to access the Spark configuration only by using the local file API after mounting.

```
blob_sas_token = mssparkutils.credentials.getConnectionStringOrCreds("myblobstorageaccount")

spark.conf.set('fs.azure.sas.mycontainer.<blobStorageAccountName>.blob.core.windows.net',
blob_sas_token)
```

2. Create the linked service `myblobstorageaccount`, and mount the Blob Storage account by using the linked service:

```
%%spark
mssparkutils.fs.mount(
    "wasbs://mycontainer@<blobStorageAccountName>.blob.core.windows.net",
    "/test",
    Map("linkedService" -> "myblobstorageaccount")
)
```

3. Mount the Blob Storage container, and then read the file by using a mount path through the local file API:

```
    # mount the Blob Storage container, and then read the file by using a mount path
    with open("/synfs/64/test/myFile.txt") as f:
    print(f.read())
```

4. Read the data from the mounted Blob Storage container through the Spark read API:

```
%%spark
// mount blob storage container and then read file using mount path
val df = spark.read.text("synfs:/49/test/myFile.txt")
df.show()
```

Unmount the mount point

Use the following code to unmount your mount point (`/test` in this example):

```
mssparkutils.fs.unmount("/test")
```

Known limitations

- The `mssparkutils fs help` function hasn't added the description about the mount/unmount part yet.
- The unmount mechanism is not automatic. When the application run finishes, to unmount the mount point to release the disk space, you need to explicitly call an unmount API in your code. Otherwise, the mount point will still exist in the node after the application run finishes.
- Mounting a Data Lake Storage Gen1 storage account is not supported for now.

Next steps

- Get started with Azure Synapse Analytics
- Monitor your Synapse workspace
- Introduction to Microsoft Spark Utilities

System views supported in Synapse SQL

8/19/2022 • 2 minutes to read • Edit Online

Links to the documentation for T-SQL statements supported in Synapse SQL.

> **NOTE**
> Synapse serverless SQL pool supports SQL Server catalog views only.

Dedicated SQL pool and serverless SQL pool catalog views

- sys.pdw_column_distribution_properties
- sys.pdw_distributions
- sys.pdw_index_mappings
- sys.pdw_loader_backup_run_details
- sys.pdw_loader_backup_runs
- sys.pdw_materialized_view_column_distribution_properties (Preview)
- sys.pdw_materialized_view_distribution_properties (Preview)
- sys.pdw_materialized_view_mappings (Preview)
- sys.pdw_nodes_column_store_dictionaries
- sys.pdw_nodes_column_store_row_groups
- sys.pdw_nodes_column_store_segments
- sys.pdw_nodes_columns
- sys.pdw_nodes_indexes
- sys.pdw_nodes_partitions
- sys.pdw_nodes_pdw_physical_databases
- sys.pdw_nodes_tables
- sys.pdw_permanent_table_mappings
- sys.pdw_replicated_table_cache_state
- sys.pdw_table_distribution_properties
- sys.pdw_table_mappings
- sys.resource_governor_workload_groups
- sys.workload_management_workload_classifier_details (Preview)
- sys.workload_management_workload_classifiers (Preview)

Dedicated SQL pool dynamic management views (DMVs)

- sys.dm_pdw_dms_cores
- sys.dm_pdw_dms_external_work
- sys.dm_pdw_dms_workers
- sys.dm_pdw_errors
- sys.dm_pdw_exec_connections
- sys.dm_pdw_exec_requests
- sys.dm_pdw_exec_sessions
- sys.dm_pdw_hadoop_operations

- sys.dm_pdw_lock_waits
- sys.dm_pdw_nodes
- sys.dm_pdw_nodes_database_encryption_keys
- sys.dm_pdw_os_threads
- sys.dm_pdw_request_steps
- sys.dm_pdw_resource_waits
- sys.dm_pdw_sql_requests
- sys.dm_pdw_sys_info
- sys.dm_pdw_wait_stats
- sys.dm_pdw_waits
- sys.dm_pdw_waits

SQL Server DMVs applicable to dedicated SQL pool

The following DMVs are applicable to dedicated SQL pool, but must be executed by connecting to the **master** database.

- sys.database_service_objectives
- sys.dm_operation_status
- sys.fn_helpcollations()

SQL Server catalog views

- sys.all_columns
- sys.all_objects
- sys.all_parameters
- sys.all_sql_modules
- sys.all_views
- sys.assemblies
- sys.assembly_modules
- sys.assembly_types
- sys.certificates
- sys.check_constraints
- sys.columns
- sys.computed_columns
- sys.credentials
- sys.data_spaces
- sys.database_credentials
- sys.database_files
- sys.database_permissions
- sys.database_principals
- sys.database_query_store_options
- sys.database_role_members
- sys.databases
- sys.default_constraints
- sys.external_data_sources
- sys.external_file_formats
- sys.external_tables

- sys.filegroups
- sys.foreign_key_columns
- sys.foreign_keys
- sys.identity_columns
- sys.index_columns
- sys.indexes
- sys.key_constraints
- sys.numbered_procedures
- sys.objects
- sys.parameters
- sys.partition_functions
- sys.partition_parameters
- sys.partition_range_values
- sys.partition_schemes
- sys.partitions
- sys.procedures
- sys.query_context_settings
- sys.query_store_plan
- sys.query_store_query
- sys.query_store_query_text
- sys.query_store_runtime_stats
- sys.query_store_runtime_stats_interval
- sys.schemas
- sys.securable_classes
- sys.sql_expression_dependencies
- sys.sql_modules
- sys.stats
- sys.stats_columns
- sys.symmetric_keys
- sys.synonyms
- sys.syscharsets
- sys.syscolumns
- sys.sysdatabases
- sys.syslanguages
- sys.sysobjects
- sys.sysreferences
- sys.system_columns
- sys.system_objects
- sys.system_parameters
- sys.system_sql_modules
- sys.system_views
- sys.systypes
- sys.sysusers
- sys.tables
- sys.types
- sys.views

SQL Server DMVs available in dedicated SQL pool

SQL pool exposes many of the SQL Server dynamic management views (DMVs). These views, when queried in dedicated SQL pool, are reporting the state of SQL Databases running on the distributions.

SQL pool and Analytics Platform System's Parallel Data Warehouse (PDW) use the same system views. Each DMV has a column called pdw_node_id, which is the identifier for the Compute node.

> **NOTE**
> To use these views, insert 'pdw_nodes_' into the name, as shown in the following table:

DMV NAME IN DEDICATED SQL POOL	SQL SERVER TRANSACT-SQL ARTICLE
sys.dm_pdw_nodes_db_column_store_row_group_physical_stats	sys.dm_db_column_store_row_group_physical_stats
sys.dm_pdw_nodes_db_column_store_row_group_operational_stats	sys.dm_db_column_store_row_group_operational_stats
sys.dm_pdw_nodes_db_file_space_usage	sys.dm_db_file_space_usage
sys.dm_pdw_nodes_db_index_usage_stats	sys.dm_db_index_usage_stats
sys.dm_pdw_nodes_db_partition_stats	sys.dm_db_partition_stats
sys.dm_pdw_nodes_db_session_space_usage	sys.dm_db_session_space_usage
sys.dm_pdw_nodes_db_task_space_usage	sys.dm_db_task_space_usage
sys.dm_pdw_nodes_exec_background_job_queue	sys.dm_exec_background_job_queue
sys.dm_pdw_nodes_exec_background_job_queue_stats	sys.dm_exec_background_job_queue_stats
sys.dm_pdw_nodes_exec_cached_plans	sys.dm_exec_cached_plans
sys.dm_pdw_nodes_exec_connections	sys.dm_exec_connections
sys.dm_pdw_nodes_exec_procedure_stats	sys.dm_exec_procedure_stats
sys.dm_pdw_nodes_exec_query_memory_grants	sys.dm_exec_query_memory_grants
sys.dm_pdw_nodes_exec_query_optimizer_info	sys.dm_exec_query_optimizer_info
sys.dm_pdw_nodes_exec_query_resource_semaphores	sys.dm_exec_query_resource_semaphores
sys.dm_pdw_nodes_exec_query_stats	sys.dm_exec_query_stats
sys.dm_pdw_nodes_exec_requests	sys.dm_exec_requests
sys.dm_pdw_nodes_exec_sessions	sys.dm_exec_sessions
sys.dm_pdw_nodes_io_pending_io_requests	sys.dm_io_pending_io_requests

DMV NAME IN DEDICATED SQL POOL	SQL SERVER TRANSACT-SQL ARTICLE
sys.dm_pdw_nodes_io_virtual_file_stats	sys.dm_io_virtual_file_stats
sys.dm_pdw_nodes_os_buffer_descriptors	sys.dm_os_buffer_descriptors
sys.dm_pdw_nodes_os_child_instances	sys.dm_os_child_instances
sys.dm_pdw_nodes_os_cluster_nodes	sys.dm_os_cluster_nodes
sys.dm_pdw_nodes_os_dispatcher_pools	sys.dm_os_dispatcher_pools
sys.dm_pdw_nodes_os_dispatchers	Transact-SQL Documentation is not available.
sys.dm_pdw_nodes_os_hosts	sys.dm_os_hosts
sys.dm_pdw_nodes_os_latch_stats	sys.dm_os_latch_stats
sys.dm_pdw_nodes_os_memory_brokers	sys.dm_os_memory_brokers
sys.dm_pdw_nodes_os_memory_cache_clock_hands	sys.dm_os_memory_cache_clock_hands
sys.dm_pdw_nodes_os_memory_cache_counters	sys.dm_os_memory_cache_counters
sys.dm_pdw_nodes_os_memory_cache_entries	sys.dm_os_memory_cache_entries
sys.dm_pdw_nodes_os_memory_cache_hash_tables	sys.dm_os_memory_cache_hash_tables
sys.dm_pdw_nodes_os_memory_clerks	sys.dm_os_memory_clerks
sys.dm_pdw_nodes_os_memory_node_access_stats	Transact-SQL Documentation is not available.
sys.dm_pdw_nodes_os_memory_nodes	sys.dm_os_memory_nodes
sys.dm_pdw_nodes_os_memory_objects	sys.dm_os_memory_objects
sys.dm_pdw_nodes_os_memory_pools	sys.dm_os_memory_pools
sys.dm_pdw_nodes_os_nodes	sys.dm_os_nodes
sys.dm_pdw_nodes_os_performance_counters	sys.dm_os_performance_counters
sys.dm_pdw_nodes_os_process_memory	sys.dm_os_process_memory
sys.dm_pdw_nodes_os_schedulers	sys.dm_os_schedulers
sys.dm_pdw_nodes_os_spinlock_stats	Transact-SQL Documentation is not available.
sys.dm_pdw_nodes_os_sys_info	sys.dm_os_sys_info
sys.dm_pdw_nodes_os_sys_memory	sys.dm_os_memory_nodes

DMV NAME IN DEDICATED SQL POOL	SQL SERVER TRANSACT-SQL ARTICLE
sys.dm_pdw_nodes_os_tasks	sys.dm_os_tasks
sys.dm_pdw_nodes_os_threads	sys.dm_os_threads
sys.dm_pdw_nodes_os_virtual_address_dump	sys.dm_os_virtual_address_dump
sys.dm_pdw_nodes_os_wait_stats	sys.dm_os_wait_stats
sys.dm_pdw_nodes_os_waiting_tasks	sys.dm_os_waiting_tasks
sys.dm_pdw_nodes_os_workers	sys.dm_os_workers
sys.dm_pdw_nodes_tran_active_snapshot_database_transactions	sys.dm_tran_active_snapshot_database_transactions
sys.dm_pdw_nodes_tran_active_transactions	sys.dm_tran_active_transactions
sys.dm_pdw_nodes_tran_commit_table	sys.dm_tran_commit_table
sys.dm_pdw_nodes_tran_current_snapshot	sys.dm_tran_current_snapshot
sys.dm_pdw_nodes_tran_current_transaction	sys.dm_tran_current_transaction
sys.dm_pdw_nodes_tran_database_transactions	sys.dm_tran_database_transactions
sys.dm_pdw_nodes_tran_locks	sys.dm_tran_locks
sys.dm_pdw_nodes_tran_session_transactions	sys.dm_tran_session_transactions
sys.dm_pdw_nodes_tran_top_version_generators	sys.dm_tran_top_version_generators

SQL Server 2016 PolyBase DMVs available in dedicated SQL pool

The following DMVs are applicable to dedicated SQL pool, but must be executed by connecting to the **master** database.

- sys.dm_exec_compute_node_errors
- sys.dm_exec_compute_node_status
- sys.dm_exec_compute_nodes
- sys.dm_exec_distributed_request_steps
- sys.dm_exec_distributed_requests
- sys.dm_exec_distributed_sql_requests
- sys.dm_exec_dms_services
- sys.dm_exec_dms_workers
- sys.dm_exec_external_operations
- sys.dm_exec_external_work

SQL Server INFORMATION_SCHEMA views

- CHECK_CONSTRAINTS

- COLUMNS
- PARAMETERS
- ROUTINES
- SCHEMATA
- TABLES
- VIEW_COLUMN_USAGE
- VIEW_TABLE_USAGE
- VIEWS

Next steps

For more reference information, see T-SQL statements in Synapse SQL, and T-SQL language elements in Synapse SQL.

Azure Policy built-in definitions for Azure Synapse Analytics

8/19/2022 • 4 minutes to read • Edit Online

This page is an index of Azure Policy built-in policy definitions for Azure Synapse. For additional Azure Policy built-ins for other services, see Azure Policy built-in definitions.

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Version** column to view the source on the Azure Policy GitHub repo.

Azure Synapse

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
Auditing on Synapse workspace should be enabled	Auditing on your Synapse workspace should be enabled to track database activities across all databases on the dedicated SQL pools and save them in an audit log.	AuditIfNotExists, Disabled	1.0.0
Azure Synapse Workspace SQL Server should be running TLS version 1.2 or newer	Setting TLS version to 1.2 or newer improves security by ensuring your Azure Synapse workspace SQL server can only be accessed from clients using TLS 1.2 or newer. Using versions of TLS less than 1.2 is not recommended since they have well documented security vulnerabilities.	Audit, Deny, Disabled	1.0.0
Azure Synapse workspaces should allow outbound data traffic only to approved targets	Increase security of your Synapse workspace by allowing outbound data traffic only to approved targets. This helps prevention against data exfiltration by validating the target before sending data.	Audit, Disabled, Deny	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
Azure Synapse workspaces should disable public network access	Disabling public network access improves security by ensuring that the Synapse workspace isn't exposed on the public internet. Creating private endpoints can limit exposure of your Synapse workspaces. Learn more at: https://docs.microsoft.com/azure/synapse-analytics/security/connectivity-settings.	Audit, Deny, Disabled	1.0.0
Azure Synapse workspaces should use customer-managed keys to encrypt data at rest	Use customer-managed keys to control the encryption at rest of the data stored in Azure Synapse workspaces. Customer-managed keys deliver double encryption by adding a second layer of encryption on top of the default encryption with service-managed keys.	Audit, Deny, Disabled	1.0.0
Azure Synapse workspaces should use private link	Azure Private Link lets you connect your virtual network to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network. By mapping private endpoints to Azure Synapse workspace, data leakage risks are reduced. Learn more about private links at: https://docs.microsoft.com/azure/synapse-analytics/security/how-to-connect-to-workspace-with-private-links.	Audit, Disabled	1.0.1

NAME	DESCRIPTION	EFFECT(S)	VERSION
Configure Azure Synapse Workspace Dedicated SQL minimum TLS version	Customers can raise or lower the minimal TLS version using the API, for both new Synapse workspaces or existing workspaces. So users who need to use a lower client version in the workspaces can connect while users who has security requirement can raise the minimum TLS version. Learn more at: https://docs.microsoft.com/azure/synapse-analytics/security/connectivity-settings.	Modify, Disabled	1.0.0
Configure Azure Synapse workspaces to disable public network access	Disable public network access for your Synapse workspace so that it is not accessible over the public internet. This can reduce data leakage risks. Learn more at: https://docs.microsoft.com/azure/synapse-analytics/security/connectivity-settings.	Modify, Disabled	1.0.0
Configure Azure Synapse workspaces with private endpoints	Private endpoints connect your virtual network to Azure services without a public IP address at the source or destination. By mapping private endpoints to Azure Synapse workspaces, you can reduce data leakage risks. Learn more about private links at: https://docs.microsoft.com/azure/synapse-analytics/security/how-to-connect-to-workspace-with-private-links.	DeployIfNotExists, Disabled	1.0.0
Configure Synapse workspaces to have auditing enabled	To ensure the operations performed against your SQL assets are captured, Synapse workspaces should have auditing enabled. This is sometimes required for compliance with regulatory standards.	DeployIfNotExists, Disabled	2.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
Configure Synapse workspaces to have auditing enabled to Log Analytics workspace	To ensure the operations performed against your SQL assets are captured, Synapse workspaces should have auditing enabled. If auditing is not enabled, this policy will configure auditing events to flow to the specified Log Analytics workspace.	DeployIfNotExists, Disabled	1.0.0
IP firewall rules on Azure Synapse workspaces should be removed	Removing all IP firewall rules improves security by ensuring your Azure Synapse workspace can only be accessed from a private endpoint. This configuration audits creation of firewall rules that allow public network access on the workspace.	Audit, Disabled	1.0.0
Managed workspace virtual network on Azure Synapse workspaces should be enabled	Enabling a managed workspace virtual network ensures that your workspace is network isolated from other workspaces. Data integration and Spark resources deployed in this virtual network also provides user level isolation for Spark activities.	Audit, Deny, Disabled	1.0.0
Synapse managed private endpoints should only connect to resources in approved Azure Active Directory tenants	Protect your Synapse workspace by only allowing connections to resources in approved Azure Active Directory (Azure AD) tenants. The approved Azure AD tenants can be defined during policy assignment.	Audit, Disabled, Deny	1.0.0
Synapse workspace auditing settings should have action groups configured to capture critical activities	To ensure your audit logs are as thorough as possible, the AuditActionsAndGroups property should include all the relevant groups. We recommend adding at least SUCCESSFUL_DATABASE_AUTHENTICATION_GROUP, FAILED_DATABASE_AUTHENTICATION_GROUP, and BATCH_COMPLETED_GROUP. This is sometimes required for compliance with regulatory standards.	AuditIfNotExists, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
Synapse workspaces with SQL auditing to storage account destination should be configured with 90 days retention or higher	For incident investigation purposes, we recommend setting the data retention for your Synapse workspace' SQL auditing to storage account destination to at least 90 days. Confirm that you are meeting the necessary retention rules for the regions in which you are operating. This is sometimes required for compliance with regulatory standards.	AuditIfNotExists, Disabled	2.0.0
Vulnerability assessment should be enabled on your Synapse workspaces	Discover, track, and remediate potential vulnerabilities by configuring recurring SQL vulnerability assessment scans on your Synapse workspaces.	AuditIfNotExists, Disabled	1.0.0

Next steps

- See the built-ins on the Azure Policy GitHub repo.
- Review the Azure Policy definition structure.
- Review Understanding policy effects.

T-SQL language elements for dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

Links to the documentation for T-SQL language elements supported in dedicated SQL pool.

Core elements

- syntax conventions
- object naming rules
- reserved keywords
- collations
- comments
- constants
- data types
- EXECUTE
- expressions
- KILL
- IDENTITY property workaround
- PRINT
- USE

Batches, control-of-flow, and variables

- BEGIN...END
- BREAK
- DECLARE @local_variable
- IF...ELSE
- RAISERROR
- SET@local_variable
- THROW
- TRY...CATCH
- WHILE

Operators

- + (Add)
- + (String Concatenation)
- - (Negative)
- - (Subtract)
- * (Multiply)
- / (Divide)
- Modulo

Wildcard character(s) to match

- = (Equals)
- > (Greater than)
- < (Less than)
- >= (Great than or equal to)
- <= (Less than or equal to)
- <> (Not equal to)
- != (Not equal to)
- AND
- BETWEEN
- EXISTS
- IN
- IS [NOT] NULL
- LIKE
- NOT
- OR

Bitwise operators

- & (Bitwise AND)
- | (Bitwise OR)
- ^ (Bitwise exclusive OR)
- ~ (Bitwise NOT)
- ^= (Bitwise Exclusive OR EQUALS)
- |= (Bitwise OR EQUALS)
- &= (Bitwise AND EQUALS)

Functions

- @@DATEFIRST
- @@ERROR
- @@LANGUAGE
- @@SPID
- @@TRANCOUNT
- @@VERSION
- ABS
- ACOS
- ASCII
- ASIN
- ATAN
- ATN2
- BINARY_CHECKSUM
- CASE
- CAST and CONVERT
- CEILING
- CHAR
- CHARINDEX
- CHECKSUM
- COALESCE
- COL_NAME

- COLLATIONPROPERTY
- CONCAT
- COS
- COT
- COUNT
- CONCAT_WS
- COUNT_BIG
- CUME_DIST
- CURRENT_TIMESTAMP
- CURRENT_USER
- DATABASEPROPERTYEX
- DATALENGTH
- DATEADD
- DATEDIFF
- DATEFROMPARTS
- DATENAME
- DATEPART
- DATETIME2FROMPARTS
- DATETIMEFROMPARTS
- DATETIMEOFFSETFROMPARTS
- DAY
- DB_ID
- DB_NAME
- DEGREES
- DENSE_RANK
- DIFFERENCE
- EOMONTH
- ERROR_MESSAGE
- ERROR_NUMBER
- ERROR_PROCEDURE
- ERROR_SEVERITY
- ERROR_STATE
- EXP
- FIRST_VALUE
- FLOOR
- GETDATE
- GETUTCDATE
- HAS_DBACCESS
- HASHBYTES
- INDEXPROPERTY
- ISDATE
- ISNULL
- ISNUMERIC
- LAG
- LAST_VALUE
- LEAD

- LEFT
- LEN
- LOG
- LOG10
- LOWER
- LTRIM
- MAX
- MIN
- MONTH
- NCHAR
- NTILE
- NULLIF
- OBJECT_ID
- OBJECT_NAME
- OBJECTPROPERTY
- OIBJECTPROPERTYEX
- ODBCS scalar functions
- OVER clause
- PARSENAME
- PATINDEX
- PERCENTILE_CONT
- PERCENTILE_DISC
- PERCENT_RANK
- PI
- POWER
- QUOTENAME
- RADIANS
- RAND
- RANK
- REPLACE
- REPLICATE
- REVERSE
- RIGHT
- ROUND
- ROW_NUMBER
- RTRIM
- SCHEMA_ID
- SCHEMA_NAME
- SERVERPROPERTY
- SESSION_USER
- SIGN
- SIN
- SMALLDATETIMEFROMPARTS
- SOUNDEX
- SPACE
- SQL_VARIANT_PROPERTY

- SQRT
- SQUARE
- STATS_DATE
- STDEV
- STDEVP
- STR
- STRING_AGG
- STRING_SPLIT
- STUFF
- SUBSTRING
- SUM
- SUSER_SNAME
- SWITCHOFFSET
- SYSDATETIME
- SYSDATETIMEOFFSET
- SYSTEM_USER
- SYSUTCDATETIME
- TAN
- TERTIARY_WEIGHTS
- TIMEFROMPARTS
- TRIM
- TODATETIMEOFFSET
- TYPE_ID
- TYPE_NAME
- TYPEPROPERTY
- UNICODE
- UPPER
- USER
- USER_NAME
- VAR
- VARP
- YEAR
- XACT_STATE

Transactions

- Transactions

Diagnostic sessions

- CREATE DIAGNOSTICS SESSION

Procedures

- sp_addrolemember
- sp_columns
- sp_configure
- sp_datatype_info_90

- sp_droprolemember
- sp_execute
- sp_executesql
- sp_fkeys
- sp_pdw_add_network_credentials
- sp_pdw_database_encryption
- sp_pdw_database_encryption_regenerate_system_keys
- sp_pdw_log_user_data_masking
- sp_pdw_remove_network_credentials
- sp_pkeys
- sp_prepare
- sp_rename (preview)
- sp_spaceused
- sp_special_columns_100
- sp_sproc_columns
- sp_statistics
- sp_tables
- sp_unprepare

SET statements

- SET ANSI_DEFAULTS
- SET ANSI_NULL_DFLT_OFF
- SET ANSI_NULL_DFLT_ON
- SET ANSI_NULLS
- SET ANSI_PADDING
- SET ANSI_WARNINGS
- SET ARITHABORT
- SET ARITHIGNORE
- SET CONCAT_NULL_YIELDS_NULL
- SET DATEFIRST
- SET DATEFORMAT
- SET FMTONLY
- SET IMPLICIT_TRANSACITONS
- SET LOCK_TIMEOUT
- SET NUMBERIC_ROUNDABORT
- SET QUOTED_IDENTIFIER
- SET ROWCOUNT
- SET TEXTSIZE
- SET TRANSACTION ISOLATION LEVEL
- SET XACT_ABORT

Next steps

For more reference information, see T-SQL statements in dedicated SQL pool, and System views in dedicated SQL pool.

T-SQL statements supported for dedicated SQL pool in Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

Links to the documentation for T-SQL statements supported in dedicated SQL pool.

Data Definition Language (DDL) statements

- ALTER DATABASE
- ALTER INDEX
- ALTER MATERIALIZED VIEW (preview)
- ALTER PROCEDURE
- ALTER SCHEMA
- ALTER TABLE
- CREATE COLUMNSTORE INDEX
- CREATE DATABASE
- CREATE DATABASE SCOPED CREDENTIAL
- CREATE EXTERNAL DATA SOURCE
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE
- CREATE FUNCTION
- CREATE INDEX
- CREATE MATERIALIZED VIEW AS SELECT (preview)
- CREATE PROCEDURE
- CREATE SCHEMA
- CREATE STATISTICS
- CREATE TABLE
- CREATE TABLE AS SELECT
- CREATE VIEW
- CREATE WORKLOAD CLASSIFIER
- DROP EXTERNAL DATA SOURCE
- DROP EXTERNAL FILE FORMAT
- DROP EXTERNAL TABLE
- DROP INDEX
- DROP PROCEDURE
- DROP STATISTICS
- DROP TABLE
- DROP SCHEMA
- DROP VIEW
- DROP WORKLOAD CLASSIFIER
- RENAME
- SET RESULT_SET_CACHING
- TRUNCATE TABLE
- UPDATE STATISTICS

Data Manipulation Language (DML) statements

- DELETE
- INSERT
- UPDATE

Database Console Commands

- DBCC DROPCLEANBUFFERS
- DBCC DROPRESULTSETCACHE (preview)
- DBCC FREEPROCCACHE
- DBCC SHRINKLOG
- DBCC PDW_SHOWEXECUTIONPLAN
- DBCC PDW_SHOWMATERIALIZEDVIEWOVERHEAD
- DBCC SHOWRESULTCACHESPACEUSED (preview)
- DBCC PDW_SHOWPARTITIONSTATS
- DBCC PDW_SHOWSPACEUSED
- DBCC SHOW_STATISTICS

Query statements

- SELECT
- WITH common_table_expression
- EXCEPT and INTERSECT
- EXPLAIN
- FROM
- Using PIVOT and UNPIVOT
- GROUP BY
- HAVING
- ORDER BY
- OPTION
- UNION
- WHERE
- TOP
- Aliasing
- Search condition
- Subqueries

Security statements

- Permissions:
  - GRANT
  - DENY
  - REVOKE
- ALTER AUTHORIZATION
- ALTER CERTIFICATE (preview)
- ALTER DATABASE ENCRYPTION KEY (preview)

- ALTER LOGIN
- ALTER MASTER KEY
- ALTER ROLE
- ALTER USER
- BACKUP CERTIFICATE (preview)
- CLOSE MASTER KEY
- CREATE CERTIFICATE (preview)
- CREATE DATABASE ENCRYPTION KEY (preview)
- CREATE LOGIN
- CREATE MASTER KEY
- CREATE ROLE
- CREATE USER
- DROP CERTIFICATE (preview)
- DROP DATABASE ENCRYPTION KEY
- DROP LOGIN
- DROP MASTER KEY
- DROP ROLE
- DROP USER
- OPEN MASTER KEY

Next steps

For more reference information, see T-SQL language elements in dedicated SQL pool, and System views in dedicated SQL pool.

Previous monthly updates in Azure Synapse Analytics

8/19/2022 • 22 minutes to read • Edit Online

This article describes previous month updates to Azure Synapse Analytics. For the most current month's release, check out Azure Synapse Analytics latest updates. Each update links to the Azure Synapse Analytics blog and an article that provides more information.

May 2022 update

The following updates are new to Azure Synapse Analytics this month.

General

Get connected with the new Azure Synapse Influencer program! Join a community of Azure Synapse Influencers who are helping each other achieve more with cloud analytics! The Azure Synapse Influencer program recognizes Azure Synapse Analytics users and advocates who actively support the community by sharing Synapse-related content, announcements, and product news via social media.

SQL

- **Data Warehouse Migration guide for Dedicated SQL Pools in Azure Synapse Analytics** - With the benefits that cloud migration offers, we hear that you often look for steps, processes, or guidelines to follow for quick and easy migrations from existing data warehouse environments. We just released a set of Data Warehouse migration guides to make your transition to dedicated SQL Pools in Azure Synapse Analytics easier.

- **Automatic character column length calculation** - It's no longer necessary to define character column lengths! Serverless SQL pools let you query files in the data lake without knowing the schema upfront. The best practice was to specify the lengths of character columns to get optimal performance. Not anymore! With this new feature, you can get optimal query performance without having to define the schema. The serverless SQL pool will calculate the average column length for each inferred character column or character column defined as larger than 100 bytes. The schema will stay the same, while the serverless SQL pool will use the calculated average column lengths internally. It will also automatically calculate the cardinality estimation in case there was no previously created statistic.

Apache Spark for Synapse

- **Azure Synapse Dedicated SQL Pool Connector for Apache Spark Now Available in Python** - Previously, the Azure Synapse Dedicated SQL Pool connector was only available using Scala. Now, it can be used with Python on Spark 3. The only difference between the Scala and Python implementations is the optional Scala callback handle, which allows you to receive post-write metrics.

  The following are now supported in Python on Spark 3:

  - Read using Azure Active Directory (AD) Authentication or Basic Authentication
  - Write to Internal Table using Azure AD Authentication or Basic Authentication
  - Write to External Table using Azure AD Authentication or Basic Authentication

  To learn more about the connector in Python, read Azure Synapse Dedicated SQL Pool Connector for Apache Spark.

- **Manage Azure Synapse Apache Spark configuration** - Apache Spark configuration management is always a challenging task because Spark has hundreds of properties. It is also challenging for you to

know the optimal value for Spark configurations. With the new Spark configuration management feature, you can create a standalone Spark configuration artifact with auto-suggestions and built-in validation rules. The Spark configuration artifact allows you to share your Spark configuration within and across Azure Synapse workspaces. You can also easily associate your Spark configuration with a Spark pool, a Notebook, and a Spark job definition for reuse and minimize the need to copy the Spark configuration in multiple places. To learn more about the new Spark configuration management feature, read Manage Apache Spark configuration.

Synapse Data Explorer

- **Synapse Data Explorer live query in Excel** - Using the new Data Explorerweb experience Open in Excel feature, you can now provide access to live results of your query by sharing the connected Excel Workbook with colleagues and team members.You can open the live query in an Excel Workbook and refresh it directly from Excel to get the most up to date query results. To learn more about Excel live query, read Open live query in Excel.

- **Use Managed Identities for External SQL Server Tables** - One of the key benefits of Azure Synapse is the ability to bring together data integration, enterprise data warehousing, and big data analytics. With Managed Identity support, Synapse Data Explorer table definition is now simpler and more secure. You can now use managed identities instead of entering in your credentials.

  An external SQL table is a schema entity that references data stored outside the Synapse Data Explorer database. Using the Create and alter SQL Server external tables command, External SQL tables can easily be added to the Synapse Data Explorer database schema.

  To learn more about managed identities, read Managed identities overview.

  To learn more about external tables, read Create and alter SQL Server external tables.

- **New KQL Learn module (2 out of 3) is live!** - The power of Kusto Query Language (KQL) is its simplicity to query structured, semi-structured, and unstructured data together. To make it easier for you to learn KQL, we are releasing Learn modules. Previously, we released Write your first query with Kusto Query Language. New this month is Gain insights from your data by using Kusto Query Language.

  KQL is the query language used to query Synapse Data Explorer big data. KQL has a fast-growing user community, with hundreds of thousands of developers, data engineers, data analysts, and students.

  Check out the newest KQL Learn Model and see for yourself how easy it is to become a KQL master.

  To learn more about KQL, read Kusto Query Language (KQL) overview.

- **Azure Synapse Data Explorer connector for Microsoft Power Automate, Logic Apps, and Power Apps [Generally Available]** - The Azure Data Explorer connector forPower Automateenables you to orchestrate and schedule flows, send notifications, and alerts, as part of a scheduled or triggered task. To learn more, read Azure Data Explorer connector for Microsoft Power Automate and Usage examples for Azure Data Explorer connector to Power Automate.

- **Dynamic events routing from event hub to multiple databases** - Routing events from Event Hub/IOT Hub/Event Grid is an activity commonly performed by Azure Data Explorer (ADX) users. Previously, you could route events only to a single database per defined connection. If you wanted to route the events to multiple databases, you needed to create multiple ADX cluster connections.

  To simplify the experience, we now support routing events data to multiple databases hosted in a single ADX cluster. To learn more about dynamic routing, read Ingest from event hub.

- **Configure a database using a KQL inline script as part of JSON ARM deployment template** - Previously, Azure Data Explorer supported running a Kusto Query Language (KQL) script to configure your database during Azure Resource Management (ARM) template deployment. Now, this can be done using an inline script provided inline as a parameter to a JSON ARM template. To learn more about using

a KQL inline script, read Configure a database using a Kusto Query Language script.

Data Integration

- **Export pipeline monitoring as a CSV** - The ability to export pipeline monitoring to CSV has been added after receiving many community requests for the feature. Simply filter the Pipeline runs screen to the data you want and click 'Export to CSV'. To learn more about exporting pipeline monitoring and other monitoring improvements, read Azure Data Factory monitoring improvements.

- **Incremental data loading made easy for Synapse and Azure Database for PostgreSQL and MySQL** - In a data integration solution, incrementally loading data after an initial full data load is a widely used scenario. Automatic incremental source data loading is now natively available for Synapse SQL and Azure Database for PostgreSQL and MySQL. With a simple click, users can "enable incremental extract" and only inserted or updated rows will be read by the pipeline. To learn more about incremental data loading, read Incrementally copy data from a source data store to a destination data store.

- **User-Defined Functions for Mapping Data Flows [Public Preview]** - We hear you that you can find yourself doing the same string manipulation, math calculations, or other complex logic several times. Now, with the new user-defined function feature, you can create customized expressions that can be reused across multiple mapping data flows. User-defined functions will be grouped in libraries to help developers group common sets of functions. Once you've created a data flow library, you can add in your user-defined functions. You can even add in multiple arguments to make your function more reusable. To learn more about user-defined functions, read User defined functions in mapping data flows.

- **Assert Error Handling** - Error handling has now been added to sinks following an assert transformation. Assert transformations enable you to build custom rules for data quality and data validation. You can now choose whether to output the failed rows to the selected sink or to a separate file. To learn more about error handling, read Assert data transformation in mapping data flow.

- **Mapping data flows projection editing** - New UI updates have been made to source projection editing in mapping data flows. You can now update source projection column names and column types with the click of a button. To learn more about source projection editing, read Source transformation in mapping data flow.

Synapse Link

Azure Synapse Link for SQL [Public Preview] - At Microsoft Build 2022, we announced the Public Preview availability of Azure Synapse Link for SQL, for both SQL Server 2022 and Azure SQL Database. Data-driven, quality insights are critical for companies to stay competitive. The speed to achieve those insights can make all the difference. The costly and time-consuming nature of traditional ETL and ELT pipelines is no longer enough. With this release, you can now take advantage of low- and no-code, near real-time data replication from your SQL-based operational stores into Azure Synapse Analytics. This makes it easier to run BI reporting on operational data in near real-time, with minimal impact on your operational store. To learn more, read Announcing the Public Preview of Azure Synapse Link for SQL and watch our YouTube video.

Apr 2022 update

The following updates are new to Azure Synapse Analytics this month.

SQL

- Cross-subscription restore for Azure Synapse SQL is now generally available. Previously, it took many undocumented steps to restore a dedicated SQL pool to another subscription. Now, with the PowerShell Az.Sql module 3.8 update, the Restore-AzSqlDatabase cmdlet can be used for cross-subscription restore. To learn more, see Restore a dedicated SQL pool (formerly SQL DW) to a different subscription.

- It is now possible to recover a SQL pool from a dropped server or workspace. With the PowerShell Restore cmdlets in Az.Sql and Az.Synapse modules, you can now restore from a deleted server or workspace without filing a support ticket. For more information, read Synapse workspace SQL pools or standalone SQL pools (formerly SQL DW), depending on your scenario.

Synapse database templates and database designer

- Based on popular customer feedback, we've made significant improvements to our exploration experience when creating a lake database using an industry template. To learn more, read Quickstart: Create a new Lake database leveraging database templates.

- We've added the option to clone a lake database. This unlocks additional opportunities to manage new versions of databases or support schemas that evolve in discrete steps. You can quickly clone a database using the action menu available on the lake database. To learn more, read How-to: Clone a lake database.

- You can now use wildcards to specify custom folder hierarchies. Lake databases sit on top of data that is in the lake and this data can live in nested folders that don't fit into clean partition patterns. Previously, querying lake databases required that your data exists in a simple directory structure that you could browse using the folder icon without the ability to manually specify directory structure or use wildcard characters. To learn more, read How-to: Modify a datalake.

Apache Spark for Synapse

- We are excited to announce the preview availability of Apache Spark™ 3.2 on Synapse Analytics. This new version incorporates user-requested enhancements and resolves 1,700+ Jira tickets. Please review the official release notes for the complete list of fixes and features and review the migration guidelines between Spark 3.1 and 3.2 to assess potential changes to your applications. For more details, read Apache Spark version support and Azure Synapse Runtime for Apache Spark 3.2.

- Assigning parameters dynamically based on variables, metadata, or specifying Pipeline specific parameters has been one of your top feature requests. Now, with the release of parameterization for the Spark job definition activity, you can do just that. For more details, read Transform data using Apache Spark job definition.

- We often receive customer requests to access the snapshot of the Notebook when there is a Pipeline Notebook run failure or there is a long-running Notebook job. With the release of the Synapse Notebook snapshot feature, you can now view the snapshot of the Notebook activity run with the original Notebook code, the cell output, and the input parameters. You can also access the snapshot of the referenced Notebook from the referencing Notebook cell output if you refer to other Notebooks through Spark utils. To learn more, read Transform data by running a Synapse notebook and Introduction to Microsoft Spark utilities.

Security

- The Synapse Monitoring Operator RBAC role is now generally available. Since the GA of Synapse, customers have asked for a fine-grained RBAC (role-based access control) role that allows a user persona to monitor the execution of Synapse Pipelines and Spark applications without having the ability to run or cancel the execution of these applications. Now, customers can assign the Synapse Monitoring Operator role to such monitoring personas. This allows organizations to stay compliant while having flexibility in the delegation of tasks to individuals or teams. Learn more by reading Synapse RBAC Roles.

Data integration

- Microsoft has added Dataverse as a source and sink connector to Synapse Data Flows so that you can now build low-code data transformation ETL jobs in Synapse directly accessing your Dataverse environment. For more details on how to use this new connector, read Mapping data flow properties.

- We heard from you that a 1-minute timeout for Web activity was not long enough, especially in cases of synchronous APIs. Now, with the response timeout property 'httpRequestTimeout', you can define

timeout for the HTTP request up to 10 minutes. Learn more by reading Web activity response timeout improvements.

Developer experience

- Previously, if you wanted to reference a notebook in another notebook, you could only reference published or committed content. Now, when using %run notebooks, you can enable 'unpublished notebook reference' which will allow you to reference unpublished notebooks. When enabled, notebook run will fetch the current contents in the notebook web cache, meaning the changes in your notebook editor can be referenced immediately by other notebooks without having to be published (Live mode) or committed (Git mode). To learn more, read Reference unpublished notebook.

Mar 2022 update

The following updates are new to Azure Synapse Analytics this month.

Developer Experience

- Code cells in Synapse notebooks that result in exception will now show standard output along with the exception message. This feature is supported for Python and Scala languages. To learn more, see the example output when a code statement fails.
- Synapse notebooks now support partial output when running code cells. To learn more, see the examples at this blog post
- You can now dynamically control Spark session configuration for the notebook activity with pipeline parameters. To learn more, see the variable explorer feature of Synapse notebooks.
- You can now reuse and manage notebook sessions without having to start a new one. You can easily connect a selected notebook to an active session in the list started from another notebook. You can detach a session from a notebook, stop the session, and monitor it. To learn more, see how to manage your active notebook sessions.
- Synapse notebooks now capture anything written through the Python logging module, in addition to the driver logs. To learn more, see support for Python logging.

SQL

- Column Level Encryption for Azure Synapse dedicated SQL Pools is now Generally Available. With column level encryption, you can use different protection keys for each column with each key having its own access permissions. The data in CLE-enforced columns are encrypted on disk and remain encrypted in memory until the DECRYPTBYKEY function is used to decrypt it. To learn more, see how to encrypt a data column.
- Serverless SQL pools now support better performance for CETAS (Create External Table as Select) and subsequent SELECT queries. The performance improvements include, a parallel execution plan resulting in faster CETAS execution and outputting multiple files. To learn more, see CETAS with Synapse SQL article and the blog post

Apache Spark for Synapse

- Synapse Spark Common Data Model (CDM) Connector is now Generally Available. The CDM format reader/writer enables a Spark program to read and write CDM entities in a CDM folder via Spark dataframes. To learn more, see how the CDM connector supports reading, writing data, examples, & known issues.
- Synapse Spark Dedicated SQL Pool (DW) Connector now supports improved performance. The new architecture eliminates redundant data movement and uses COPY-INTO instead of PolyBase. You can authenticate through SQL basic authentication or opt into the Azure Active Directory/Azure AD based authentication method. It now has ~5x improvements over the previous version. To learn more, see Azure

Synapse Dedicated SQL Pool Connector for Apache Spark

- Synapse Spark Dedicated SQL Pool (DW) Connector now supports all Spark Dataframe SaveMode choices. It supports Append, Overwrite, ErrorIfExists, and Ignore modes. The Append and Overwrite are critical for managing data ingestion at scale. To learn more, see DataFrame write SaveMode support
- Accelerate Spark execution speed using the new Intelligent Cache feature. This feature is currently in public preview. Intelligent Cache automatically stores each read within the allocated cache storage space, detecting underlying file changes and refreshing the files to provide the most recent data. To learn more, see how to Enable/Disable the cache for your Apache Spark pool or see the blog post

Security

- Azure Synapse Analytics now supports Azure Active Directory (Azure AD) authentication. You can turn on Azure AD authentication during the workspace creation or after the workspace is created. To learn more, see how to use Azure AD authentication with Synapse SQL.
- API support to raise or lower minimal TLS version for workspace managed SQL Server Dedicated SQL. To learn more, see how to update the minimum TLS setting or read the blog post for more details.

Data Integration

- Flowlets and CDC Connectors are now Generally Available. Flowlets in Synapse Data Flows allow for reusable and composable ETL logic. To learn more, see Flowlets in mapping data flow or see the blog post.
- sFTP connector for Synapse data flows. You can read and write data while transforming data from sftp using the visual low-code data flows interface in Synapse. To learn more, see source transformation
- Data flow improvements to Data Preview. To learn more, see Data Preview and debug improvements in Mapping Data Flows
- Pipeline script activity. The Script Activity enables data engineers to build powerful data integration pipelines that can read from and write to Synapse databases, and other database types. To learn more, see Transform data by using the Script activity in Azure Data Factory or Synapse Analytics

Feb 2022 update

The following updates are new to Azure Synapse Analytics this month.

SQL

- Serverless SQL Pools now support more consistent query execution times. Learn how Serverless SQL pools automatically detect spikes in read latency and support consistent query execution time.
- The `OPENJSON` function makes it easy to get array element indexes. To learn more, see how the OPENJSON function in a serverless SQL pool allows you to parse nested arrays and return one row for each JSON array element with the index of each element.

Data integration

- Upserting data is now supported by the copy activity. See how you can natively load data into a temporary table and then merge that data into a sink table with upsert.
- Transform Dynamics Data Visually in Synapse Data Flows. Learn more on how to use a Dynamics dataset or an inline dataset as source and sink types to transform data at scale.
- Connect to your SQL sources in data flows using Always Encrypted. To learn more, see how to securely connect to your SQL databases from Synapse data flows using Always Encrypted.
- Capture descriptions from asserts in Data Flows To learn more, see how to define your own dynamic

descriptive messages in the assert data flow transformation at the row or column level.

- Easily define schemas for complex type fields. To learn more, see how you can make the engine to automatically detect the schema of an embedded complex field inside a string column.

Jan 2022 update

The following updates are new to Azure Synapse Analytics this month.

Apache Spark for Synapse

You can now use four new database templates in Azure Synapse. Learn more about Automotive, Genomics, Manufacturing, and Pharmaceuticals templates from the blog post or the database templates article. These templates are currently in public preview and are available within the Synapse Studio gallery.

Machine Learning

Improvements to the Synapse Machine Learning library v0.9.5 (previously called MMLSpark). This release simplifies the creation of massively scalable machine learning pipelines with Apache Spark. To learn more, read the blog post about the new capabilities in this release or see the full release notes

Security

- The Azure Synapse Analytics security overview - A whitepaper that covers the five layers of security. The security layers include authentication, access control, data protection, network security, and threat protection. Understand each security feature in detailed to implement an industry-standard security baseline and protect your data on the cloud.

- TLS 1.2 is now required for newly created Synapse Workspaces. To learn more, see how TLS 1.2 provides enhanced security using this article or the blog post. Login attempts to a newly created Synapse workspace from connections using TLS versions lower than 1.2 will fail.

Data Integration

- Data quality validation rules using Assert transformation - You can now easily add data quality, data validation, and schema validation to your Synapse ETL jobs by using Assert transformation in Synapse data flows. To learn more, see the Assert transformation in mapping data flow article or the blog post.

- Native data flow connector for Dynamics - Synapse data flows can now read and write data directly to Dynamics through the new data flow Dynamics connector. Learn more on how to Create data sets in data flows to read, transform, aggregate, join, etc. using this article or the blog post. You can then write the data back into Dynamics using the built-in Synapse Spark compute.

- IntelliSense and auto-complete added to pipeline expressions - IntelliSense makes creating expressions, editing them easy. To learn more, see how to check your expression syntax, find functions, and add code to your pipelines.

Synapse SQL

- COPY schema discovery for complex data ingestion. To learn more, see the blog post or how GitHub leveraged this functionality in Introducing Automatic Schema Discovery with auto table creation for complex datatypes.

- Serverless SQL pools now support the HASHBYTES function. HASHBYTES is a T-SQL function, which hashes values. Learn how to use hash values in distributing data using this article or the blog post.

December 2021 update

The following updates are new to Azure Synapse Analytics this month.

Apache Spark for Synapse

- Accelerate Spark workloads with NVIDIA GPU acceleration blog article
- Mount remote storage to a Synapse Spark pool blog article
- Natively read & write data in ADLS with Pandas blog article
- Dynamic allocation of executors for Spark blog article

Machine Learning

- The Synapse Machine Learning library blog article
- Getting started with state-of-the-art pre-built intelligent models blog article
- Building responsible AI systems with the Synapse ML library blog article
- PREDICT is now GA for Synapse Dedicated SQL pools blog article
- Simple & scalable scoring with PREDICT and MLFlow for Apache Spark for Synapse blog article
- Retail AI solutions blog article

Security

- User-Assigned managed identities now supported in Synapse Pipelines in preview blog article
- Browse ADLS Gen2 folders in an Azure Synapse Analytics workspace in preview blog article

Data Integration

- Pipeline Fail activity blog article
- Mapping Data Flow gets new native connectors blog article
- More notebook export formats: HTML, Python, and LaTeX blog
- Three new chart types in notebook view: box plot, histogram, and pivot table blog
- Reconnect to lost notebook session blog

Integrate

- Synapse Link for Dataverse blog article
- Custom partitions for Synapse link for Azure Cosmos DB in preview blog article
- Map data tool (Public Preview), a no-code guided ETL experience blog article
- Quick reuse of spark cluster blog article
- External Call transformation blog article
- Flowlets (Public Preview) blog article

November 2021 update

The following updates are new to Azure Synapse Analytics this month.

Synapse Data Explorer

- Synapse Data Explorer now available in preview blog article

Working with Databases and Data Lakes

- Introducing Lake databases (formerly known as Spark databases) blog article
- Lake database designer now available in preview blog article
- Database Templates and Database Designer blog article

SQL

- Delta Lake support for serverless SQL is generally available blog article
- Query multiple file paths using OPENROWSET in serverless SQL blog article
- Serverless SQL queries can now return up to 200 GB of results blog article
- Handling invalid rows with OPENROWSET in serverless SQL blog article

Apache Spark for Synapse

- Accelerate Spark workloads with NVIDIA GPU acceleration blog article
- Mount remote storage to a Synapse Spark pool blog article
- Natively read & write data in ADLS with Pandas blog article
- Dynamic allocation of executors for Spark blog article

Machine Learning

- The Synapse Machine Learning library blog article
- Getting started with state-of-the-art pre-built intelligent models blog article
- Building responsible AI systems with the Synapse ML library blog article
- PREDICT is now GA for Synapse Dedicated SQL pools blog article
- Simple & scalable scoring with PREDICT and MLFlow for Apache Spark for Synapse blog article
- Retail AI solutions blog article

Security

- User-Assigned managed identities now supported in Synapse Pipelines in preview blog article
- Browse ADLS Gen2 folders in an Azure Synapse Analytics workspace in preview blog article

Data Integration

- Pipeline Fail activity blog article
- Mapping Data Flow gets new native connectors blog article

Synapse Link

- Synapse Link for Dataverse blog article
- Custom partitions for Synapse link for Azure Cosmos DB in preview blog article

October 2021 update

The following updates are new to Azure Synapse Analytics this month.

General

- Manage your cost with Azure Synapse pre-purchase plans blog article
- Move your Azure Synapse workspace across Azure regions blog article

Apache Spark for Synapse

- Spark performance optimizations blog

Security

- All Synapse RBAC roles are now generally available for use in production blog article
- Apply User-Assigned Managed Identities for Double Encryption blog article
- Synapse Administrators now have elevated access to dedicated SQL pools blog article

Governance

- Synapse workspaces can now automatically push lineage data to Microsoft Purview blog article

Integrate

- Use Stringify in data flows to easily transform complex data types to strings blog article
- Control Spark session time-to-live (TTL) in data flows blog article

CI/CD & Git

- Deploy Synapse workspaces using GitHub Actions blog article
- More control creating Git branches in Synapse Studio blog article

Developer Experience

- Enhanced Markdown editing in Synapse notebooks preview blog article
- Pandas dataframes automatically render as nicely formatted HTML tables blog article
- Use IPython widgets in Synapse Notebooks blog article
- Mssparkutils runtime context now available for Python and Scala blog article

Next steps

Get started with Azure Synapse Analytics

Request quota increases and get support for Azure Synapse Analytics

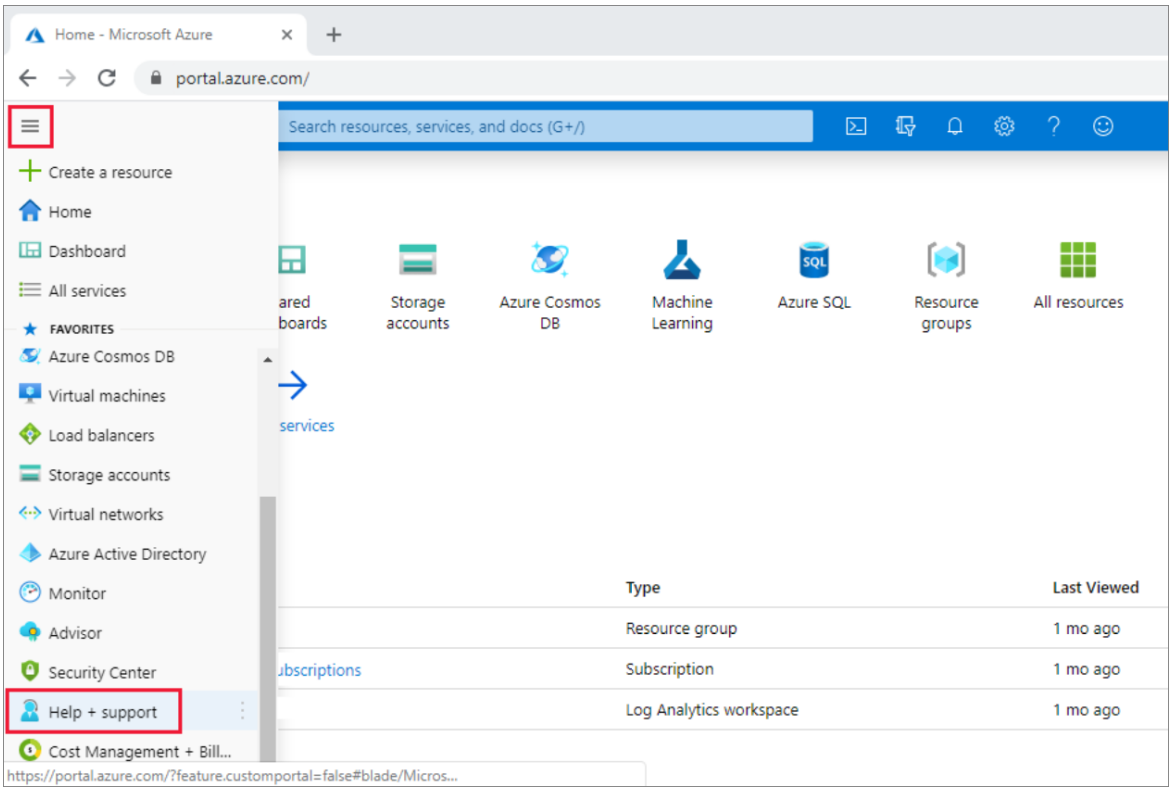
8/19/2022 • 3 minutes to read • Edit Online

This article describes how to submit a support ticket in the Azure portal for Azure Synapse Analytics. This process enables you to request a quota increase or to submit a technical support request for the engineering support team.

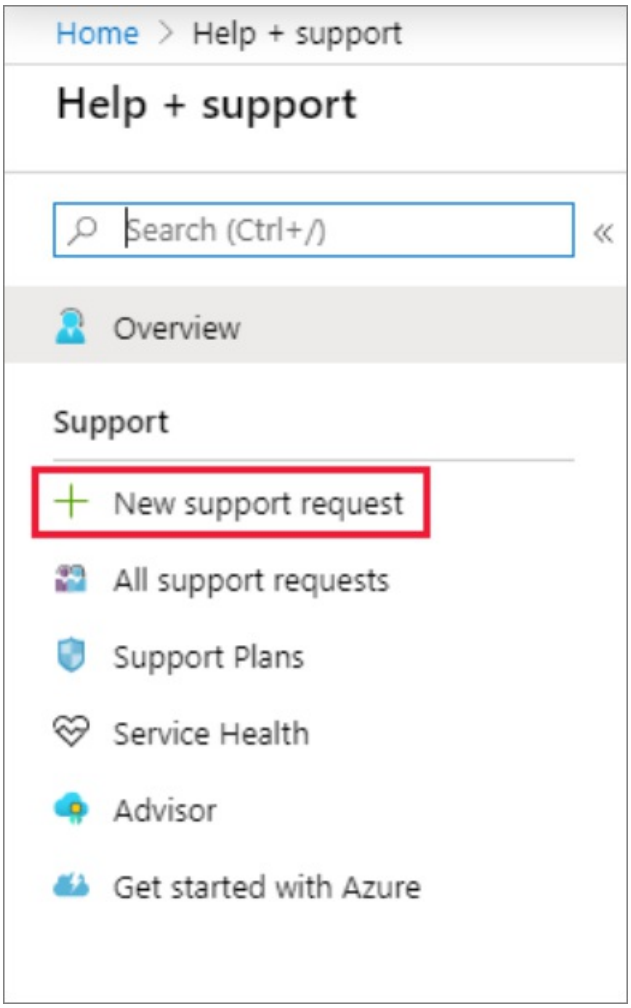
Create a support ticket

Use the following steps to create a new support request from the Azure portal for Azure Synapse Analytics.

1. On the Azure portal menu, select **Help + support**.



2. In **Help + support**, select **New support request**.

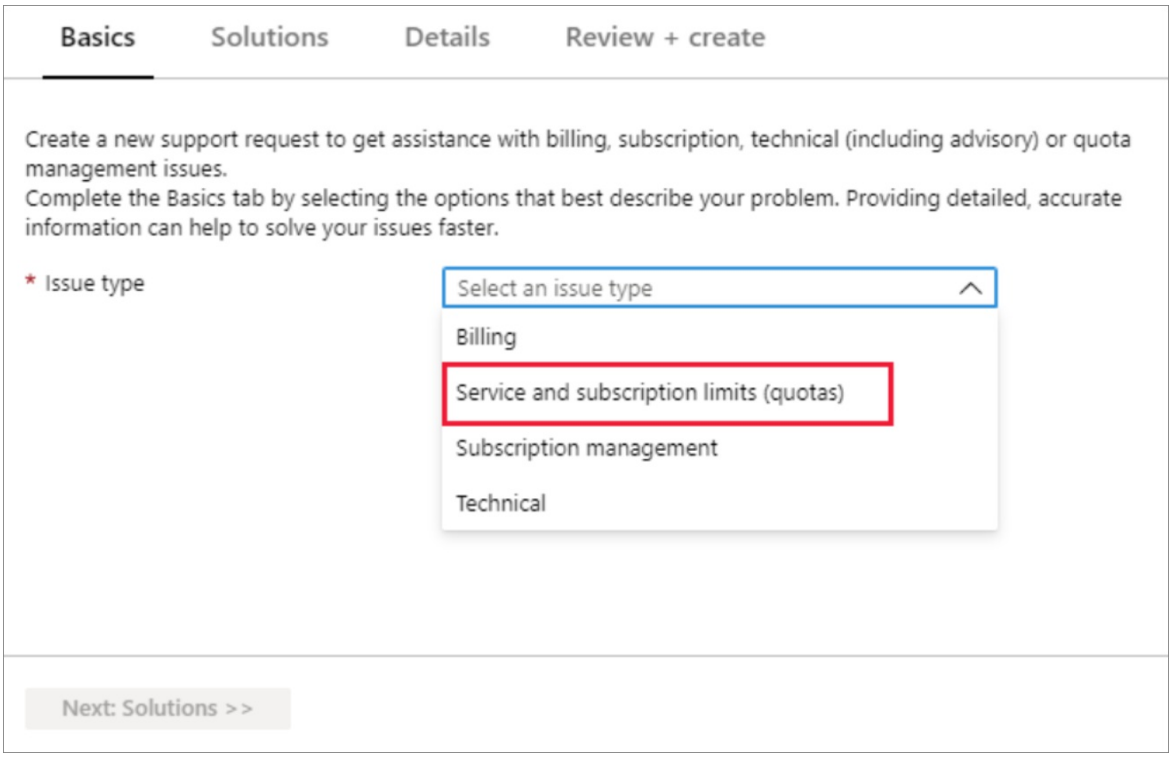


3. Review your Azure support plan.

   - **Billing, quota, and subscription management** support are available at all support levels.
   - **Break-fix** support is provided through Developer, Standard, Professional Direct, or Premier support. Break-fix issues are problems experienced by customers while using Azure where there is a reasonable expectation that Microsoft caused the problem.
   - **Developer mentoring** and **advisory services** are available at the Professional Direct and Premier support levels.

   If you have a Premier support plan, you can also report Azure Synapse Analytics issues on the Microsoft Premier online portal. See Azure support plans to learn more about the various support plans, including scope, response times, pricing, etc. For frequently asked questions about Azure support, see Azure support FAQs.

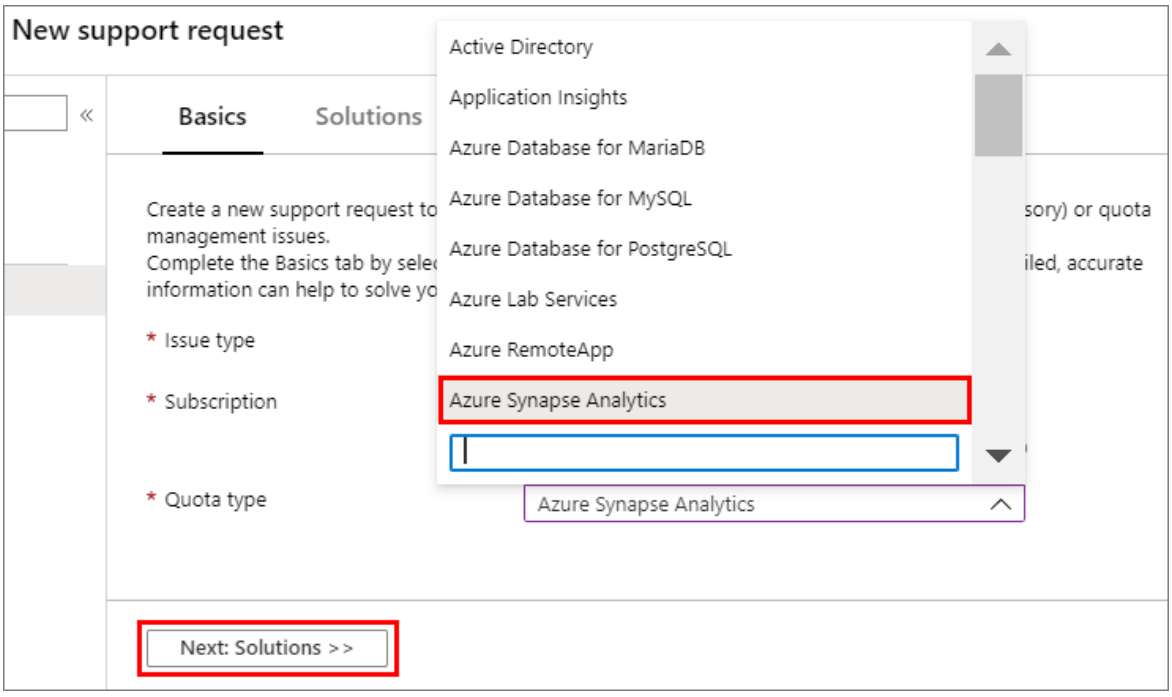
4. For **Issue type**, select the appropriate issue type. For break-fix problems, select **Technical**. For quota increase requests, select **Service and subscription limits (quotas)**.



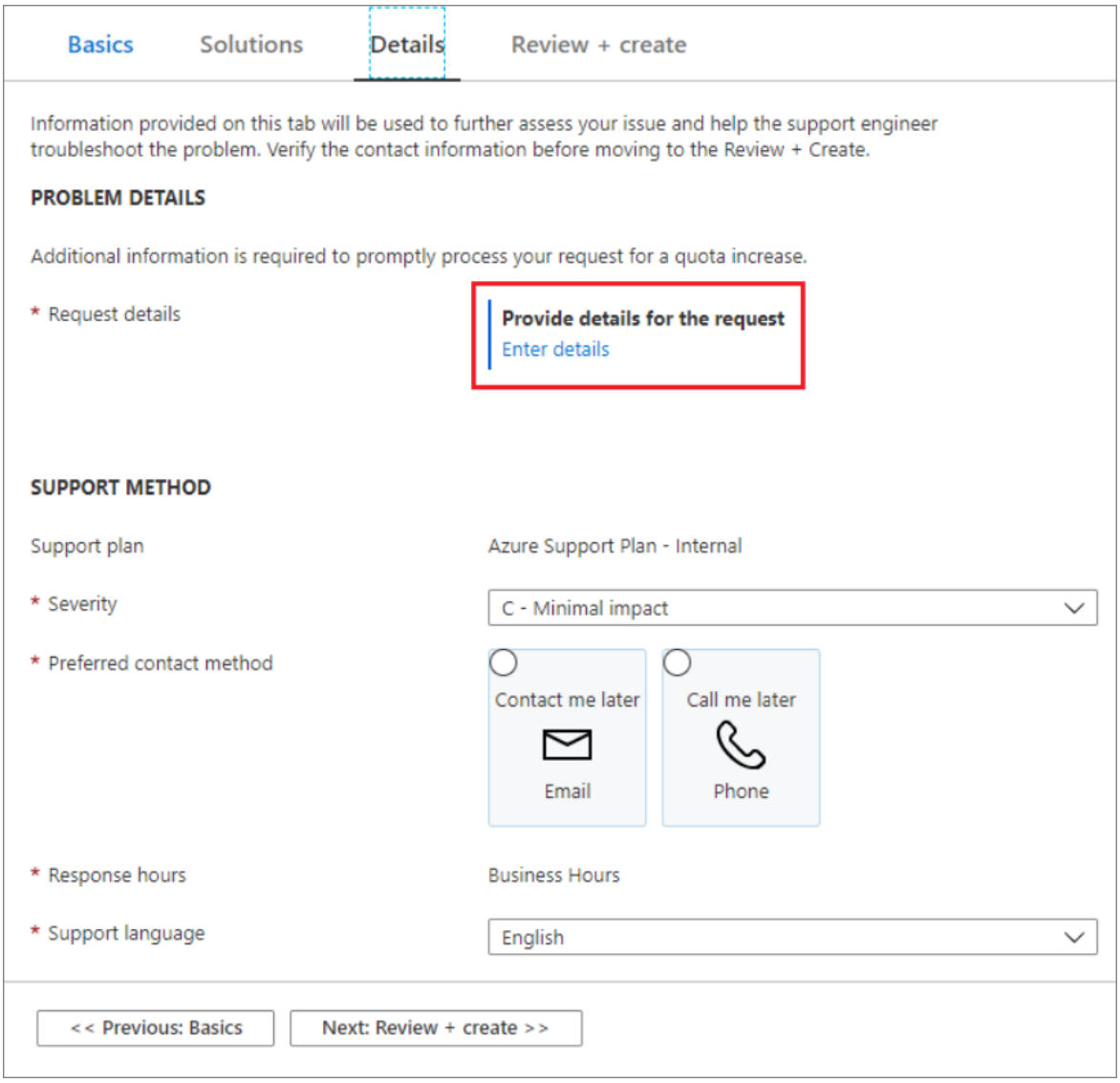
NOTE

This remainder of this article focusses on quota-increase requests. But you can also select **Technical** here for problem-resolution support requests. If you select **Technical**, you are asked to provide a summary and then identify a problem type by selecting **Select problem type**. You may see solutions to help resolve your issue. If the solutions presented do not resolve your issue, select **Next:Details** and complete the form to submit the support ticket.

5. For quota increase requests, select **Azure Synapse Analytics** for the **Quota type**. Then select **Next: Solutions >>**.



6. In the **Details** window, select **Enter details** to enter additional information.



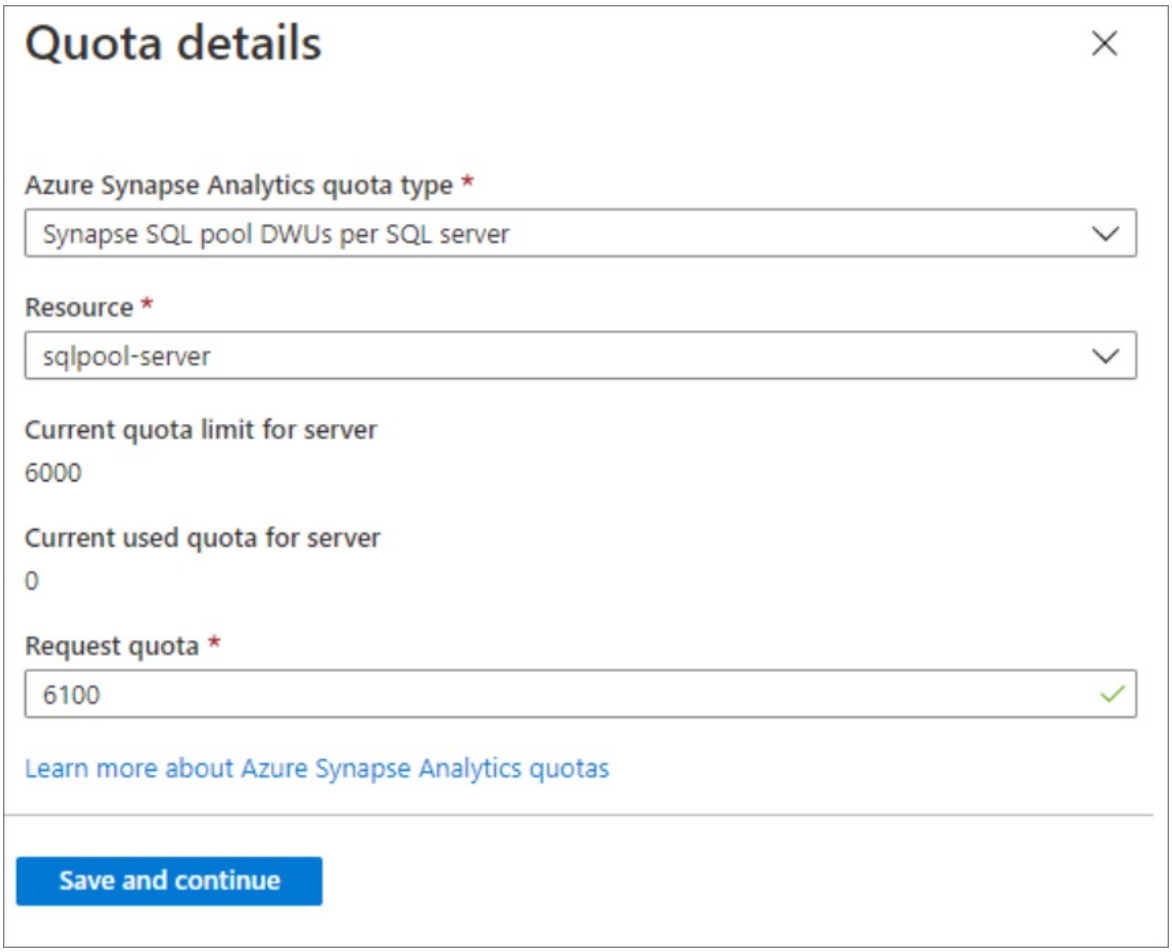
Quota request types

Selecting **Enter details** displays the **Quota details** window that allows you to add additional information. The following sections describe the different quota requests available for Azure Synapse Analytics.

Synapse SQL pool Data Warehouse Units (DWUs) per server

Use the following steps to request an increase in the DWUs per server.

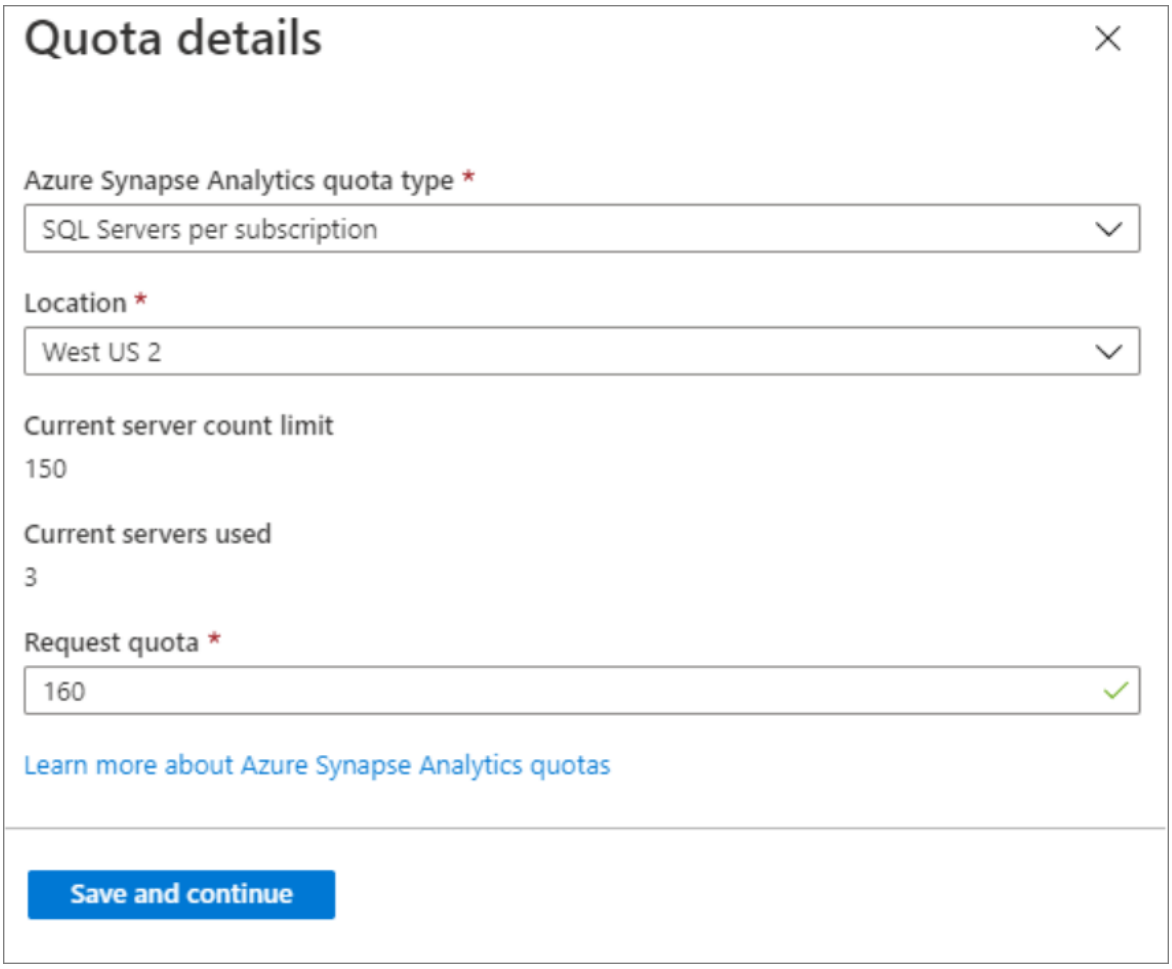
1. Select the **Synapse SQL pool DWUs per server** quota type.
2. Select the **Resource** you want to apply the quota increase to by using the dropdown list.
3. Enter your new quota into the **Request quota** section.
4. Select **Save and continue**.



Servers per subscription

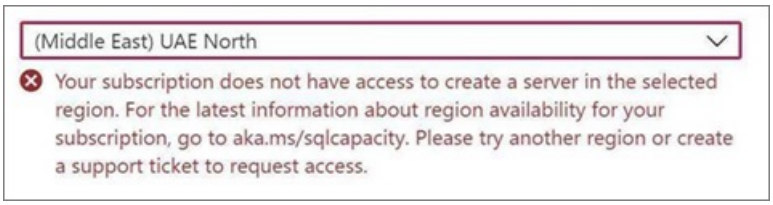
To request an increase in the number of servers per subscription, you'll need to complete the following steps:

1. Select the **SQL Servers per subscription** as the quota type.
2. In the **Location** list, select the Azure region to use. The quota is per subscription in each region.
3. In the **Request quota** field, enter your request for the maximum number of servers in that region.



4. Select **Save and continue**.

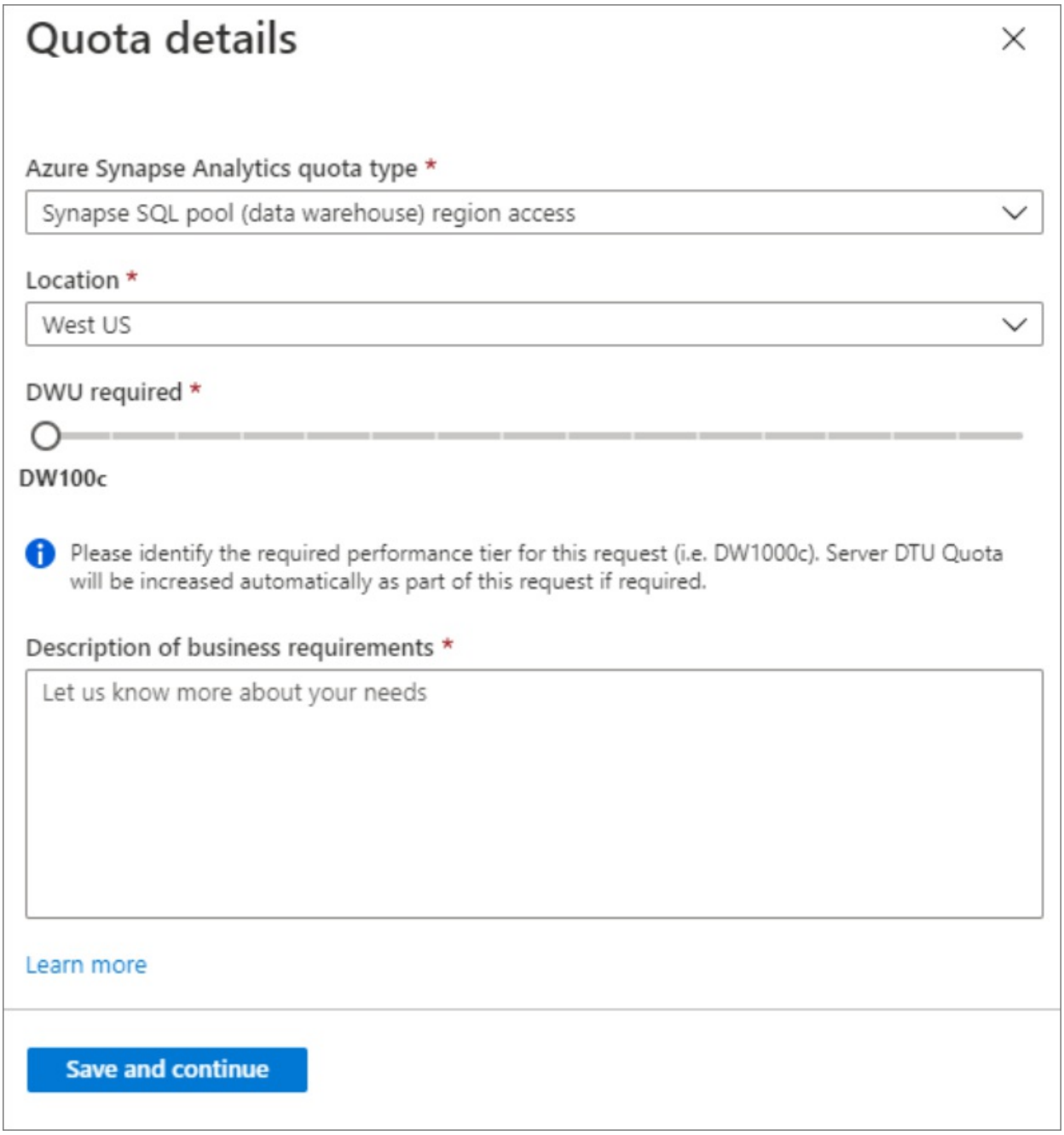
Some offer types aren't available in every region. You may see the following error:



Enable subscription access to a region

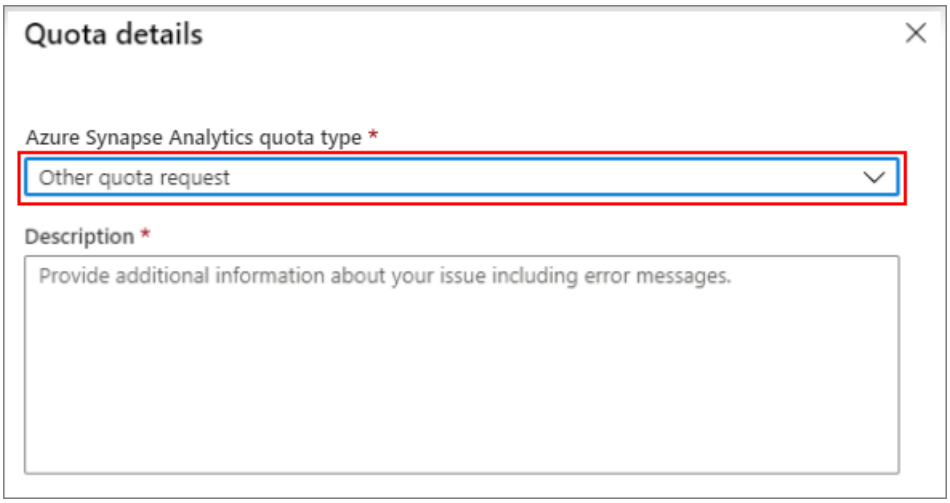
To enable region access for a subscription, you'll need to complete the following steps:

1. Select the **Synapse SQL pool (data warehouse) region access** quota type.
2. Select the region by choosing a **Location** from the dropdown list.
3. Indicate your DWU performance requirement in the **DWU required** section.
4. Enter your **Description of business requirements**.
5. Select **Save and continue**.



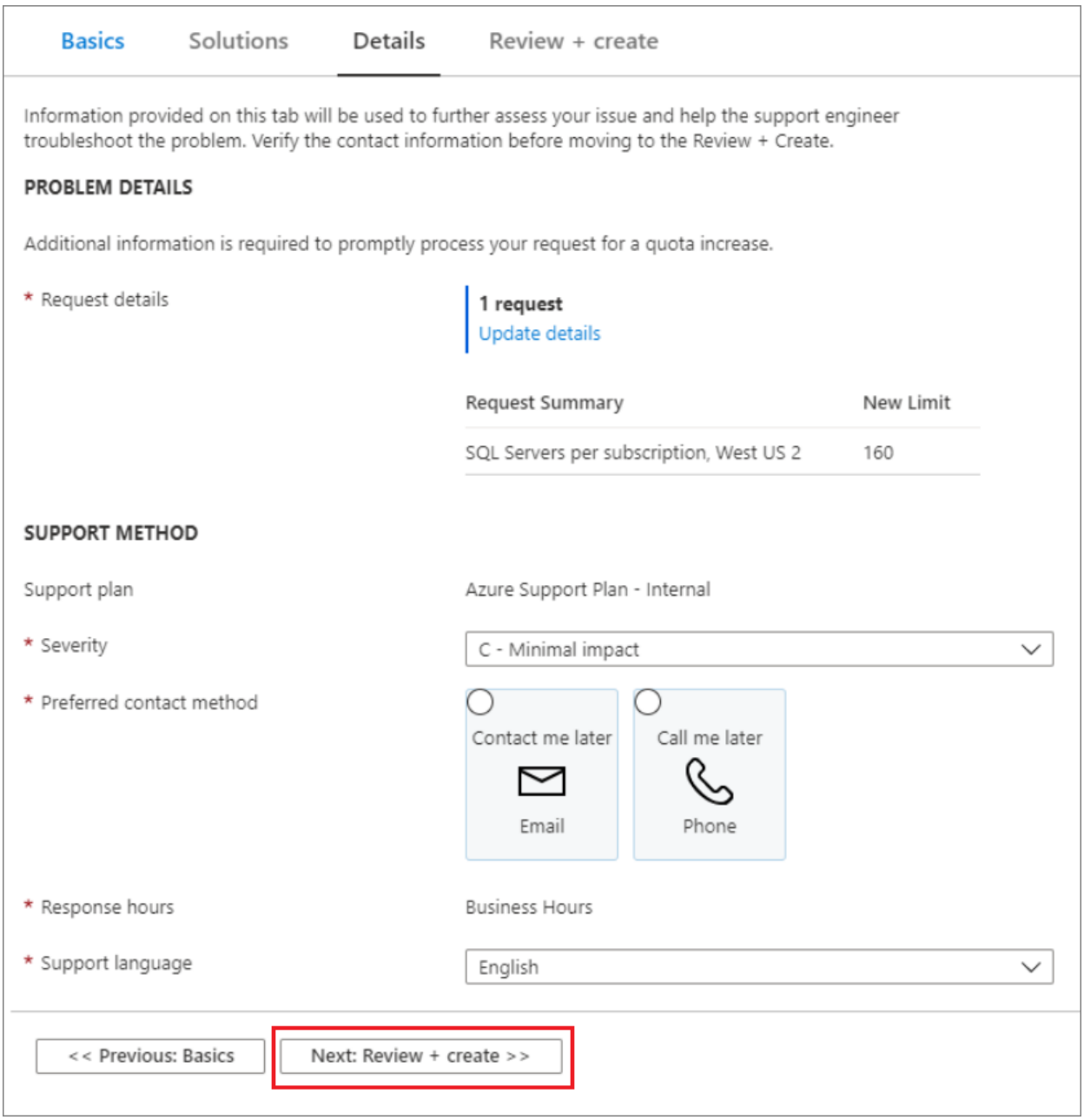
For other quota requests

Select **Other quota request** from the quota type drop-down menu for other quota request types:

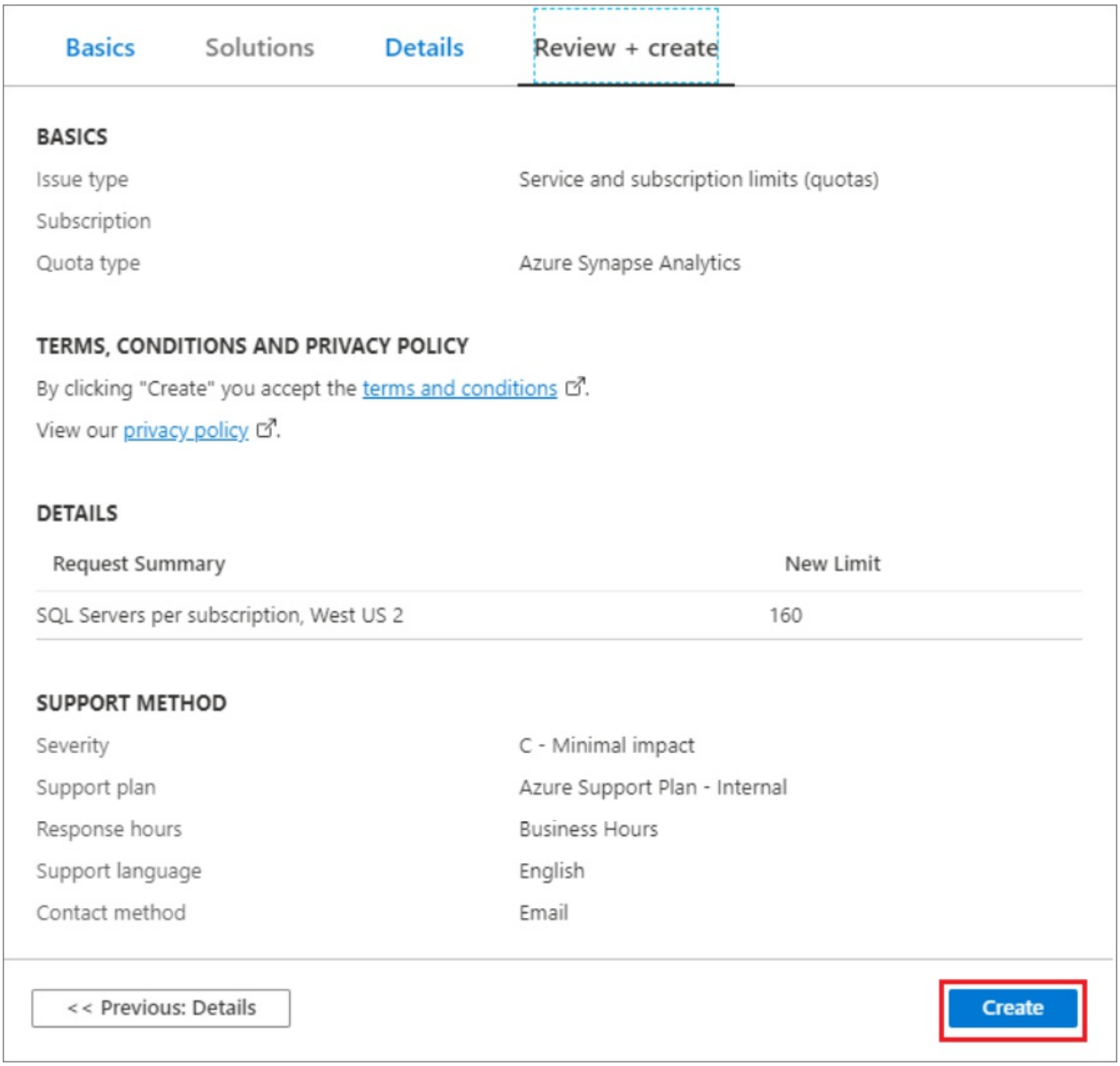


Submit your request

The final step is to fill in the remaining details of your SQL Database support request. Then select **Next: Review + create>>**.

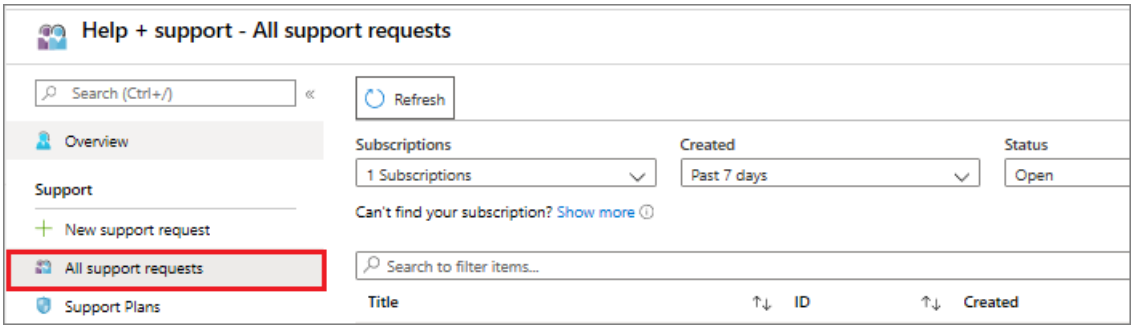


After reviewing the request details, select **Create** to submit the request.



Monitor a support ticket

After you've submitted the support request, the Azure support team will contact you. To check your request status and details, select **All support requests** on the dashboard.



Other resources

You can also connect with the Azure Synapse Analytics community on Stack Overflow or through the Microsoft Q&A question page for Azure Synapse Analytics.

Self-help for serverless SQL pool

8/19/2022 • 45 minutes to read • Edit Online

This article contains information about how to troubleshoot the most frequent problems with serverless SQL pool in Azure Synapse Analytics.

Synapse Studio

Synapse Studio is an easy-to-use tool that you can use to access your data by using a browser without a need to install database access tools. Synapse Studio isn't designed to read a large set of data or full management of SQL objects.

Serverless SQL pool is grayed out in Synapse Studio

If Synapse Studio can't establish a connection to serverless SQL pool, you'll notice that serverless SQL pool is grayed out or shows the status **Offline**. Usually, this problem occurs when one of the following cases happens:

- Your network prevents communication to the Azure Synapse Analytics back-end. The most frequent case is that port 1443 is blocked. To get serverless SQL pool to work, unblock this port. Other problems could prevent serverless SQL pool from working too. For more information, see the Troubleshooting guide.
- You don't have permission to sign in to serverless SQL pool. To gain access, an Azure Synapse workspace administrator must add you to the workspace administrator role or the SQL administrator role. For more information, see Azure Synapse access control.

Websocket connection closed unexpectedly

Your query might fail with the error message "Websocket connection was closed unexpectedly." This message means that your browser connection to Synapse Studio was interrupted, for example, because of a network issue.

To resolve this issue, rerun this query. If this message occurs often in your environment, get help from your network administrator. You can also check firewall settings, and check the Troubleshooting guide.

If the issue continues, create a support ticket through the Azure portal. Try Azure Data Studio or SQL Server Management Studio for the same queries instead of Synapse Studio for further investigation.

Serverless databases aren't shown in Synapse Studio

If you don't see the databases that are created in serverless SQL pool, check to see if your serverless SQL pool started. If serverless SQL pool is deactivated, the databases won't show. Execute any query, for example, `SELECT 1`, on serverless SQL pool to activate it and make the databases appear.

Synapse Serverless SQL pool shows as unavailable

Incorrect network configuration is often the cause of this behavior. Make sure the ports are properly configured. If you use a firewall or private endpoints, check these settings too. Finally, make sure the appropriate roles are granted.

Storage access

If you get errors while you try to access files in storage, make sure that you have permission to access data. You should be able to access publicly available files. If you try to access data without credentials, make sure that your Azure Active Directory (Azure AD) identity can directly access the files.

If you have a shared access signature key that you should use to access files, make sure that you created a

server-level or database-scoped credential that contains that credential. The credentials are required if you need to access data by using the workspace managed identity and custom service principal name (SPN).

Can't read, list, or access files in Azure Data Lake Storage

If you use an Azure AD sign-in without explicit credentials, make sure that your Azure AD identity can access the files in storage. Your Azure AD identity must have Blob Data Reader or list/read access control list (ACL) permissions to access the files. For more information, see Query fails because file cannot be opened.

If you access storage by using credentials, make sure that your managed identity or SPN has a Data Reader or Contributor role or ACL permissions. If you used a shared access signature token, make sure that it has `rl` permission and that it hasn't expired.

If you use a SQL sign-in and the `OPENROWSET` function without a data source, make sure that you have a server-level credential that matches the storage URI and has permission to access the storage.

Query fails because file can't be opened

If your query fails with the error "File cannot be opened because it does not exist or it is used by another process" and you're sure that both files exist and aren't used by another process, serverless SQL pool can't access the file. This problem usually happens because your Azure AD identity doesn't have rights to access the file or because a firewall is blocking access to the file.

By default, serverless SQL pool tries to access the file by using your Azure AD identity. To resolve this issue, you must have proper rights to access the file. The easiest way is to grant yourself a Storage Blob Data Contributor role on the storage account you're trying to query.

For more information, see:

- Azure AD access control for storage
- Control storage account access for serverless SQL pool in Synapse Analytics

Alternative to Storage Blob Data Contributor role

Instead of granting yourself a Storage Blob Data Contributor role, you can also grant more granular permissions on a subset of files.

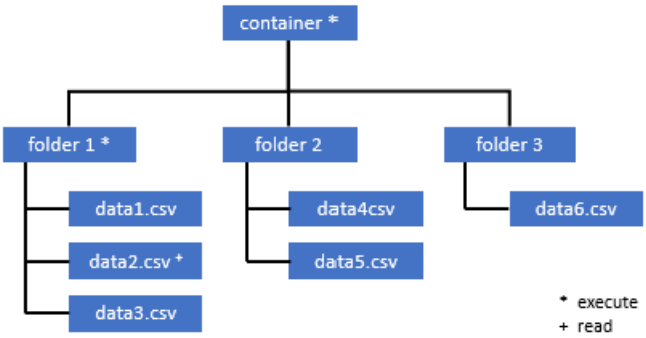
All users who need access to some data in this container also must have EXECUTE permission on all parent folders up to the root (the container).

Learn more about how to set ACLs in Azure Data Lake Storage Gen2.

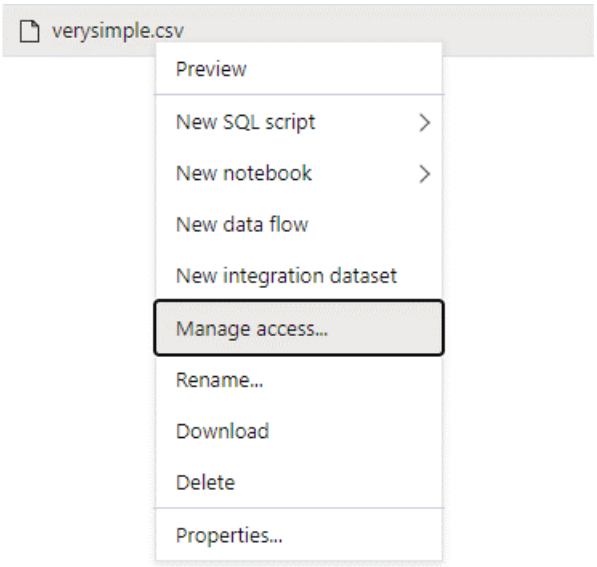
> **NOTE**
> Execute permission on the container level must be set within Data Lake Storage Gen2. Permissions on the folder can be set within Azure Synapse.

If you want to query data2.csv in this example, the following permissions are needed:

- Execute permission on container
- Execute permission on folder1
- Read permission on data2.csv



1. Sign in to Azure Synapse with an admin user that has full permissions on the data you want to access.
2. In the data pane, right-click the file and select **Manage access**.



3. Select at least **Read** permission. Enter the user's UPN or object ID, for example, user@contoso.com. Select **Add**.
4. Grant read permission for this user.

> **NOTE**
> For guest users, this step needs to be done directly with Azure Data Lake because it can't be done directly through Azure Synapse.

Content of directory on the path can't be listed

This error indicates that the user who's querying Azure Data Lake can't list the files in storage. There are several scenarios where this error might happen:

- The Azure AD user who's using Azure AD pass-through authentication doesn't have permission to list the

files in Data Lake Storage.

- The Azure AD or SQL user who's reading data by using a shared access signature key or workspace managed identity and that key or identity doesn't have permission to list the files in storage.
- The user who's accessing Dataverse data who doesn't have permission to query data in Dataverse. This scenario might happen if you use SQL users.
- The user who's accessing Delta Lake might not have permission to read the Delta Lake transaction log.

The easiest way to resolve this issue is to grant yourself the Storage Blob Data Contributor role in the storage account you're trying to query.

For more information, see:

- Azure AD access control for storage
- Control storage account access for serverless SQL pool in Synapse Analytics

Content of Dataverse table can't be listed

If you are using the Synapse link for Dataverse to read the linked DataVerse tables, you need to use Azure AD account to access the linked data using the serverless SQL pool. For more information, see Azure Synapse Link for Dataverse with Azure Data Lake.

If you try to use a SQL login to read an external table that is referencing the DataVerse table, you will get the following error:

```
External table '???' is not accessible because content of directory cannot be listed.
```

Dataverse external tables always use Azure AD passthrough authentication. You *can't* configure them to use a shared access signature key or workspace managed identity.

Content of Delta Lake transaction log can't be listed

The following error is returned when serverless SQL pool can't read the Delta Lake transaction log folder:

```
Content of directory on path 'https://.....core.windows.net/.../_delta_log/*.json' cannot be listed.
```

Make sure the `_delta_log` folder exists. Maybe you're querying plain Parquet files that aren't converted to Delta Lake format. If the `_delta_log` folder exists, make sure you have both read and list permission on the underlying Delta Lake folders. Try to read *.json files directly by using `FORMAT='csv'`. Put your URI in the BULK parameter:

```
select top 10 *
from openrowset(BULK 'https://.....core.windows.net/.../_delta_log/*.json',FORMAT='csv', FIELDQUOTE =
'0x0b', FIELDTERMINATOR ='0x0b',ROWTERMINATOR = '0x0b')
with (line varchar(max)) as logs
```

If this query fails, the caller doesn't have permission to read the underlying storage files.

Query execution

You might get errors during the query execution in the following cases:

- The caller can't access some objects.
- The query can't access external data.
- The query contains some functionalities that aren't supported in serverless SQL pools.

Query fails because it can't be executed due to current resource constraints

Your query might fail with the error message "This query cannot be executed due to current resource constraints." This message means serverless SQL pool can't execute at this moment because of resource constraints. Here are some troubleshooting options:

- Make sure data types of reasonable sizes are used.
- If your query targets Parquet files, consider defining explicit types for string columns because they'll be VARCHAR(8000) by default. Check inferred data types.
- If your query targets CSV files, consider creating statistics.
- To optimize your query, see Performance best practices for serverless SQL pool.

Query timeout expired

The error "Query timeout expired" is returned if the query executed more than 30 minutes on serverless SQL pool. This limit for serverless SQL pool can't be changed.

Try to optimize your query by applying best practices. Or try to materialize parts of your queries by using create external table as select (CETAS). Check if there's a concurrent workload running on serverless SQL pool because the other queries might take the resources. In that case, you might split the workload on multiple workspaces.

Invalid object name

The error "Invalid object name 'table name'" indicates that you're using an object, such as a table or view, that doesn't exist in the serverless SQL pool database. Try these options:

- List the tables or views and check if the object exists. Use SQL Server Management Studio or Azure Data Studio because Synapse Studio might show some tables that aren't available in serverless SQL pool.
- If you see the object, check that you're using some case-sensitive/binary database collation. Maybe the object name doesn't match the name that you used in the query. With a binary database collation, `Employee` and `employee` are two different objects.
- If you don't see the object, maybe you're trying to query a table from a lake or Spark database. The table might not be available in the serverless SQL pool because:
  - The table has some column types that can't be represented in serverless SQL pool.
  - The table has a format that isn't supported in serverless SQL pool. Examples are Delta or ORC.

Unclosed quotation mark after the character string

In rare cases, where you use the LIKE operator on a string column or some comparison with the string literals, you might get the following error:

```
Msg 105, Level 15, State 1, Line 88
Unclosed quotation mark after the character string
```

This error might happen if you use the `Latin1_General_100_BIN2_UTF8` collation on the column. Try to set `Latin1_General_100_CI_AS_SC_UTF8` collation on the column instead of the `Latin1_General_100_BIN2_UTF8` collation to resolve the issue. If the error is still returned, raise a support request through the Azure portal.

Couldn't allocate tempdb space while transferring data from one distribution to another

The error "Could not allocate tempdb space while transferring data from one distribution to another" is returned when the query execution engine can't process data and transfer it between the nodes that are executing the query. It's a special case of the generic query fails because it cannot be executed due to current resource constraints error. This error is returned when the resources allocated to the `tempdb` database are insufficient to run the query.

Apply best practices before you file a support ticket.

Query fails with an error handling an external file (max errors count reached)

If your query fails with the error message 'error handling external file: Max errors count reached', it means that there is a mismatch of a specified column type and the data that needs to be loaded.

To get more information about the error and which rows and columns to look at, change the parser version from `2.0` to `1.0`.

Example

If you want to query the file `names.csv` with this Query 1, Azure Synapse serverless SQL pool returns with the following error:

names.csv

```
Id,first name,
1, Adam
2,Bob
3,Charles
4,David
5,Eva
```

Query 1:

```
SELECT
TOP100*
FROM
OPENROWSET(
BULK'[FILE-PATH OF CSV FILE]',
FORMAT='CSV',
PARSER_VERSION='2.0',
       FIELDTERMINATOR =';',
       FIRSTROW = 2
    )
    WITH (
    [ID] SMALLINT,
    [Text] VARCHAR (1) COLLATE Latin1_General_BIN2
)

    AS[result]
```

Cause

"Error handling external file: 'Max error count reached'. File/External table name: [filepath]."

As soon as the parser version is changed from version 2.0 to 1.0, the error messages help to identify the problem. The new error message is now "Bulk load data conversion error (truncation) for row 1, column 2 (Text) in data file [filepath]."

Truncation tells you that your column type is too small to fit your data. The longest first name in this names.csv file has seven characters. The according data type to be used should be at least VARCHAR(7). The error is caused by this line of code:

```
    [Text] VARCHAR (1) COLLATE Latin1_General_BIN2
```

Changing the query accordingly resolves the error. After debugging, change the parser version to 2.0 again to achieve maximum performance.

For more information about when to use which parser version, see Use OPENROWSET using serverless SQL pool in Synapse Analytics.

```
SELECT
TOP100*
FROM
OPENROWSET(
BULK'[FILE-PATH OF CSV FILE]',
FORMAT='CSV',
PARSER_VERSION='2.0',
        FIELDTERMINATOR =';',
        FIRSTROW = 2
    )
    WITH (
    [ID] SMALLINT,
    [Text] VARCHAR (7) COLLATE Latin1_General_BIN2
)

    AS[result]
```

Can't bulk load because the file couldn't be opened

The error "Cannot bulk load because the file could not be opened" is returned if a file is modified during the query execution. Usually, you might get an error like "Cannot bulk load because the file {file path} could not be opened. Operating system error code 12. (The access code is invalid.)"

The serverless SQL pools can't read files that are being modified while the query is running. The query can't take a lock on the files. If you know that the modification operation is *append*, you can try to set the following option:

`{"READ_OPTIONS":["ALLOW_INCONSISTENT_READS"]}`.

For more information, see how to query append-only files or create tables on append-only files.

Query fails with data conversion error

Your query might fail with the error message "Bulk load data conversion error (type mismatches or invalid character for the specified code page) for row n, column m [columnname] in the data file [filepath]." This message means your data types didn't match the actual data for row number n and column m.

For instance, if you expect only integers in your data, but in row n there's a string, this error message is the one you'll get.

To resolve this problem, inspect the file and the data types you chose. Also check if your row delimiter and field terminator settings are correct. The following example shows how inspecting can be done by using VARCHAR as the column type.

For more information on field terminators, row delimiters, and escape quoting characters, see Query CSV files.

Example

If you want to query the file names.csv:

names.csv

```
Id, first name,
1,Adam
2,Bob
3,Charles
4,David
five,Eva
```

with the following Query 1:

Query 1:

```
SELECT
TOP100*
FROM
OPENROWSET(
BULK'[FILE-PATH OF CSV FILE]',
FORMAT='CSV',
        PARSER_VERSION='1.0',
       FIELDTERMINATOR =',',
       FIRSTROW = 2
    )
    WITH (
    [ID] SMALLINT,
    [Firstname] VARCHAR (25) COLLATE Latin1_General_BIN2
)

    AS[result]
```

Azure Synapse serverless SQL pool returns the error "Bulk load data conversion error (type mismatch or invalid character for the specified code page) for row 6, column 1 (ID) in data file [filepath]."

It's necessary to browse the data and make an informed decision to handle this problem. To look at the data that causes this problem, the data type needs to be changed first. Instead of querying the ID column with the data type SMALLINT, VARCHAR(100) is now used to analyze this issue.

It is necessary to browse the data and make an informed decision to handle this problem. To look at the data that causes this problem, the data type needs to be changed first. Instead of querying column "ID" with the data type "SMALLINT", VARCHAR(100) is now used to analyze this issue.

With this slightly changed Query 2, the data can now be processed to return the list of names.

Query 2:

```
SELECT
TOP100*
FROM
OPENROWSET(
BULK'[FILE-PATH OF CSV FILE]',
FORMAT='CSV',
PARSER_VERSION='1.0',
       FIELDTERMINATOR =',',
       FIRSTROW = 2
    )
    WITH (
    [ID] VARCHAR(100),
    [Firstname] VARCHAR (25) COLLATE Latin1_General_BIN2
)

    AS[result]
```

names.csv

```
Id, first name,
1, Adam
2, Bob
3, Charles
4, David
five, Eva
```

You might observe that the data has unexpected values for ID in the fifth row. In such circumstances, it's important to align with the business owner of the data to agree on how corrupt data like this example can be

avoided. If prevention isn't possible at the application level, reasonable-sized VARCHAR might be the only option here.

> **TIP**
> Try to make VARCHAR() as short as possible. Avoid VARCHAR(MAX) if possible because it can impair performance.

The query result doesn't look as expected

Your query might not fail, but you might see that your result set isn't as expected. The resulting columns might be empty or unexpected data might be returned. In this scenario, it's likely that a row delimiter or field terminator was incorrectly chosen.

To resolve this problem, take another look at the data and change those settings. Debugging this query is easy, as shown in the following example.

Example

If you want to query the file `names.csv` with the query in Query 1, Azure Synapse serverless SQL pool returns with a result that looks odd:

names.csv

```
Id,first name,
1, Adam
2, Bob
3, Charles
4, David
5, Eva
```

```
SELECT
TOP100*
FROM
OPENROWSET(
BULK'[FILE-PATH OF CSV FILE]',
FORMAT='CSV',
        PARSER_VERSION='1.0',
       FIELDTERMINATOR =';',
       FIRSTROW = 2
    )
    WITH (
    [ID] VARCHAR(100),
    [Firstname] VARCHAR (25) COLLATE Latin1_General_BIN2
)

    AS[result]
```

ID	FIRSTNAME
1,Adam	NULL
2,Bob	NULL
3,Charles	NULL
4,David	NULL
5,Eva	NULL

There seems to be no value in the column `Firstname`. Instead, all values ended up being in the `ID` column. Those values are separated by a comma. The problem was caused by this line of code because it's necessary to choose the comma instead of the semicolon symbol as field terminator:

```
FIELDTERMINATOR =';',
```

Changing this single character solves the problem:

```
FIELDTERMINATOR =',',
```

The result set created by Query 2 now looks as expected:

Query 2:

```
SELECT
TOP100*
FROM
OPENROWSET(
BULK'[FILE-PATH OF CSV FILE]',
FORMAT='CSV',
        PARSER_VERSION='1.0',
       FIELDTERMINATOR =',',
       FIRSTROW = 2
    )
    WITH (
    [ID] VARCHAR(100),
    [Firstname] VARCHAR (25) COLLATE Latin1_General_BIN2
)

    AS[result]
```

returns

ID	FIRSTNAME
1	Adam
2	Bob
3	Charles
4	David
5	Eva

Column of type isn't compatible with external data type

If your query fails with the error message "Column [column-name] of type [type-name] is not compatible with external data type [...]," it's likely that a PARQUET data type was mapped to an incorrect SQL data type. For instance, if your Parquet file has a column price with float numbers (like 12.89) and you tried to map it to INT, this error message is the one you'll get.

To resolve this issue, inspect the file and the data types you chose. This mapping table helps to choose a correct SQL data type. As a best practice, specify mapping only for columns that would otherwise resolve into the VARCHAR data type. Avoiding VARCHAR when possible leads to better performance in queries.

Example

If you want to query the file taxi-data.parquet with this Query 1, Azure Synapse serverless SQL pool returns the following error:

taxi-data.parquet:

PASSENGERCOUNT	SUMTRIPDISTANCE	AVGTRIPDISTANCE
1	2635668.66000064	6.72731710678951
2	172174.330000005	2.97915543404919
3	296384.390000011	2.8991352022851
4	12544348.58999806	6.30581582240281
5	13091570.2799993	111.065989028627

Query 1:

```
SELECT
    *
FROM
    OPENROWSET(
        BULK '<filepath>taxi-data.parquet',
        FORMAT='PARQUET'
    )  WITh
        (
        PassengerCount INT,
        SumTripDistance INT,
        AVGTripDistance FLOAT
        )

    AS [result]
```

```
Column 'SumTripDistance' of type 'INT' is not compatible with external data type 'Parquet physical type:
DOUBLE', please try with 'FLOAT'. File/External table name: '<filepath>taxi-data.parquet'.
```

This error message tells you that data types aren't compatible and comes with the suggestion to use FLOAT instead of INT. The error is caused by this line of code:

```
SumTripDistance INT,
```

With this slightly changed Query 2, the data can now be processed and shows all three columns:

Query 2:

```
SELECT
    *
FROM
    OPENROWSET(
        BULK '<filepath>taxi-data.parquet',
        FORMAT='PARQUET'
    )  WITh
        (
        PassengerCount INT,
        SumTripDistance FLOAT,
        AVGTripDistance FLOAT
        )

    AS [result]
```

Query references an object that isn't supported in distributed processing mode

The error "The query references an object that is not supported in distributed processing mode" indicates that you've used an object or function that can't be used while you query data in Azure Storage or Azure Cosmos DB analytical storage.

Some objects, like system views, and functions can't be used while you query data stored in Azure Data Lake or Azure Cosmos DB analytical storage. Avoid using the queries that join external data with system views, load external data in a temp table, or use some security or metadata functions to filter external data.

WaitIOCompletion call failed

The error message "WaitIOCompletion call failed" indicates that the query failed while waiting to complete the I/O operation that reads data from the remote storage, Azure Data Lake.

The error message has the following pattern:

```
Error handling external file: 'WaitIOCompletion call failed. HRESULT = ???'. File/External table name...
```

Make sure that your storage is placed in the same region as serverless SQL pool. Check the storage metrics and verify there are no other workloads on the storage layer, such as uploading new files, that could saturate I/O requests.

The field HRESULT contains the result code. The following error codes are the most common along with their potential solutions.

- 0x80070002
- 0x80070005
- 0x80070008
- 0x8007000C
- 0x800700A1
- 0x80070057
- 0x80072EE7

This error code means the source file isn't in storage.

There are reasons why this error code can happen:

- The file was deleted by another application.
  - In this common scenario, the query execution starts, it enumerates the files, and the files are found. Later, during the query execution, a file is deleted. For example, it could be deleted by Databricks, Spark, or Azure Data Factory. The query fails because the file isn't found.
- This issue can also occur with the Delta format. The query might succeed on retry because there's a new

version of the table and the deleted file isn't queried again.

- An invalid execution plan is cached.
  - As a temporary mitigation, run the command `DBCC FREEPROCCACHE`. If the problem persists, create a support ticket.

Incorrect syntax near NOT

The error "Incorrect syntax near 'NOT'" indicates there are some external tables with columns that contain the NOT NULL constraint in the column definition. Update the table to remove NOT NULL from the column definition. This error can sometimes also occur transiently with tables created from a CETAS statement. If the problem doesn't resolve, you can try dropping and re-creating the external table.

Partitioning column returns NULL values

If your query returns NULL values instead of partitioning columns or can't find the partition columns, you have a few possible troubleshooting steps:

- If you use tables to query a partitioned dataset, be aware that tables don't support partitioning. Replace the table with the partitioned views.
- If you use the partitioned views with the OPENROWSET that queries partitioned files by using the FILEPATH() function, make sure you correctly specified the wildcard pattern in the location and used the proper index for referencing the wildcard.
- If you're querying the files directly in the partitioned folder, be aware that the partitioning columns aren't the parts of the file columns. The partitioning values are placed in the folder paths and not the files. For this reason, the files don't contain the partitioning values.

Inserting value to batch for column type DATETIME2 failed

The error "Inserting value to batch for column type DATETIME2 failed" indicates that the serverless pool can't read the date values from the underlying files. The datetime value stored in the Parquet or Delta Lake file can't be represented as a `DATETIME2` column.

Inspect the minimum value in the file by using Spark, and check that some dates are less than 0001-01-03. If you stored the files by using Spark 2.4, the datetime values before are written by using the Julian calendar that isn't aligned with the proleptic Gregorian calendar used in serverless SQL pools.

There might be a two-day difference between the Julian calendar used to write the values in Parquet (in some Spark versions) and the proleptic Gregorian calendar used in serverless SQL pool. This difference might cause conversion to a negative date value, which is invalid.

Try to use Spark to update these values because they're treated as invalid date values in SQL. The following sample shows how to update the values that are out of SQL date ranges to NULL in Delta Lake:

```
from delta.tables import *
from pyspark.sql.functions import *

deltaTable = DeltaTable.forPath(spark,
             "abfss://my-container@myaccount.dfs.core.windows.net/delta-lake-data-set")
deltaTable.update(col("MyDateTimeColumn") < '0001-02-02', { "MyDateTimeColumn": null } )
```

This change removes the values that can't be represented. The other date values might be properly loaded but incorrectly represented because there's still a difference between Julian and proleptic Gregorian calendars. You might see unexpected date shifts even for the dates before `1900-01-01` if you use Spark 3.0 or older versions.

Consider migrating to Spark 3.1 or higher. It uses a proleptic Gregorian calendar that's aligned with the calendar in serverless SQL pool. Reload your legacy data with the higher version of Spark, and use the following setting to correct the dates:

```
spark.conf.set("spark.sql.legacy.parquet.int96RebaseModeInWrite", "CORRECTED")
```

Query failed because of a topology change or compute container failure

This error might indicate that some internal process issue happened in serverless SQL pool. File a support ticket with all necessary details that could help the Azure support team investigate the issue.

Describe anything that might be unusual compared to the regular workload. For example, perhaps there was a large number of concurrent requests or a special workload or query started executing before this error happened.

Configuration

Serverless SQL pools enable you to use T-SQL to configure database objects. There are some constraints:

- You can't create objects in master and lakehouse or Spark databases.
- You must have a master key to create credentials.
- You must have permission to reference data that's used in the objects.

Can't create a database

If you get the error "CREATE DATABASE failed. User database limit has been already reached," you've created the maximal number of databases that are supported in one workspace. For more information, see Constraints.

- If you need to separate the objects, use schemas within the databases.
- If you need to reference Azure Data Lake storage, create lakehouse databases or Spark databases that will be synchronized in serverless SQL pool.

Create a master key in the database or open the master key in the session before performing this operation

If your query fails with the error message "Please create a master key in the database or open the master key in the session before performing this operation," it means that your user database has no access to a master key at the moment.

Most likely, you created a new user database and haven't created a master key yet.

To resolve this problem, create a master key with the following query:

```
CREATE MASTER KEY [ ENCRYPTION BY PASSWORD ='password' ];
```

> **NOTE**
> Replace `'password'` with a different secret here.

CREATE statement isn't supported in the master database

If your query fails with the error message "Failed to execute query. Error: CREATE EXTERNAL TABLE/DATA SOURCE/DATABASE SCOPED CREDENTIAL/FILE FORMAT is not supported in master database," it means that the master database in serverless SQL pool doesn't support the creation of:

- External tables.
- External data sources.
- Database scoped credentials.
- External file formats.

Here's the solution:

1. Create a user database:

```
CREATE DATABASE <DATABASE_NAME>
```

2. Execute a CREATE statement in the context of <DATABASE_NAME>, which failed earlier for the master database.

   Here's an example of the creation of an external file format:

```
USE <DATABASE_NAME>
CREATE EXTERNAL FILE FORMAT [SynapseParquetFormat]
WITH ( FORMAT_TYPE = PARQUET)
```

Operation isn't allowed for a replicated database

If you're trying to create SQL objects, users, or change permissions in a database, you might get errors like "Operation CREATE USER is not allowed for a replicated database." This error is returned when you try to create objects in a database that's shared with Spark pool. The databases that are replicated from Apache Spark pools are read only. You can't create new objects into a replicated database by using T-SQL.

Create a separate database and reference the synchronized tables by using three-part names and cross-database queries.

Can't create Azure AD sign-in or user

If you get an error while you're trying to create a new Azure AD sign-in or user in a database, check the sign-in you used to connect to your database. The sign-in that's trying to create a new Azure AD user must have permission to access the Azure AD domain and check if the user exists. Be aware that:

- SQL sign-ins don't have this permission, so you'll always get this error if you use SQL authentication.
- If you use an Azure AD sign-in to create new sign-ins, check to see if you have permission to access the Azure AD domain.

Azure Cosmos DB

Serverless SQL pools enable you to query Azure Cosmos DB analytical storage by using the `OPENROWSET` function. Make sure that your Azure Cosmos DB container has analytical storage. Make sure that you correctly specified the account, database, and container name. Also, make sure that your Azure Cosmos DB account key is valid. For more information, see Prerequisites.

Can't query Azure Cosmos DB by using the OPENROWSET function

If you can't connect to your Azure Cosmos DB account, look at the prerequisites. Possible errors and troubleshooting actions are listed in the following table.

ERROR	ROOT CAUSE
Syntax errors: - Incorrect syntax near `OPENROWSET`. - `...` isn't a recognized `BULK OPENROWSET` provider option. - Incorrect syntax near `...`.	Possible root causes: - Not using Azure Cosmos DB as the first parameter. - Using a string literal instead of an identifier in the third parameter. - Not specifying the third parameter (container name).
There was an error in the Azure Cosmos DB connection string.	- The account, database, or key isn't specified. - An option in a connection string isn't recognized. - A semicolon (`;`) is placed at the end of a connection string.

ERROR	ROOT CAUSE
Resolving Azure Cosmos DB path has failed with the error "Incorrect account name" or "Incorrect database name."	The specified account name, database name, or container can't be found, or analytical storage hasn't been enabled to the specified collection.
Resolving Azure Cosmos DB path has failed with the error "Incorrect secret value" or "Secret is null or empty."	The account key isn't valid or is missing.

UTF-8 collation warning is returned while reading Azure Cosmos DB string types

Serverless SQL pool returns a compile-time warning if the `OPENROWSET` column collation doesn't have UTF-8 encoding. You can easily change the default collation for all `OPENROWSET` functions running in the current database by using the T-SQL statement `alter database current collate Latin1_General_100_CI_AS_SC_UTF8`.

Latin1_General_100_BIN2_UTF8 collation provides the best performance when you filter your data by using string predicates.

Missing rows in Azure Cosmos DB analytical store

Some items from Azure Cosmos DB might not be returned by the `OPENROWSET` function. Be aware that:

- There's a synchronization delay between the transactional and analytical store. The document you entered in the Azure Cosmos DB transactional store might appear in the analytical store after two to three minutes.
- The document might violate some schema constraints.

Query returns NULL values in some Azure Cosmos DB items

Azure Synapse SQL returns NULL instead of the values that you see in the transaction store in the following cases:

- There's a synchronization delay between the transactional and analytical store. The value that you entered in the Azure Cosmos DB transactional store might appear in the analytical store after two to three minutes.
- There might be a wrong column name or path expression in the WITH clause. The column name (or path expression after the column type) in the WITH clause must match the property names in the Azure Cosmos DB collection. Comparison is case sensitive. For example, `productCode` and `ProductCode` are different properties. Make sure that your column names exactly match the Azure Cosmos DB property names.
- The property might not be moved to the analytical storage because it violates some schema constraints, such as more than 1,000 properties or more than 127 nesting levels.
- If you use well-defined schema representation, the value in the transactional store might have a wrong type. Well-defined schema locks the types for each property by sampling the documents. Any value added in the transactional store that doesn't match the type is treated as a wrong value and not migrated to the analytical store.
- If you use full-fidelity schema representation, make sure that you're adding the type suffix after the property name like `$.price.int64`. If you don't see a value for the referenced path, maybe it's stored under a different type path, for example, `$.price.float64`. For more information, see Query Azure Cosmos DB collections in the full-fidelity schema.

Column isn't compatible with external data type

The error "Column `column name` of the type `type name` is not compatible with the external data type `type name`" is returned if the specified column type in the WITH clause doesn't match the type in the Azure Cosmos DB container. Try to change the column type as it's described in the section Azure Cosmos DB to SQL type mappings or use the VARCHAR type.

Resolving Azure Cosmos DB path has failed with error

If you get the error "Resolving CosmosDB path has failed with error 'This request is not authorized to perform

this operation'," check to see if you used private endpoints in Azure Cosmos DB. To allow serverless SQL pool to access an analytical store with private endpoints, you must configure private endpoints for the Azure Cosmos DB analytical store.

Azure Cosmos DB performance issues

If you experience some unexpected performance issues, make sure that you applied best practices, such as:

- Make sure that you placed the client application, serverless pool, and Azure Cosmos DB analytical storage in the same region.
- Make sure that you use the WITH clause with optimal data types.
- Make sure that you use Latin1_General_100_BIN2_UTF8 collation when you filter your data by using string predicates.
- If you have repeating queries that might be cached, try to use CETAS to store query results in Azure Data Lake Storage.

Delta Lake

There are some limitations and known issues that you might see in Delta Lake support in serverless SQL pools:

- Make sure that you're referencing the root Delta Lake folder in the OPENROWSET function or external table location.
  - The root folder must have a subfolder named `_delta_log`. The query fails if there's no `_delta_log` folder. If you don't see that folder, you're referencing plain Parquet files that must be converted to Delta Lake by using Apache Spark pools.
  - Don't specify wildcards to describe the partition schema. The Delta Lake query automatically identifies the Delta Lake partitions.
- Delta Lake tables created in the Apache Spark pools aren't automatically available in serverless SQL pool. To query such Delta Lake tables by using the T-SQL language, run the CREATE EXTERNAL TABLE statement and specify Delta as the format.
- External tables don't support partitioning. Use partitioned views on the Delta Lake folder to use the partition elimination. See known issues and workarounds later in the article.
- Serverless SQL pools don't support time travel queries. Use Apache Spark pools in Synapse Analytics to read historical data.
- Serverless SQL pools don't support updating Delta Lake files. You can use serverless SQL pool to query the latest version of Delta Lake. Use Apache Spark pools in Synapse Analytics to update Delta Lake.
  - You can't store query results to storage in Delta Lake format by using the CETAS command. The CETAS command supports only Parquet and CSV as the output formats.
- Serverless SQL pools in Synapse Analytics don't support the datasets with the BLOOM filter. The serverless SQL pool ignores the BLOOM filters.
- Delta Lake support isn't available in dedicated SQL pools. Make sure that you use serverless SQL pools to query Delta Lake files.

JSON text isn't properly formatted

This error indicates that serverless SQL pool can't read the Delta Lake transaction log. You'll probably see the following error:

```
Msg 13609, Level 16, State 4, Line 1
JSON text is not properly formatted. Unexpected character '' is found at position 263934.
Msg 16513, Level 16, State 0, Line 1
Error reading external metadata.
```

Make sure that your Delta Lake dataset isn't corrupted. Verify that you can read the content of the Delta Lake

folder by using Apache Spark pool in Azure Synapse. This way you'll ensure that the `_delta_log` file isn't corrupted.

Workaround

Try to create a checkpoint on the Delta Lake dataset by using Apache Spark pool and rerun the query. The checkpoint aggregates transactional JSON log files and might solve the issue.

If the dataset is valid, create a support ticket and provide more information:

- Don't make any changes like adding or removing the columns or optimizing the table because this operation might change the state of the Delta Lake transaction log files.
- Copy the content of the `_delta_log` folder into a new empty folder. *Do not* copy the `.parquet data` files.
- Try to read the content that you copied in the new folder and verify that you're getting the same error.
- Send the content of the copied `_delta_log` file to Azure support.

Now you can continue using the Delta Lake folder with Spark pool. You'll provide copied data to Microsoft support if you're allowed to share this information. The Azure team will investigate the content of the `delta_log` file and provide more information about possible errors and workarounds.

Resolving Delta logs failed

The following error indicates that serverless SQL pool cannot resolve Delta logs:

```
Resolving Delta logs on path '%ls' failed with error: Cannot parse json object from log folder.
```

The most common cause is that `last_checkpoint_file` in `_delta_log` folder is larger than 200 bytes due to the `checkpointSchema` field added in Spark 3.3.

There are two options available to circumvent this error:

- Modify appropriate config in Spark notebook and generate a new checkpoint, so that `last_checkpoint_file` gets re-created. In case you are using Azure Databricks, the config modification is the following: `spark.conf.set("spark.databricks.delta.checkpointSchema.writeThresholdLength", 0);`
- Downgrade to Spark 3.2.1.

Our engineering team is currently working on a full support for Spark 3.3.

Performance

Serverless SQL pool assigns the resources to the queries based on the size of the dataset and query complexity. You can't change or limit the resources that are provided to the queries. There are some cases where you might experience unexpected query performance degradations and you might have to identify the root causes.

Query duration is very long

If you have queries with a query duration longer than 30 minutes, the query slowly returning results to the client are slow. Serverless SQL pool has a 30-minute limit for execution. Any more time is spent on result streaming. Try the following workarounds:

- If you use Synapse Studio, try to reproduce the issues with some other application like SQL Server Management Studio or Azure Data Studio.
- If your query is slow when executed by using SQL Server Management Studio, Azure Data Studio, Power BI, or some other application, check networking issues and best practices.
- Put the query in the CETAS command and measure the query duration. The CETAS command stores the results to Azure Data Lake Storage and doesn't depend on the client connection. If the CETAS command finishes faster than the original query, check the network bandwidth between the client and serverless SQL

pool.

Query is slow when executed by using Synapse Studio

If you use Synapse Studio, try using a desktop client such as SQL Server Management Studio or Azure Data Studio. Synapse Studio is a web client that connects to serverless SQL pool by using the HTTP protocol, which is generally slower than the native SQL connections used in SQL Server Management Studio or Azure Data Studio.

Query is slow when executed by using an application

Check the following issues if you experience slow query execution:

- Make sure that the client applications are collocated with the serverless SQL pool endpoint. Executing a query across the region can cause additional latency and slow streaming of result set.
- Make sure that you don't have networking issues that can cause the slow streaming of result set
- Make sure that the client application has enough resources (for example, not using 100% CPU).
- Make sure that the storage account or Cosmos DB analytical storage is placed in the same region as your serverless SQL endpoint.

See best practices for collocating the resources.

High variations in query durations

If you're executing the same query and observing variations in the query durations, several reasons might cause this behavior:

- Check if this is the first execution of a query. The first execution of a query collects the statistics required to create a plan. The statistics are collected by scanning the underlying files and might increase the query duration. In Synapse Studio, you'll see the "global statistics creation" queries in the SQL request list that are executed before your query.
- Statistics might expire after some time. Periodically, you might observe an impact on performance because the serverless pool must scan and rebuild the statistics. You might notice another "global statistics creation" queries in the SQL request list that are executed before your query.
- Check if there's some workload that's running on the same endpoint when you executed the query with the longer duration. The serverless SQL endpoint equally allocates the resources to all queries that are executed in parallel, and the query might be delayed.

Connections

Serverless SQL pool enables you to connect by using the TDS protocol and by using the T-SQL language to query data. Most of the tools that can connect to SQL Server or Azure SQL Database can also connect to serverless SQL pool.

SQL pool is warming up

Following a longer period of inactivity, serverless SQL pool will be deactivated. The activation happens automatically on the first next activity, such as the first connection attempt. The activation process might take a bit longer than a single connection attempt interval, so the error message is displayed. Retrying the connection attempt should be enough.

As a best practice, for the clients that support it, use ConnectionRetryCount and ConnectRetryInterval connection string keywords to control the reconnect behavior.

If the error message persists, file a support ticket through the Azure portal.

Can't connect from Synapse Studio

See the Synapse Studio section.

Can't connect to the Azure Synapse pool from a tool

Some tools might not have an explicit option that you can use to connect to the Azure Synapse serverless SQL

pool. Use an option that you would use to connect to SQL Server or SQL Database. The connection dialog doesn't need to be branded as "Synapse" because the serverless SQL pool uses the same protocol as SQL Server or SQL Database.

Even if a tool enables you to enter only a logical server name and predefines the `database.windows.net` domain, put the Azure Synapse workspace name followed by the `-ondemand` suffix and the `database.windows.net` domain.

Security

Make sure that a user has permissions to access databases, permissions to execute commands, and permissions to access Azure Data Lake or Azure Cosmos DB storage.

Can't access Azure Cosmos DB account

You must use a read-only Azure Cosmos DB key to access your analytical storage, so make sure that it didn't expire or that it isn't regenerated.

If you get the error "Resolving Azure Cosmos DB path has failed with error", make sure that you configured a firewall.

Can't access lakehouse or Spark database

If a user can't access a lakehouse or Spark database, the user might not have permission to access and read the database. A user with CONTROL SERVER permission should have full access to all databases. As a restricted permission, you might try to use CONNECT ANY DATABASE and SELECT ALL USER SECURABLES.

SQL user can't access Dataverse tables

Dataverse tables access storage by using the caller's Azure AD identity. A SQL user with high permissions might try to select data from a table, but the table wouldn't be able to access Dataverse data. This scenario isn't supported.

Azure AD service principal sign-in failures when SPI creates a role assignment

If you want to create a role assignment for a service principal identifier (SPI) or Azure AD app by using another SPI, or you've already created one and it fails to sign in, you'll probably receive the following error:

```
Login error: Login failed for user '<token-identified principal>'.
```

For service principals, sign-in should be created with an application ID as a security ID (SID) not with an object ID. There's a known limitation for service principals, which prevents Azure Synapse from fetching the application ID from Microsoft Graph when it creates a role assignment for another SPI or app.

Solution 1

Go to the **Azure portal** > **Synapse Studio** > **Manage** > **Access control** and manually add **Synapse Administrator** or **Synapse SQL Administrator** for the desired service principal.

Solution 2

You must manually create a proper sign-in with SQL code:

```
use master
go
CREATE LOGIN [<service_principal_name>] FROM EXTERNAL PROVIDER;
go
ALTER SERVER ROLE sysadmin ADD MEMBER [<service_principal_name>];
go
```

Solution 3

You can also set up a service principal Azure Synapse admin by using PowerShell. You must have the Az.Synapse module installed.

The solution is to use the cmdlet New-AzSynapseRoleAssignment with `-ObjectId "parameter"`. In that parameter field, provide the application ID instead of the object ID by using the workspace admin Azure service principal credentials.

PowerShell script:

```
$spAppId = "<app_id_which_is_already_an_admin_on_the_workspace>"
$SPPassword = "<application_secret>"
$tenantId = "<tenant_id>"
$secpasswd = ConvertTo-SecureString -String $SPPassword -AsPlainText -Force
$cred = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList $spAppId, $secpasswd

Connect-AzAccount -ServicePrincipal -Credential $cred -Tenant $tenantId

New-AzSynapseRoleAssignment -WorkspaceName "<workspaceName>" -RoleDefinitionName "Synapse Administrator" -
ObjectId "<app_id_to_add_as_admin>" [-Debug]
```

Validation

Connect to the serverless SQL endpoint and verify that the external sign-in with SID `app_id_to_add_as_admin` is created:

```
select name, convert(uniqueidentifier, sid) as sid, create_date
from sys.server_principals where type in ('E', 'X')
```

Or try to sign in on the serverless SQL endpoint by using the set admin app.

Constraints

Some general system constraints might affect your workload:

PROPERTY	LIMITATION
Maximum number of Azure Synapse workspaces per subscription	See limits.
Maximum number of databases per serverless pool	20 (not including databases synchronized from Apache Spark pool).
Maximum number of databases synchronized from Apache Spark pool	Not limited.
Maximum number of databases objects per database	The sum of the number of all objects in a database can't exceed 2,147,483,647. See Limitations in SQL Server database engine.
Maximum identifier length in characters	128. See Limitations in SQL Server database engine.
Maximum query duration	30 minutes.
Maximum size of the result set	Up to 400 GB shared between concurrent queries.

PROPERTY	LIMITATION
Maximum concurrency	Not limited and depends on the query complexity and amount of data scanned. One serverless SQL pool can concurrently handle 1,000 active sessions that are executing lightweight queries. The numbers will drop if the queries are more complex or scan a larger amount of data.

Can't create a database in serverless SQL pool

Serverless SQL pools have limitations, and you can't create more than 20 databases per workspace. If you need to separate objects and isolate them, use schemas.

If you get the error "CREATE DATABASE failed. User database limit has been already reached," you've created the maximum number of databases that are supported in one workspace.

You don't need to use separate databases to isolate data for different tenants. All data is stored externally on a data lake and Azure Cosmos DB. The metadata like table, views, and function definitions can be successfully isolated by using schemas. Schema-based isolation is also used in Spark where databases and schemas are the same concepts.

Query Azure data

Serverless SQL pools enable you to query data in Azure Storage or Azure Cosmos DB by using external tables and the OPENROWSET function. Make sure that you have proper permission set up on your storage.

Query CSV data

Learn how to query a single CSV file or folders and multiple CSV files. You can also query partitioned files

Query Parquet data

Learn how to query Parquet files with nested types. You can also query partitioned files.

Query Delta Lake

Learn how to query Delta Lake files with nested types.

Query Azure Cosmos DB data

Learn how to query Azure Cosmos DB analytical store. You can use an online generator to generate the WITH clause based on a sample Azure Cosmos DB document. You can create views on top of Azure Cosmos DB containers.

Query JSON data

Learn how to query JSON files. You can also query partitioned files.

Create views, tables, and other database objects

Learn how to create and use views and external tables or set up row-level security. If you have partitioned files, make sure you use partitioned views.

Copy and transform data (CETAS)

Learn how to store query results to storage by using the CETAS command.

Azure Synapse Analytics business intelligence partners

8/19/2022 • 5 minutes to read • Edit Online

To create your data warehouse solution, you can choose from different kinds of industry-leading tools. This article highlights Microsoft partners with business intelligence (BI) solutions officially supporting Azure Synapse Analytics.

Our business intelligence partners

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	AtScale AtScale provides a single, secured, and governed workspace for distributed data. AtScale's Cloud OLAP, Autonomous Data Engineering™, and Universal Semantic Layer™ powers business intelligence results for faster, more accurate business decisions.	Product page
	Birst Birst connects the entire organization through a network of interwoven virtualized BI instances on-top of a shared common analytical fabric	Product page
	Count Count is the next generation SQL editor, giving you the fastest way to explore and share your data with your team. At Count's core is a data notebook built for SQL, allowing you to structure your code, iterate quickly and stay in flow. Visualize your results instantly or customize them to build beautifully detailed charts in just a few clicks. Instantly share anything from one-off queries to full interactive data stories built off any of your Azure Synapse data sources.	Product page

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Dremio Analysts and data scientists can discover, explore and curate data using Dremio's intuitive UI, while IT maintains governance and security. Dremio makes it easy to join ADLS with Blob Storage, Azure SQL Database, Azure Synapse SQL, HDInsight, and more. With Dremio, Power BI analysts can search for new datasets stored on ADLS, immediately access that data in Power BI with no preparation by IT, create visualizations, and iteratively refine reports in real-time. And analysts can create new reports that combine data between ADLS and other databases.	Product page Azure Marketplace
	Dundas BI Dundas Data Visualization is a leading, global provider of Business Intelligence and Data Visualization software. Dundas dashboards, reporting, and visual data analytics provide seamless integration into business applications, enabling better decisions and faster insights.	Product page
	IBM Cognos Analytics Cognos Analytics includes self-service capabilities that make it simple, clear, and easy to use, whether you're an experienced business analyst examining a vast supply chain, or a marketer optimizing a campaign. Cognos Analytics uses AI and other capabilities to guide data exploration. It makes it easier for users to get the answers they need	Product page
	Information Builders (WebFOCUS) WebFOCUS business intelligence helps companies use data more strategically across and beyond the enterprise. It allows users and administrators to rapidly create dashboards that combine content from multiple data sources and formats. It also provides robust security and comprehensive governance that enables seamless and secure sharing of any BI and analytics content	Product page

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Jinfonet JReport JReport is an embeddable BI solution for the enterprise. The solution offers capabilities such as report creation, dashboards, and data analysis on cloud, big data, and transactional data sources. By visualizing data, you can conduct your own reporting and data discovery for agile, on-the-fly decision making.	Product page
	Logi Analytics Together, Logi Analytics enables your organization to collect, analyze, and immediately act on the largest and most diverse data sets in the world.	Product page
	Looker BI Looker gives everyone in your company the ability to explore and understand the data that drives your business. Looker also gives the data analyst a flexible and reusable modeling layer to control and curate that data. Companies have fundamentally transformed their culture using Looker as the catalyst.	Product page Azure Marketplace
	MicroStrategy The MicroStrategy platform offers a complete set of business intelligence and analytics capabilities that enable organizations to get value from their business data. MicroStrategy's powerful analytical engine, comprehensive toolsets, variety of data connectors, and open architecture ensures you have everything you need to extend access to analytics across every team.	Product page Azure Marketplace
	Mode Mode is a modern analytics and BI solution that helps teams make decisions through unreasonably fast and unexpectedly delightful data analysis. Data teams move faster through a preferred workflow that combines SQL, Python, R, and visual analysis, while stakeholders work alongside them exploring and sharing data on their own. With data more accessible to everyone, we shorten the distance from questions to answers and help businesses make better decisions, faster.	Product page

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
PYRAMID ANALYTICS	**Pyramid Analytics** Pyramid 2020 is the trusted analytics platform that connects your teams, drives confident decisions, and produces winning results. Business users can do high-end, cloud-scale analytics and data science without IT help — on any browser or device. Data scientists can take advantage of machine learning algorithms and scripting to understand difficult business problems. Power users can prepare and model their own data to create illuminating analytic content. Non-technical users can benefit from stunning visualizations and guided analytic presentations. It's the next generation of self-service analytics with governance.	Product page Azure Marketplace
Qlik	**Qlik Sense Enterprise** Drive insight discovery with the data visualization app that anyone can use. With Qlik Sense, everyone in your organization can easily create flexible, interactive visualizations and make meaningful decisions.	Product page Azure Marketplace
sas	**SAS® Viya®** SAS® Viya® is an AI, analytic, and data management solution running on a scalable, cloud-native architecture. It enables you to operationalize insights, empowering everyone – from data scientists to business users – to collaborate and realize innovative results faster. Using open source or SAS models, SAS® Viya® can be accessed through APIs or interactive interfaces to transform raw data into actions.	Product page Azure Marketplace
SISENSE	**SiSense** SiSense is a full-stack Business Intelligence software that comes with tools that a business needs to analyze and visualize data: a high-performance analytical database, the ability to join multiple sources, simple data extraction (ETL), and web-based data visualization. Start to analyze and visualize large data sets with SiSense BI and Analytics today.	Product page

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Tableau Tableau's self-service analytics help anyone see and understand their data, across many kinds of data from flat files to databases. Tableau has a native, optimized connector to Synapse SQL pool that supports both live data and in-memory analytics.	Product page Azure Marketplace
	Targit (Decision Suite) Targit Decision Suite provides a BI platform that delivers real-time dashboards, self-service analytics, user-friendly reporting, stunning mobile capabilities, and simple data-discovery technology. Everything in a single, cohesive solution. Targit gives companies the courage to act.	Product page Azure Marketplace
	ThoughtSpot Use search to get granular insights from billions of rows, or let AI uncover insights from questions you might not have thought about. ThoughtSpot helps businesspeople find insights hidden in their company data in seconds. Use search to analyze your data and get automated insights when you need them.	Product page
	Yellowfin Yellowfin is a top rated Cloud BI vendor for *ad hoc* Reporting and Dashboards by BARC; The BI Survey. Connect to a a dedicated SQL pool in Azure Synapse Analytics, then create and share beautiful reports and dashboards with award winning collaborative BI and location intelligence features.	Product page Azure Marketplace

Next Steps

To learn more about some of our other partners, see Data Integration partners, Data Management partners, and Machine Learning and AI partners.

See how to discover partner solutions through Synapse Studio.

Azure Synapse Analytics data integration partners

8/19/2022 • 12 minutes to read • Edit Online

To create your data warehouse solution using the dedicated SQL pool in Azure Synapse Analytics, you can choose from a wide variety of industry-leading tools. This article highlights Microsoft partner companies with official data integration solutions that support Azure Synapse.

Data integration partners

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Ab Initio Ab Initio's agile digital engineering platform helps you solve the toughest data processing and data management problems in corporate computing. Ab Initio's cloud-native platform lets you access and use data anywhere in your corporate ecosystem, whether in Azure or on-premises, including data stored on legacy systems. The combination of an intuitive interface with powerful automation, data quality, data governance, and active metadata capabilities enables rapid development and true data self-service, freeing analysts to do their jobs quickly and effectively. Join the world's largest businesses in using Ab Initio to turn big data into meaningful data.	Product page
	Aecorsoft AecorSoft offers fast, scalable, and real-time ELT/ETL software solution to help SAP customers bring complex SAP data to Azure Synapse Analytics and Azure data platform. With full compliance with SAP application layer security, AecorSoft solution is officially SAP Premium Certified to integrate with SAP applications. AecorSoft's unique Super Delta and Change-Data-Capture features enable SAP users to stream delta data from SAP transparent, pool, and cluster tables to Azure in CSV, Parquet, Avro, ORC, or GZIP format. Besides SAP tabular data, many other business-rule-heavy SAP objects like BW queries and S/4HANA CDS Views are fully supported.	Product page Azure Marketplace

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Alooma Alooma is an Extract, Transform, and Load (ETL) solution that enables data teams to integrate, enrich, and stream data from various data silos to an Azure Synapse data warehouse all in real time.	
	Alteryx Alteryx Designer provides a repeatable workflow for self-service data analytics that leads to deeper insights in hours, not the weeks typical of traditional approaches! Alteryx Designer helps data analysts by combining data preparation, data blending, and analytics – predictive, statistical, and spatial – using the same intuitive user interface.	Product page Azure Marketplace
	BI Builders (Xpert BI) Xpert BI helps organizations build and maintain a robust and scalable data platform in Azure faster through metadata-based automation. It extends Azure Synapse with best practices and DataOps, for agile data development with built-in data governance functionalities. Use Xpert BI to quickly test out and switch between different Azure solutions such as Azure Synapse, Azure Data Lake Storage, and Azure SQL Database, as your business and analytics needs changes and grows.	Product page Azure Marketplace
	BryteFlow With BryteFlow, you can continually replicate data from transactional sources like Oracle, SQL Server, SAP, MySQL, and more to Azure Synapse Analytics in real time, with best practices, and access reconciled data that is ready-to-use. BryteFlow extracts and replicates data in minutes using log-based Change Data Capture and merges deltas automatically to update data. It can be configured with times series as well. There's no coding for any process (just point and select!) and tables are created automatically on the destination. BryteFlow supports enterprise-scale automated data integration with extremely high throughput, ingesting terabytes of data, with smart partitioning, and multi-threaded, parallel loading.	Product page Azure Marketplace

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	CData Sync - Cloud Data Pipeline Build high-performance data pipelines for Microsoft Azure Synapse in minutes. CData Sync is an easy-to-use, go-anywhere ETL/ELT pipeline that streamlines data flow from more than 200+ enterprise data sources to Azure Synapse. With CData Sync, users can easily create automated continuous data replication between Accounting, CRM, ERP, Marketing Automation, On-Premises, and cloud data.	Product page Azure Marketplace
	Datometry Datometry Hyper-Q makes existing applications written for Teradata run natively on Azure Synapse. Datometry emulates commonly used Teradata SQL, including analytical SQL, and advanced operational concepts like stored procedures, macros, SET tables, and more. Because Hyper-Q returns results that are bit-identical to Teradata, existing applications can be replatformed to Azure Synapse without any significant modifications. With Datometry, enterprises can move to Azure rapidly and take full advantage of Synapse immediately.	Product page Azure Marketplace
	Denodo Denodo provide real-time access to data across an organization's diverse data sources. It uses data virtualization to bridge data across many sources without replication. Denodo offers broad access to structured and unstructured data residing in enterprise, big data, and cloud sources, in both batch and real time.	Product page Azure Marketplace
	Dimodelo Dimodelo Data Warehouse Studio is a data warehouse automation tool for the Azure data platform. Dimodelo enhances developer productivity through a dedicated data warehouse modeling and ETL design tool, pattern-based best practice code generation, one-click deployment, and ETL orchestration. Dimodelo enhances maintainability with change propagation, allows developers to stay focused on business outcomes, and automates portability across data platforms.	Product page Azure Marketplace

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Fivetran Fivetran helps you centralize data from disparate sources. It features a zero maintenance, zero configuration data pipeline product with a growing list of built-in connectors to all the popular data sources. Setup takes five minutes after authenticating to data sources and target data warehouse.	Product page Azure Marketplace
	HVR HVR provides a real-time cloud data replication solution that supports enterprise modernization efforts. The HVR platform is a reliable, secure, and scalable way to quickly and efficiently integrate large data volumes in complex environments, enabling real-time data updates, access, and analysis. Global market leaders in various industries trust HVR to address their real-time data integration challenges and revolutionize their businesses. HVR is a privately held company based in San Francisco, with offices across North America, Europe, and Asia.	Product page
	Incorta Incorta enables organizations to go from raw data to quickly discovering actionable insights in Azure by automating the various data preparation steps typically required to analyze complex data. which. Using a proprietary technology called Direct Data Mapping and Incorta's Blueprints (pre-built content library and best practices captured from real customer implementations), customers experience unprecedented speed and simplicity in accessing, organizing, and presenting data and insights for critical business decision-making.	Product page Azure Marketplace

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	1.Informatica Cloud Services for Azure Informatica Cloud offers a best-in-class solution for self-service data migration, integration, and management capabilities. Customers can quickly and reliably import, and export petabytes of data to Azure from different kinds of sources. Informatica Cloud Services for Azure provides native, high volume, high-performance connectivity to Azure Synapse, SQL Database, Blob Storage, Data Lake Store, and Azure Cosmos DB. **2.Informatica PowerCenter** PowerCenter is a metadata-driven data integration platform that jumpstarts and accelerates data integration projects to deliver data to the business more quickly than manual hand coding. It serves as the foundation for your data integration investments	**Informatica Cloud services for Azure** Product page Azure Marketplace **Informatica PowerCenter** Product page Azure Marketplace
	Information Builders (Omni-Gen Data Management) Information Builder's Omni-Gen data management platform provides data integration, data quality, and master data management solutions. It makes it easy to access, move, and blend all data no matter the format, location, volume, or latency.	Product page
	Loome Loome provides a unique governance workbench that seamlessly integrates with Azure Synapse. It allows you to quickly onboard your data to the cloud and load your entire data source into ADLS in Parquet format. You can orchestrate data pipelines across data engineering, data science and HPC workloads, including native integration with Azure Data Factory, Python, SQL, Synapse Spark, and Databricks. Loome allows you to easily monitor Data Quality exceptions reinforcing Synapse as your strategic Data Quality Hub. Loome keeps an audit trail of resolved issues, and proactively manages data quality with a fully automated data quality engine generating audience targeted alerts in real time.	Product page Azure Marketplace

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Lyftron Lyftron modern data hub combines an effortless data hub with agile access to data sources. Lyftron eliminates traditional ETL/ELT bottlenecks with automatic data pipeline and make data instantly accessible to BI user with the modern cloud compute of Azure Synapse, Spark & Snowflake. Lyftron connectors automatically convert any source into normalized, ready-to-query relational format and replication. It offers advanced security, data governance and transformation, with simple ANSI SQL along with search capability on your enterprise data catalog.	Product page Azure Marketplace
	Matillion Matillion is data transformation software for cloud data warehouses. Only Matillion is purpose-built for Azure Synapse enabling businesses to achieve new levels of simplicity, speed, scale, and savings. Matillion products are highly rated and trusted by companies of all sizes to meet their data integration and transformation needs. Learn more about how you can unlock the potential of your data with Matillion's cloud-based approach to data transformation.	Product page Azure Marketplace
	oh22 HEDDA.IO oh22's HEDDA.IO is a knowledge-driven data quality product built for Microsoft Azure. It enables you to build a knowledge base and use it to perform various critical data quality tasks, including correction, enrichment, and standardization of your data. HEDDA.IO also allows you to do data cleansing by using cloud-based reference data services provided by reference data providers or developed and provided by you.	Product page Azure Marketplace

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Precisely Precisely Connect ETL enables extract transfer and load (ETL) of data from multiple sources to Azure targets. Connect ETL is an easy to configure tool that doesn't require coding or tuning. ETL transformation can be done on the fly. It eliminates the need for costly database staging areas or manual pushes, allowing you to create your own data blends with consistent sustainable performance. Import legacy data from multiple sources including mainframe DB2, VSAM, IMS, Oracle, SQL Server, Teradata, and write them to cloud targets including Azure Databricks, Azure Synapse Analytics, and Azure Data Lake Storage. By using the high performance Connect ETL engine, you can expect optimal performance and consistency.	Product page Azure Marketplace
	Qlik Data Integration Qlik Data Integration provides an automated solution for loading data into an Azure Synapse. It simplifies batch loading and incremental replication of data from many sources: SQL Server, Oracle, DB2, Sybase, MySQL, and more.	Product page Azure Marketplace
	Qubole Qubole provides a cloud-native platform that enables users to conduct ETL, analytics, and AI/ML workloads. It supports different kinds of open-source engines - Apache Spark, TensorFlow, Presto, Airflow, Hadoop, Hive, and more. It provides easy-to-use end-user tools for data processing from SQL query tools, to notebooks, and dashboards that use powerful open-source engines.	Product page
	SAS® Viya® SAS® Viya® is an AI, analytic, and data management solution running on a scalable, cloud-native architecture. It enables you to operationalize insights, empowering everyone – from data scientists to business users – to collaborate and realize innovative results faster. Using open source or SAS models, SAS® Viya® can be accessed through APIs or interactive interfaces to transform raw data into actions.	Product page Azure Marketplace

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Segment Segment is a data management and analytics solution that helps you make sense of customer data coming from various sources. It allows you to connect your data to over 200 tools to create better decisions, products, and experiences. Segment will transform and load multiple data sources into your warehouse for you using its built-in data connectors	Product page
	Skyvia (data integration) Skyvia data integration provides a wizard that automates data imports. This wizard allows you to migrate data between different kinds of sources - CRMs, application database, CSV files, and more.	Product page
	SnapLogic The SnapLogic Platform enables customers to quickly transfer data into and out of an Azure Synapse data warehouse. It offers the ability to integrate hundreds of applications, services, and IoT scenarios in one solution.	Product page Azure Marketplace
	SnowMirror by GuideVision SnowMirror is a smart data replication tool for ServiceNow. It loads data from a ServiceNow instance and stores it in an on-premise or cloud database. You can then use your replicated data for custom reporting and dashboards with tools like Power BI. Because your data is replicated, it reduces load on your ServiceNow cloud instance. It can be used for system integration, disaster recovery and more. SnowMirror can be used either on premises or in the cloud, and is compatible with all leading databases, including Microsoft SQL Server and Azure Synapse.	Product page

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
StreamSets	**StreamSets** StreamSets provides a data integration platform for DataOps. It operationalizes the full design-deploy-operate lifecycle of integrating data into an Azure Synapse data warehouse. You can quickly ingest and integrate data to and from the warehouse via streaming, batch, or changed data capture. Also, you can ensure continuous operations with smart data pipelines that provide end-to-end data flow visibility and resiliency.	Product page
talend	**Talend Cloud** Talend Cloud is an enterprise data integration platform to connect, access, and transform any data across the cloud or on-premises. It's an integration platform-as-a-service that provides broad connectivity, built-in data quality, and native support for the latest big data and cloud technologies.	Product page Azure Marketplace
THEOBALD SOFTWARE	**Theobald Software** Theobald Software has been offering various solutions for data integration with SAP since 2004. Secure, stable, fast and, if required, incremental access to all types of SAP data objects on SAP ERP, S/4, BW or BW/4 systems is their area of expertise; an expertise that has been officially certified by SAP and which more than 3,500 global customers are making use of. Their products, Xtract IS for Azure and Xtract Universal, are constantly improving and have evolved into SAP ETL/ELT solutions that seamlessly integrate with Microsoft Azure, where Synapse and Data Factory pipelines can be used to orchestrate SAP data extractions, while Azure Storage serves as a destination for SAP data ingestions.	Product page Azure Marketplace

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
TIMEXTENDER	**TimeXtender** TimeXtender's Discovery Hub helps companies build a modern data estate by providing an integrated data management platform that accelerates time to data insights by up to 10 times. Going beyond everyday ETL and ELT, it provides capabilities for data access, data modeling, and compliance in a single platform. Discovery Hub provides a cohesive data fabric for cloud scale analytics. It allows you to connect and integrate various data silos, catalog, model, move, and document data for analytics and AI.	Product page Azure Marketplace
TRIFACTA	**Trifacta Wrangler** Trifacta helps individuals and organizations explore, and join together diverse data for analysis. Trifacta Wrangler is designed to handle data wrangling workloads that need to support data at scale and a large number of end users.	Product page Azure Marketplace
WhereScape®	**Wherescape RED** WhereScape RED is an IDE that provides teams with automation tools to streamline ETL workflows. The IDE provides best practice, optimized native code for popular data targets. Use WhereScape RED to cut the time to develop, deploy, and operate your data infrastructure.	Product page Azure Marketplace
Xplenty	**Xplenty** Xplenty ELT platform lets you quickly and easily prepare your data for analytics and production use cases using a simple cloud service. Xplenty's point & select, drag & drop interface enables data integration, processing and preparation without installing, deploying, or maintaining any software. Connect and integrate with a wide set of data repositories and SaaS applications including Azure Synapse, Azure blob storage, and SQL Server. Xplenty also supports all Web Services that are accessible via REST API.	Product page

Next steps

To learn more about other partners, see Business Intelligence partners, Data Management partners, and Machine Learning and AI partners.

See how to discover partner solutions through Synapse Studio.

Azure Synapse Analytics data management partners

8/19/2022 • 4 minutes to read • Edit Online

This article highlights Microsoft partner companies with data management tools and solutions supporting Azure Synapse.

Data management partners

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Aginity Aginity is an analytics development tool. It puts the full power of Microsoft's Synapse platform in the hands of analysts and engineers. The rich and intuitive SQL development environment allows team members to connect to over a dozen industry leading analytics platforms. It allows users to ingest data in a variety of formats, and quickly build complex business calculation to serve the results into Business Intelligence and Machine Learning use cases. The entire application is built around a central catalog which makes collaboration across the analytics team a reality, and the sophisticated management capabilities and fine grained security make governance a breeze.	Product page
	Alation Alation's data catalog dramatically improves the productivity, increases the accuracy, and drives confident data-driven decision making for analysts. Alation's data catalog empowers everyone in your organization to find, understand, and govern data.	Product page

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	BI Builders (Xpert BI) Xpert BI provides an intuitive and searchable catalog for the line-of-business user to find, trust, and understand data and reports. The solution covers the whole data platform including Azure Synapse Analytics, ADLS Gen 2, Azure SQL Database, Analysis Services and Power BI, and also data flows and data movement end-to-end. Data stewards can update descriptions and tag data to follow regulatory requirements. Xpert BI can be integrated via APIs to other catalogs such as Microsoft Purview. It supplements traditional data catalogs with a business user perspective.	Product page Azure Marketplace
	Coffing Data Warehousing Coffing Data Warehousing provides Nexus Chameleon, a tool with 10 years of design dedicated to querying systems. Nexus is available as a query tool for dedicated SQL pool in Azure Synapse Analytics. Use Nexus to query in-house and cloud computers and join data across different platforms. Point-Click-Report!	Product page
	Inbrein MicroERD Inbrein MicroERD provides the tools that you need to create a precise data model, reduce data redundancy, improve productivity, and observe standards. By using its UI, which was developed based on extensive user experiences, a modeler can work on DB models easily and conveniently. You can continuously enjoy new and improved functions of MicroERD through prompt functional improvements and updates.	Product page
	Infolibrarian (Metadata Management Server) InfoLibrarian catalogs, stores, and manages metadata to help you solve key pain points of data management. Infolibrarian provides metadata management, data governance, and asset management solutions for managing and publishing metadata from a diverse set of tools and technologies.	Product page Azure Marketplace

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Kyligence Founded by the creators of Apache Kylin, Kyligence is on a mission to accelerate the productivity of its customers by automating data management, discovery, interaction, and insight generation – all without barriers. Kyligence Cloud enables cluster deployment, enhances data access, and dramatically accelerates data analysis. Kyligence's AI-augmented Big Data analytics management platform makes the often-challenging task of building enterprise-scale data lakes fast and easy.	Product page Azure Marketplace
	RedPoint Data Management RedPoint Data Management enables marketers to apply all their data to drive cross-channel customer engagement while doing structured and unstructured data management. With RedPoint, you can maximize the value of your structured and unstructured data to deliver the hyper-personalized, contextual interactions needed to engage today's omni-channel customer. Drag-and-drop interface makes designing and executing data management processes easy.	Product page Azure Marketplace
	SAS® Viya® SAS® Viya® is an AI, analytic, and data management solution running on a scalable, cloud-native architecture. It enables you to operationalize insights, empowering everyone – from data scientists to business users – to collaborate and realize innovative results faster. Using open source or SAS models, SAS® Viya® can be accessed through APIs or interactive interfaces to transform raw data into actions.	Product page Azure Marketplace
	SentryOne (DW Sentry) With the intelligent data movement dashboard and event calendar, you always know exactly what is impacting your workload. Designed to give you visibility into your queries and jobs running to load, backup, or restore your data, never worry about making the most of your Azure resources.	Product page Azure Marketplace

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	SqlDBM SqlDBM is a Cloud-based Data Modeling Tool that offers you an easy, convenient way to develop your database anywhere on any browser. All while incorporating any needed database rules and objects such as database keys, schemas, indexes, column constraints, and relationships.	Product page Azure Marketplace
	Tamr With Tamr, organizations can supply Azure Synapse with mastered data, allowing them to get most from Azure Synapse's analytic capabilities. Tamr's cloud-native data mastering solutions use machine learning to do the heavy lifting to combine, cleanse, and categorize data, with intuitive human feedback workflows to bridge the gap between data and business outcomes. Tamr integrates with Azure's data services including Azure Synapse Analytics, Azure Databricks, Azure HDInsight, Azure Data Catalog, Azure Data Lake Storage, and Azure Data Factory. It allows for data mastering at scale with a lower total cost of ownership, by taking advantage of the flexibility and scale of Azure.	Product page Azure Marketplace
	Teleran Teleran's Query Control prevents inappropriate and poorly formed queries from reaching Synapse and wasting compute resources. It sends intelligent messages to analytics users guiding them to more efficiently interact with the data. The goal is to ensure good business results without needlessly driving up Azure costs. Teleran Usage Analysis delivers an analysis of user, application, query, and data usage activity. It allows you to always have the entire picture of what's going on. It enables you to improve service, increase business productivity, and optimize Synapse consumption costs.	Product page

Next steps

To learn more about other partners, see Business Intelligence partners, Data Integration partners, and Machine Learning and AI partners.

Azure Synapse Analytics machine learning and AI partners

8/19/2022 • 2 minutes to read • Edit Online

This article highlights Microsoft partners with machine learning and artificial intelligence tools and solutions supporting Azure Synapse Analytics.

Machine learning and AI partners

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Dataiku Dataiku is the centralized data platform that moves businesses along their data journey from analytics at scale to Enterprise AI, powering self-service analytics while also ensuring the operationalization of machine learning models in production.	Product page Azure Marketplace
	Matlab MATLAB® is a programming platform designed for engineers and scientists. It combines a desktop environment tuned for iterative analysis and design processes with a programming language that expresses matrix and array mathematics directly. Millions worldwide use MATLAB for a range of applications, including machine learning, deep learning, signal and image processing, control systems, and computational finance.	Product page Azure Marketplace
	Qubole Qubole provides a cloud-native platform that enables users to conduct ETL, analytics, and AI/ML workloads. It supports different kinds of open-source engines - Apache Spark, TensorFlow, Presto, Airflow, Hadoop, Hive, and more. It provides easy-to-use end-user tools for data processing from SQL query tools, to notebooks, and dashboards that use powerful open-source engines.	Product page

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	SAS® Viya® SAS® Viya® is an AI, analytic, and data management solution running on a scalable, cloud-native architecture. It enables you to operationalize insights, empowering everyone – from data scientists to business users – to collaborate and realize innovative results faster. Using open source or SAS models, SAS® Viya® can be accessed through APIs or interactive interfaces to transform raw data into actions.	Product page Azure Marketplace

Next steps

To learn more about other partners, see Business Intelligence partners, Data Integration partners, and Data Management partners.

Azure Synapse Analytics system integration partners

8/19/2022 • 2 minutes to read • Edit Online

This article highlights Microsoft system integration partner companies building solutions with Azure Synapse.

System Integration partners

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
accenture	**Accenture** Bringing together 45,000+ dedicated professionals, the Accenture Microsoft Business Group—powered by Avanade—helps enterprises to thrive in the era of digital disruption.	Partner page
adatis	**Adatis** Adatis offers services that specialize in advanced data analytics, from data strategy and consultancy, to world class delivery and managed services.	Partner page
BLUEGRANITE DATA • INSIGHTS • ANALYTICS	**Blue Granite** The BlueGranite Catalyst for Analytics is an engagement approach that features their "think big, but start small" philosophy. Starting with collaborative envisioning and strategy sessions, Blue Granite work with clients to discover, create, and realize the value of new modern data and analytics solutions, using the latest technologies on the Microsoft platform.	Partner page
capax|global	**Capax Global** We improve your business by making better use of information you already have. Building custom solutions that align to your business goals, and setting you up for long-term success. We combine well-established patterns and practices with technology while using our team's wide range of industry and commercial software development experience. We share a passion for technology, innovation, and client satisfaction. Our pride for what we do drives the success of our projects and is fundamental to why people partner with us.	Partner page

PARTNER	DESCRIPTION	WEBSITE/PRODUCT LINK
	Coeo Coeo's team includes cloud consultants with deep expertise in Azure databases, and BI consultants dedicated to providing flexible and scalable analytic solutions. Coeo can help you move to a hybrid or full Azure solution.	Partner page
	Cognizant As a Microsoft strategic partner, Cognizant has the consulting skills and experience to help customers make the journey to the cloud. For each client project, Cognizant uses its strong partnership with Microsoft to maximize customer benefits from the Azure architecture.	Partner page
	Neal Analytics Neal Analytics helps companies navigate their digital transformation journey in converting data into valuable assets and a competitive advantage. With our machine learning and data engineering expertise, we use data to drive margin increases and profitable analytics projects. Comprised of consultants specializing in Data Science, Business Intelligence, Cognitive Services, practical AI, Data Management, and IoT, Neal Analytics is trusted to solve unique business problems and optimize operations across industries.	Partner page
	Pragmatic Works Pragmatic Works can help you capitalize on the value of your data by empowering more users and applications on the same dataset. We kickstart, accelerate, and maintain your cloud environment with a range of solutions that fit your business needs.	Partner page

Next Steps

To learn more about some of our other partners, see Business intelligence partners, Data Integration partners, Data Management partners, and also Machine Learning & AI partners.

Compatibility issues with third-party applications and Azure Synapse Analytics

8/19/2022 • 2 minutes to read • Edit Online

Applications built for SQL Server will seamlessly work with Azure Synapse dedicated SQL pools. In some cases, however, features and language elements that are commonly used in SQL Server may not be available in Azure Synapse, or they may behave differently.

This article lists known issues you may come across when using third-party applications with Azure Synapse Analytics.

Tableau error: "An attempt to complete a transaction has failed. No corresponding transaction found"

Starting from Azure Synapse dedicated SQL pool version 10.0.11038.0, some Tableau queries making stored procedure calls may fail with the following error message: "**[Microsoft][ODBC Driver 17 for SQL Server][SQL Server]111214; An attempt to complete a transaction has failed. No corresponding transaction found.**"

Cause

This is an issue in Azure Synapse dedicated SQL pool caused by the introduction of new system stored procedures that are called automatically by the ODBC and JDBC drivers. One of those system stored procedures can cause open transactions to be aborted if they fail execution. This issue can happen depending on the client application logic.

Solution

Customers seeing this particular issue when using Tableau connected to Azure Synapse dedicated SQL pools should set FMTONLY to YES in the SQL connection. For Tableau Desktop and Tableau Server, you should use a Tableau Data source Customization (TDC) file to ensure Tableau passes this parameter to the driver.

> **NOTE**
> Microsoft does not provide support for third-party tools. While we have tested that this solution works with Tableau Desktop 2020.3.2, you should use this workaround on your own capacity.

- To learn how to make global customizations with a TDC file on Tableau Desktop, refer to Tableau Desktop documentation.
- To learn how to make global customizations with a TDC file on Tableau Server, refer to Using a .TDC File with Tableau Server.

The example below shows a Tableau TDC file that passes the FMTONLY=YES parameter to the SQL connection string:

```
<connection-customization class='azure_sql_dw' enabled='true' version='18.1'>
 <vendor name='azure_sql_dw' />
 <driver name='azure_sql_dw' />
 <customizations>
        <customization name='odbc-connect-string-extras' value='UseFMTONLY=yes' />
 </customizations>
</connection-customization>
```

For more details about using TDC files, contact Tableau support.

See also

- T-SQL language elements for dedicated SQL pool in Azure Synapse Analytics.
- T-SQL statements supported for dedicated SQL pool in Azure Synapse Analytics.